MB88F33x ‘Indigo2(-x)’

Hardware Manual

Fujitsu Semiconductor Europe GmbH

Rev1-20

June 18, 2014

Attached files

Preface

Intention and Target Audience of this Document

This document describes and gives you detailed insight to the stated Fujitsu semiconductor product.

The MB88F33x ‘Indigo2(-x)’ device belongs to the Indigo Family used for graphics applications.

The target audience of this document is engineers developing products which will use the MB88F33x ‘Indigo2(-x)’ devices. It describes the function and operation of the devices. Please read this document carefully.

Trademarks

Licenses

Supplementary Documentation

For additional technical documentation, please visit out Website. (http://www.fujitsu.com/emea/services/microelectronics/gdc/gdcdevices/mb88f334-indigo2.html)

- APIX FIR Setup
- APIX PCB-Design Guideline
- Device Setup and Fujitsu Developer Suite
- Host Interface and Level Shifter Handling
- Using Bootmode 2&3
- External Flash
- DFE Setup

For additional application notes that handle the use of the APIX® interface, please visit Inova Semiconductors GmbH Website (Inova Semiconductors GmbH).

Attached Files

- pinning.xlsx
- SetupTools_Iris-MVL.xls
- Erase_Sector_3.txt
- Write_16Bit_Sector.txt
- Write_32Bit_Sector.txt

History

Revision	Date	Author	Description
0-01	03.08.2012	AvT/RvR	Preliminary first version
0-02	12.10.2012	Avt/RvR	Revised and extended version
0-03	25.10.2012	AvT/RvR	Revised and updated: - Section 1.7. Unused pins added - Section 2.3. Global Control Register added - Section 7.5.11. Low voltage Detection added - Figure 7-10 updated - Table 7-10 updated
0-04	13.12.2012	AvT/RvR	Revised and updated: - Section 1.9. Figures updated - Section Interrupt Table added - Section 3.2.3. Related Pins added - Section 7.6.3. Configuration pins added
0-05	18.01.2013	AvT/RvR	The following chapters has been revised and updated: - APIX - USART/LIN - Programmable CRC - External Interrupt Controller
0-06	26.02.2013	RvR	Revised and updated: - Figures 1-9, 1-10. 1-11 updated. - Section System Watchdog - Functional Description updated. - Section command sequencer watchdog updated. - Sections Watchdog reset and watchdog set-up updated - Section Power-up updated
0-07	26.03.2013	RvR	Revised and updated: - Section AShell Messages Overview - notations revised - Section Event Messages - Example added, Register names updated. - AShell Remote Handler Register Overview table - Register description updated. Note added - AShell Remote Handler Operation - AShell Downstream lock/Unlock Write, Note added. - Iris-MVL - Display buffer: Restrictions added to sections Buffer format, Scan Directions and Simple Scaling - Section IO Circuits added - Figure 7-10: Supply Power on Sequence updated.
0-08	17.05.2013	RvR	Revised and updated: - AXI-Flash Interface chapter extended - Section 7.6.11 SMC Outputs added
0-09	02.07.2013	RvR	Revised and updated: - Figure 2-2: Clock structure - Table 2-7: Boostrap setting - CFG-4 Function - Interrupt table added. - ConfigFIFO - Max. size for one channel added. - APIX2 MII - Half duplex operation not supported. - APIX2 MII - Section MII Multiplexer added. - Section E2IP Dynamic Re-Configuration revised. - Section “Daisy-chain Operation” revised. - Clock Setup of Iris-MVL - Figures updated - Power Consumption - Section updated. - IO Circuits - Revised and updated. New type (IN50) added. - AC Limits - Section ADC added. - Iris MVL - Dual channel operation setting added. - Figures 3-33, and 3-36 updated. - Section 2.4.5 Clock Synthesis - Note added.

Revision	Date	Author	Description
1-00	12.11.2013	RvR	Completely revised and updated. - Section “2.4.5 Clock Synthesis” - “Example:”updated. - Section “2.4.7 Internal Display PLL” added. - Section “2.9 Software Interface” updated. - Section “3.3 Bus Matrix - Address Map” and Figure 3-16 rectified. - Table 5-4 in section Configuring Display Output Pins (Multiplexing) updated. - Section “Writing 8bit/16bit Register” added - Section “6.8.3 Operation of Programmable Pulse Generator” updated. - Table 6-32 in section “6.9.4.3 Trigger Input/Output” updated. - “Chapter 7: Electrical Characteristics” completely revised. Figures and tables updated.
1-10	14.03.2014	RvR	Revised and updated: - Table 2-8, “Interrupts” rectified. - Figure 3-7, “Host Interface (clock timing and phase)” rectified. - Table 4-7, “Event messages” rectified. - Register names in sections “6.5.3.1 PWM Generation” and “6.5.3.5 Sound Generator Output Generation Logic” rectified. - Table 6-36, “Clock sources for CSL0/1/2 bit settings” in section “Reload Timer Additional Register Information” corrected. - For the IO circuit type “DISP_D” in Table 7-6, “IO circuit types” , two additional tables for “Differential mode” added.
1-11	20.03.2014	RvR	Revised and updated: - Figure 3-7, “Host Interface (clock timing and phase)” rectified. - Max. pic_clk for RSDS single channel (Table 5-1) rectified. - Added Differential Mode table in IO circuit type “DISP_D” - Table 7-6, “IO circuit types”” rectified.
1-20	18.06.2014	RvR	Revised and updated. Major changes: - Note added to section “4.7.5.4 Power-Up Initialization” - Figure 7-10, “Clock Input” updated - Table 7-15, “Clock Input” updated - Table 7-6, “IO circuit types” updated - Section “7.11 FLASH Memory Program/Erase Characteristics” added

Table of Contents

Table of Contents

Section Page

Table of Contents

Table of Contents

Section Page

Table of Contents

Section Page

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Section Page

Table of Contents

Table of Contents

Table of Contents

Chapter 1: Overview

NOTE The content of this document is subject to changes without prior warning. Significant changes to the last version are marked with revision bars.

1.1 General

The Indigo2 family devices (collectively referred to as MB88F33x ‘Indigo2(-x)’ in this document) is a family of graphics controllers, designed for remote display applications in the automotive industry. They are optimized to work together with MB86R12 ’Emerald-P’, MB86R91 ’ApCo’, and INAP37x von Inova Semiconductors GmbH to control a dashboard display, Head-up-Display (HUD) systems and a Central Information Display (CID). In addition, the MB88F33x ‘Indigo2(-x)’ Display Controller can be used to enable APIX2 (APIX version 2.0) based display systems in multiple applications in the automotive and industrial market segments.

The differences between the Indigo2 family devices are documented here. Summarizing, the following statements can be made:

- **MB88F334 ‘Indigo2’**: This is the main, fully-featured device.
- **MB88F335 ‘Indigo2-S’**: This is a scaled-down version of the MB88F33x ‘Indigo2(-x)’ device. Various features have been stripped out or reduced for price competitiveness. The maximum pixel clock for this device is 35 MHz.
- **MB88F336 ‘Indigo2-N’**: This is, basically, the **MB88F334 ‘Indigo2’** device, but without HDCP functionality at the APIX link.

1.2 Features

- Technology
  - CMOS 90nm (CU100F)
  - Power Supply Voltages:
    3.3 V $\rightarrow$ I/O Display Interface
    5.0 V (or 3.3 V) $\rightarrow$ I/O Peripherals
    5.0 V $\rightarrow$ Stepper motor
    1.2 V $\rightarrow$ Internal
- Package
  - QFP208
  - Ambient temperature range: -40°C...+105°C
- System Features
  - 160 MHz System Clock
  - Embedded flash Memory with ECC
    - 32kB in **MB88F334 ‘Indigo2’**, **MB88F335 ‘Indigo2-S’**, and **MB88F336 ‘Indigo2-N’**
  - Embedded SRAM
    - 64kB in **MB88F334 ‘Indigo2’**, **MB88F335 ‘Indigo2-S’**, and **MB88F336 ‘Indigo2-N’**
  - CPU/MCU/HOST Interface: Synchronous Serial Peripheral Interface (SPI), Automotive shell (AShell) sideband communication/link
  - Command Sequencer
  - DMA controller
  - Touch controller support (hardware accelerated communication with external touch devices) **(MB88F335 ‘Indigo2-S’ $\rightarrow$ not implemented)**

  - Configuration FIFO (to decouple host command stream and generate isochronous reconfiguration with internal peripherals)
  - High-Speed (quad) mode SPI for connection to external SPI flash **(MB88F335 'Indigo2-S' → not implemented)**
  - Spread spectrum clock modulation
  - Watchdog, alive sender, low voltage detection
  - CRC checksum calculation for memory content
- APIX2 features
  - RX interface
    - Up to 3 GBit/s **(MB88F335 'Indigo2-S' → 1 GBits/s)**
    - APIX1 mode compatible
    - HDCP for video link **(MB88F335 'Indigo2-S'/MB88F336 'Indigo2-N' → not implemented)**
  - Daisy chain output, i.e., connect through of the high-speed downstream APIX signal from the RX input to an additional APIX compatible TX interface, as well as the APIX upstream channel in the other direction **(MB88F335 'Indigo2-S' → not implemented)**
  - Sideband link
    - AShell remote handler
    - MII Interface/Ethernet over APIX **(MB88F335 'Indigo2-S' → not implemented)**
    - I2S output **(MB88F335 'Indigo2-S' → not implemented)**
- Graphics features
  - Integrated Pixel Engine (as opposed to Sprite Engine in the other Indigo family devices)
  - Maximum pixel frequencies supported up to 144MHz (eg. 1600x600 @100Hz, 1920x768 @60Hz)
    (MB88F335 'Indigo2-S' → max. 35 MHz pixel clock = e.g., 960x 480@60 Hz)
  - Display of run length encoded (RLE) background image (on-the-fly decoding)
  - Display of icons with 1, 2, 4, 8bpp (indirect, i.e., color palette) or 16bpp, 24bpp (direct) color depth. Icon size up to 2048x2048 pixel, depending on internal memory available.
  - Icon on top of APIX video stream or on top of run length encoded background
  - Flicker-free/seamless switch between an APIX video stream and a background video stream
  - Dither and gamma unit
  - Four signature units and each can compute a value for a display output frame to be compared against a pre-computed reference in order to detect corrupted data.
  - Connection to displays with
    - RSDS interface using a TCON with single or double 18 bpp or 24 bpp mode **(MB88F335 'Indigo2-S' → double mode not implemented)**
    - TTL interface with single 18 bpp or 24 bpp mode (support of data inversion for low EMI) or
    - LVDS single mode (24bit or 18bit per pixel, balanced or unbalanced) up to 75MHz pixel frequency **(MB88F335 'Indigo2-S' → not implemented)**
    - LVDS dual mode (2x24bit or 2x18bit per pixel, balanced or unbalanced) up to 144MHz pixel frequency **(MB88F335 'Indigo2-S' → not implemented)**
- Peripherals
  - 6x stepper motor controllers
  - 16 channel ADC+ 12 for Zero Point Detection (ZPD)
  - 2x I^2C
  - 1x USART or 1x LIN

- SPI interface for up to 4 target devices (only one can be simultaneously served)
- Sound capability I2S via APIX **(MB88F335 ‘Indigo2-S’ → not implemented)** and internal sound generator
- 16 x PWMs (Pulse Width Modulation)
- Max. 110 GPIOs (General Purpose I/Os). This is the maximum count when all I/O pins are switched to GPIO functionality.
- 8x External Interrupts

1.3 Block Diagrams

1.3.1 MB88F334 ‘Indigo2’

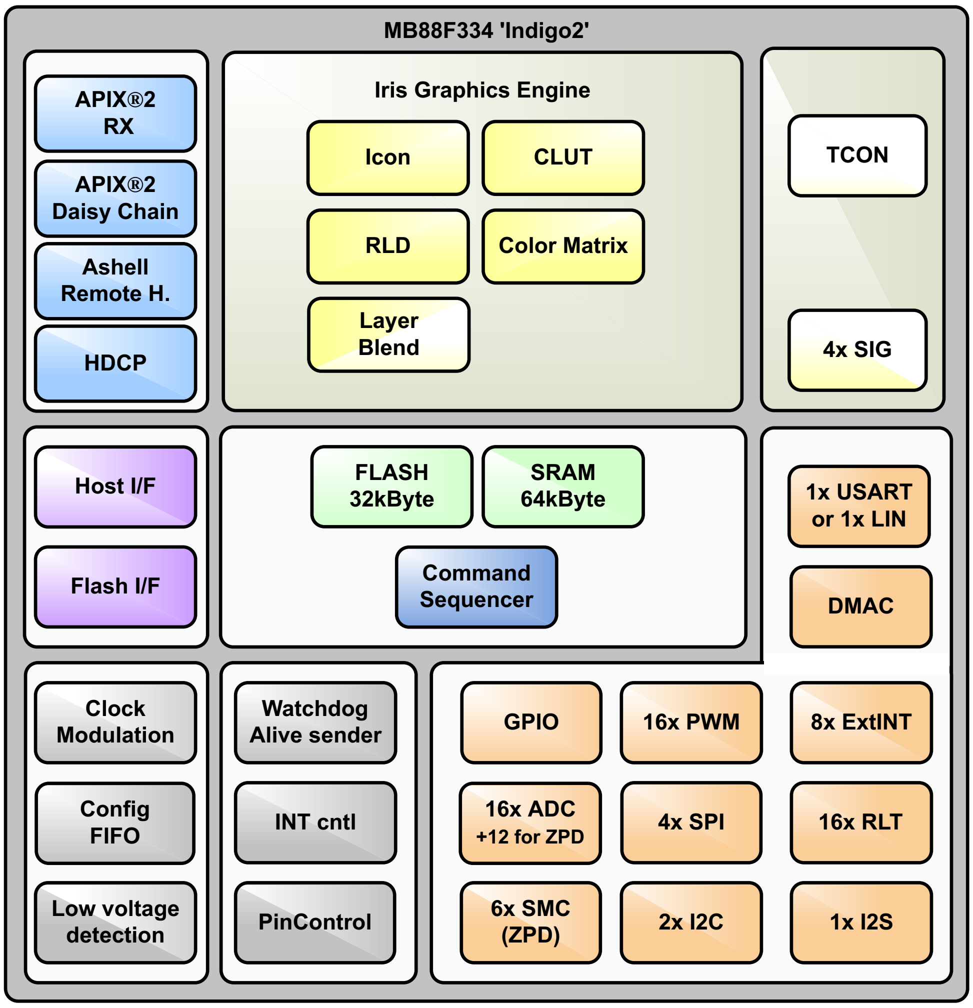


Figure 1-1: Block Diagram **MB88F334 ‘Indigo2’**

1.3.2 MB88F335 ‘Indigo2-S’

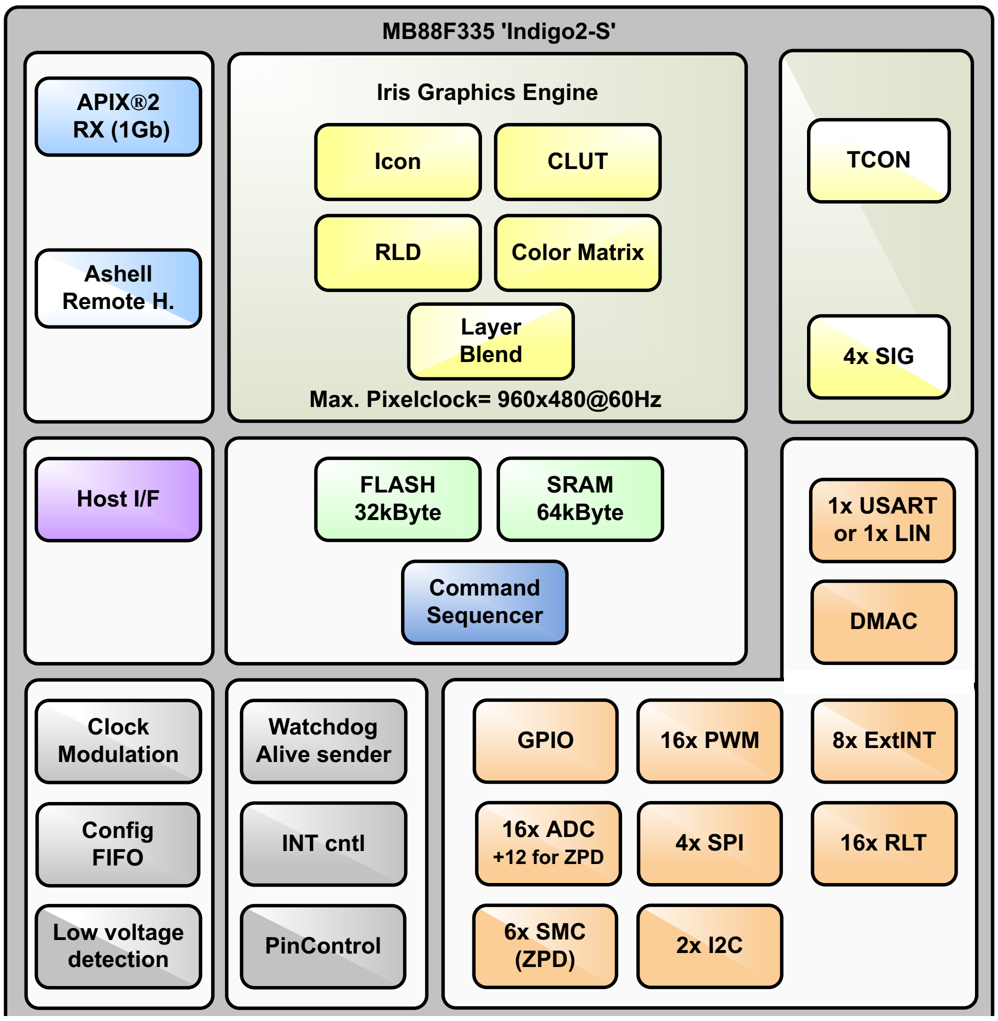


Figure 1-2: Block Diagram **MB88F335 ‘Indigo2-S’**

1.3.3 MB88F336 ‘Indigo2-N’

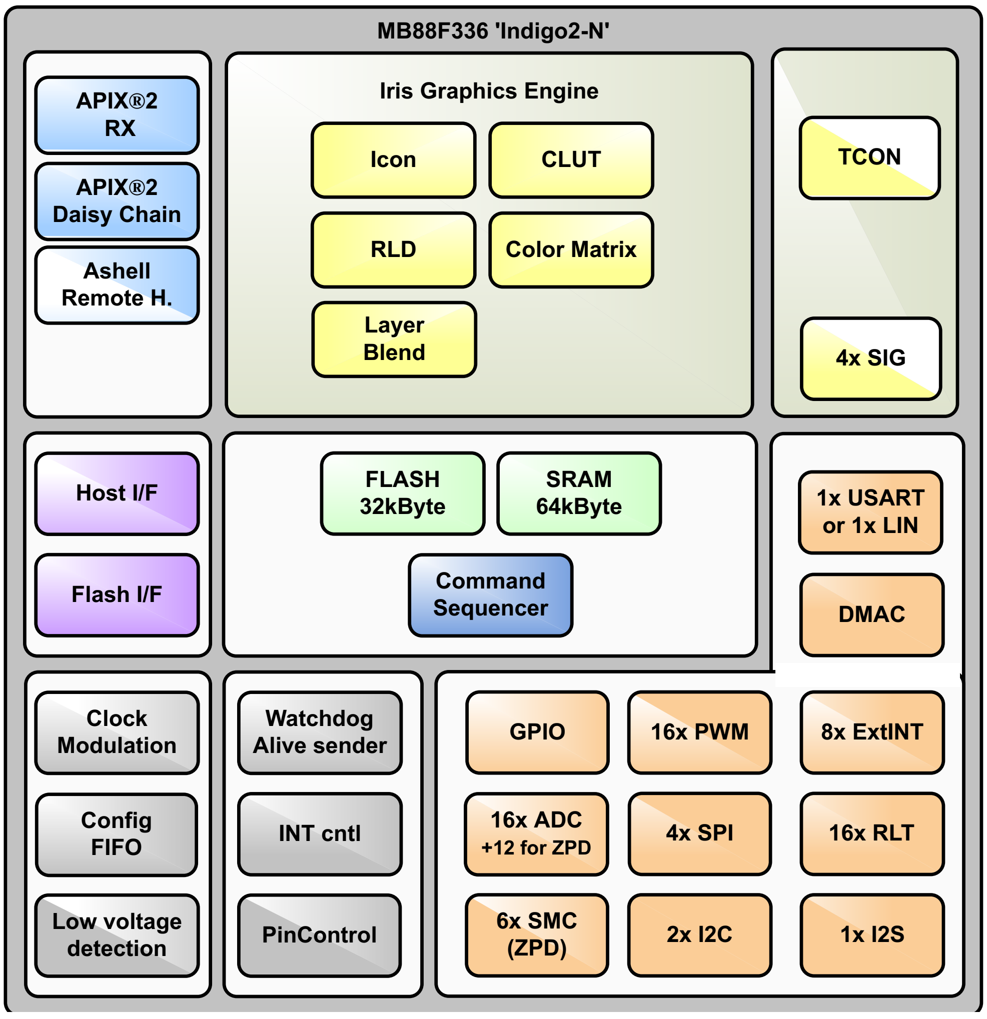
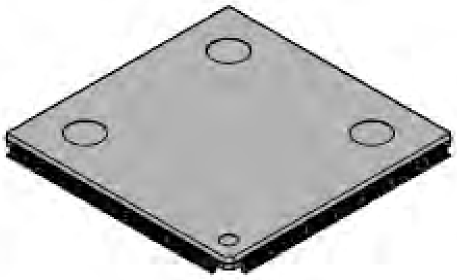


Figure 1-3: Block Diagram **MB88F336 ‘Indigo2-N’**

Figure 1-4:

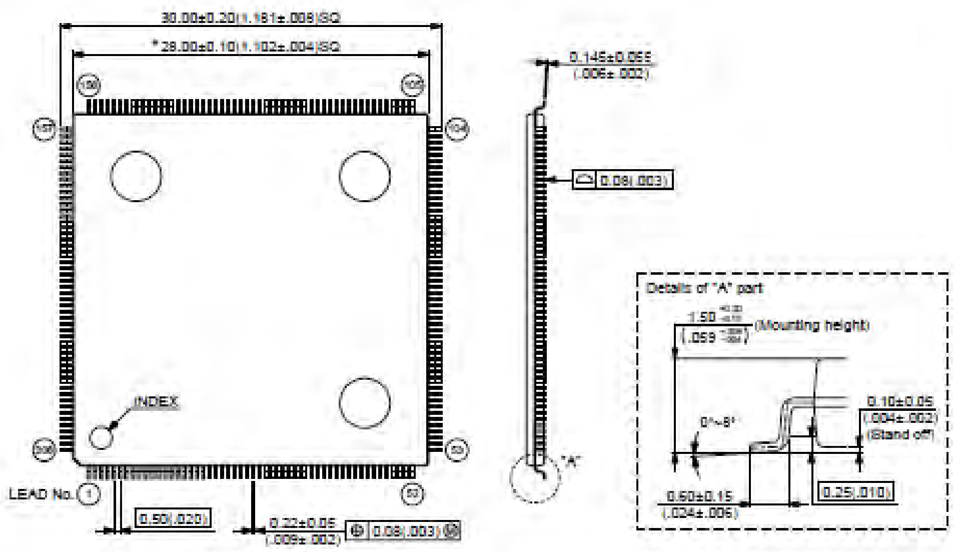
1.4 Package

FPT-208P-M06

208-pin plastic LQFP	Lead pitch	0.50 mm
	Package width × package length	28.0 × 28.0 mm
	Lead shape	Gullwing
	Sealing method	Plastic mold
	Mounting height	1.70 mm MAX
	Weight	2.55g
(FPT-208P-M06)	Code (Reference)	P-LFQFP208-28×28-0.50

208-pin plastic LQFP
(FPT-208P-M06)

Note 1) * : These dimensions do not include resin protrusion.
Note 2) Pins width and pins thickness include plating thickness.
Note 3) Pins width do not include tie bar cutting remainder.



Dimensions in mm (inches).
Note: The values in parentheses are reference values.

Figure 1-5: FPT-208P-M06

1.5 Pinning

1.5.1 MB88F33x ‘Indigo2(-x)’

1.5.1.1 Pinning Overview

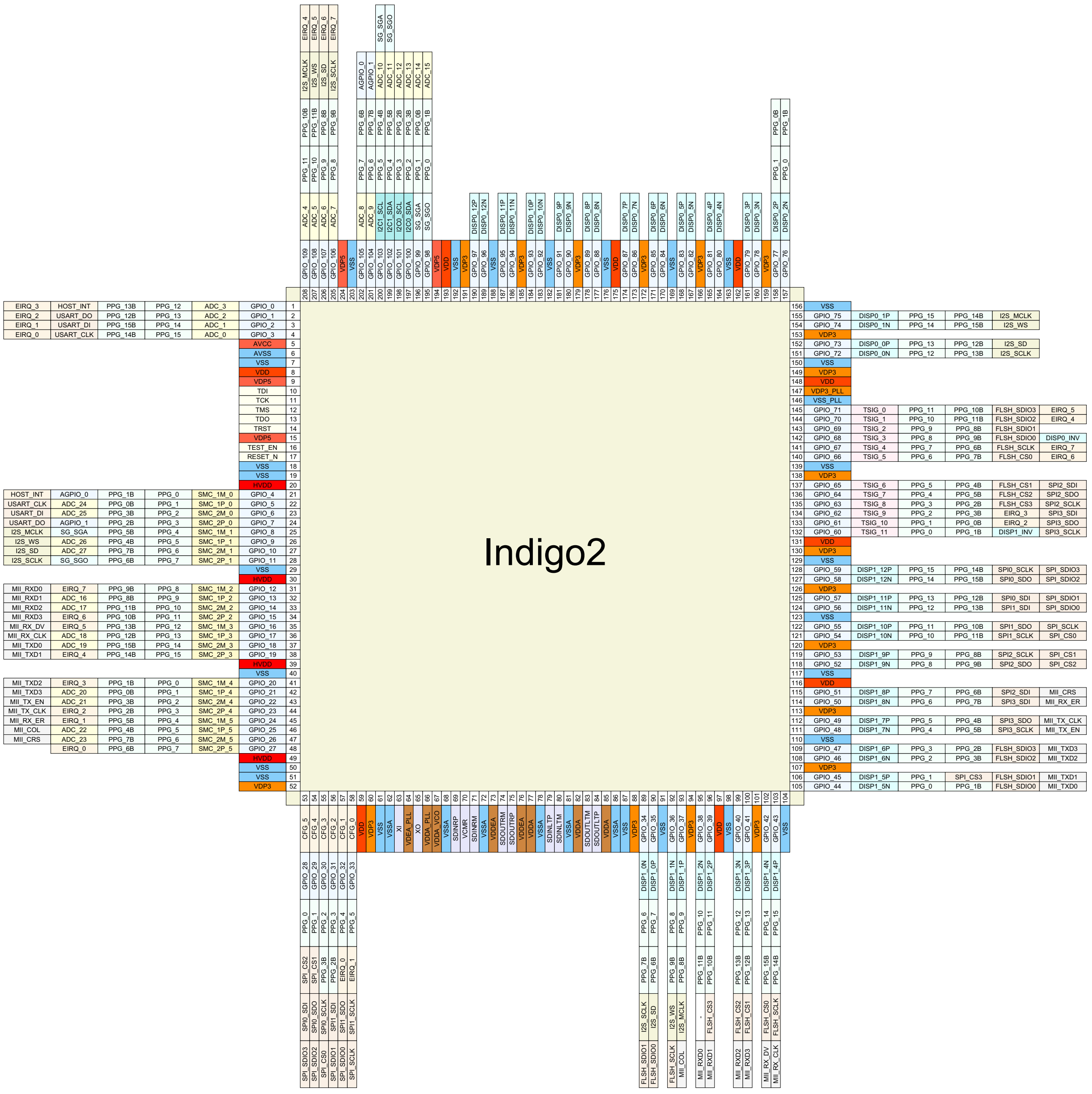


Figure 1-6: MB88F33x ‘Indigo2(-x)’ - Pinning Overview (shows multiplex functionality)

NOTE If you are already familiar with Fujitsu GDCs and SoCs, it is possible that you associate the names DISP0... and DISP1... etc. with the control of multiple external displays (this was the naming convention used in previous Hardware Manuals and documentation). However, the Indigo2 device can only be connected to a SINGLE external display panel. The names DISP0..., DISP1... etc. refer to the data CHANNELS used for the various physical connections (TTL, RSDS, LVDS) to a single panel.

1.6 Pin Descriptions

For detailed information, please refer to the attached pin list (excel sheet).

1.7 Unused Pins

For detailed information, please refer to the attached pin list (excel sheet).

1.8 Pin Multiplexing

The functionality of many pins changes according to the pin multiplexing mode that is set. The information concerning this is a part of the pinning diagram in Section 1.5. The multiplex pin functions are shown as extra columns around the package.

NOTE The pinmux functionality of MB88F33x ‘Indigo2(-x)’ follows a fundamentally different concept to previous Fujitsu GDCs. There are no specific pinmux modes which switch the functionality of groups of pins! Instead, each pin’s characteristics (including pinmux function) is controlled via a specific register (e.g. pin ADC3 is controlled by register ADC3_CTL).

1.8.1 Pinmux Registers

The so-called Pinmux Registers, among the global control registers, are those registers that control the pin multiplexing mode. Other global control registers are needed to control, i.e., the display output modes.

1.9 Display Output

NOTE Please note that it is only possible to assign and relocate the functionality of a display output *within its parent DISPn pin group,* i.e., it is not possible to assign or relocate a DISP**0** output function to the DISP**1** pin group.

1.9.1 TTL/RSDS/LVDS Channel Assignment

For a correct display output, the pin control in different places have to be set correctly. The needed number of display pairs have to be enabled in the global pin multiplexer. But for that, the output pairs have to be set first into display mode in the pinmux registers.

The disp[I]p[O]_mode and disp[I]n[O]_mode in the DISP[I]_[O]_CTL registers have to be set to DISP[I]_[O]P or DISP[I]_[O]N, where [I] is the interface channel number and [O] is the output pair number.

The table summarizes the number of needed display pairs for every display interface mode.

Table 1-1: Display output pairs

Mode	Pair Count	Display Pairs
TTL single channel (24 bit)	13	DISP0_0 ... DISP0_12
TTL single channel (18 bit)	10	DISP0_3 ... DISP0_12
RSDS single channel (24 bit)	13	DISP0_0 ... DISP0_12
RSDS single channel (18 bit)	10	DISP0_3 ... DISP0_12
RSDS dual channel (2x 24 bit)	26	DISP0_0 ... DISP0_12 and DISP1_0 ... DISP1_12
RSDS dual channel (2x 18 bit)	20	DISP0_3 ... DISP0_12 and 10 pairs out of DISP1_0 ... DISP1_12
LVDS single channel (24 bit)	5	DISP0_8 ... DISP0_12
LVDS single channel (18 bit)	4	DISP0_8 ... DISP0_11
LVDS dual channel (2x 24 bit)	10	DISP0_3 ... DISP0_12
LVDS dual channel (2x 18 bit)	8	DISP0_4 ... DISP0_11

The next step is to select the clock output pair. For single channel mode, one output pair has to be a clock output. For dual channel mode two output pairs have to be a clock output. The pair, which will be clock can be selected with the D[**I**]_CLKSEL_[**O**] field in the DISP[**I**]_PN[**O**]_CTL registers.

Where [**I**] is the interface channel number and [**O**] is the output pair number.

For the display pixel mapping in dual channel mode, please refer to Figure 5-19 in chapter Data Modes.

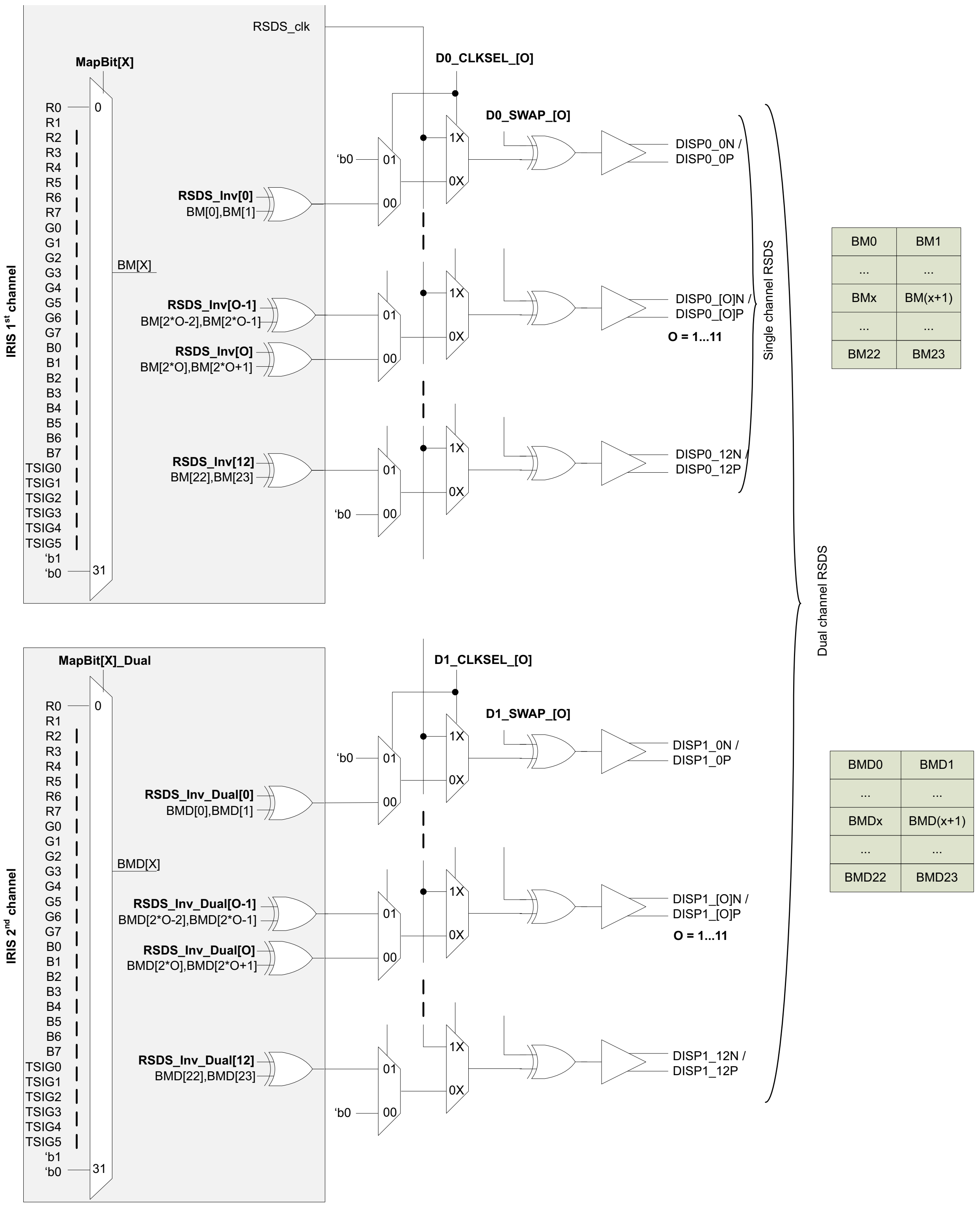


Figure 1-7: RSDS

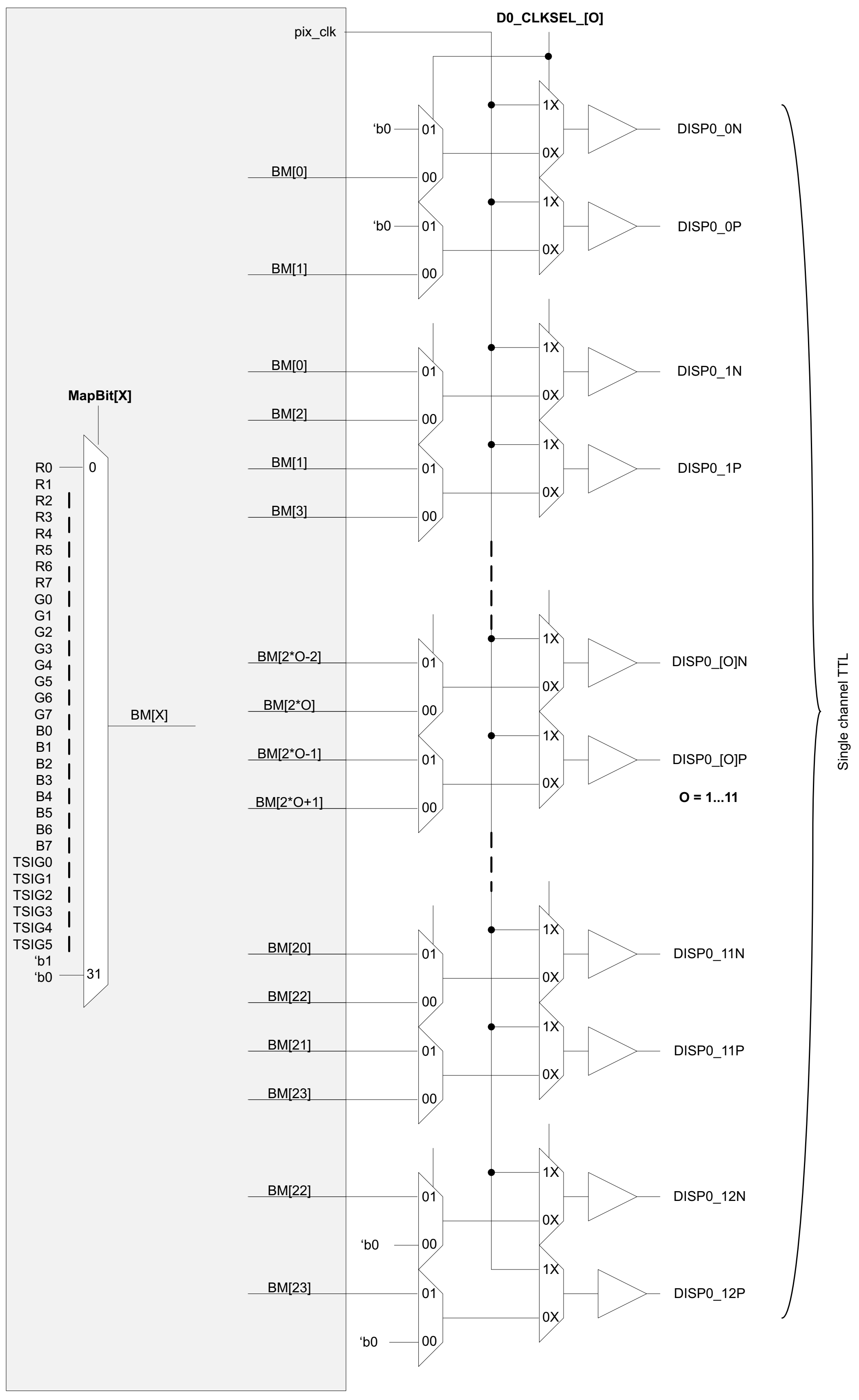


Figure 1-8: TTL

NOTE TSIG0 .. TSIG5 are not available for use.

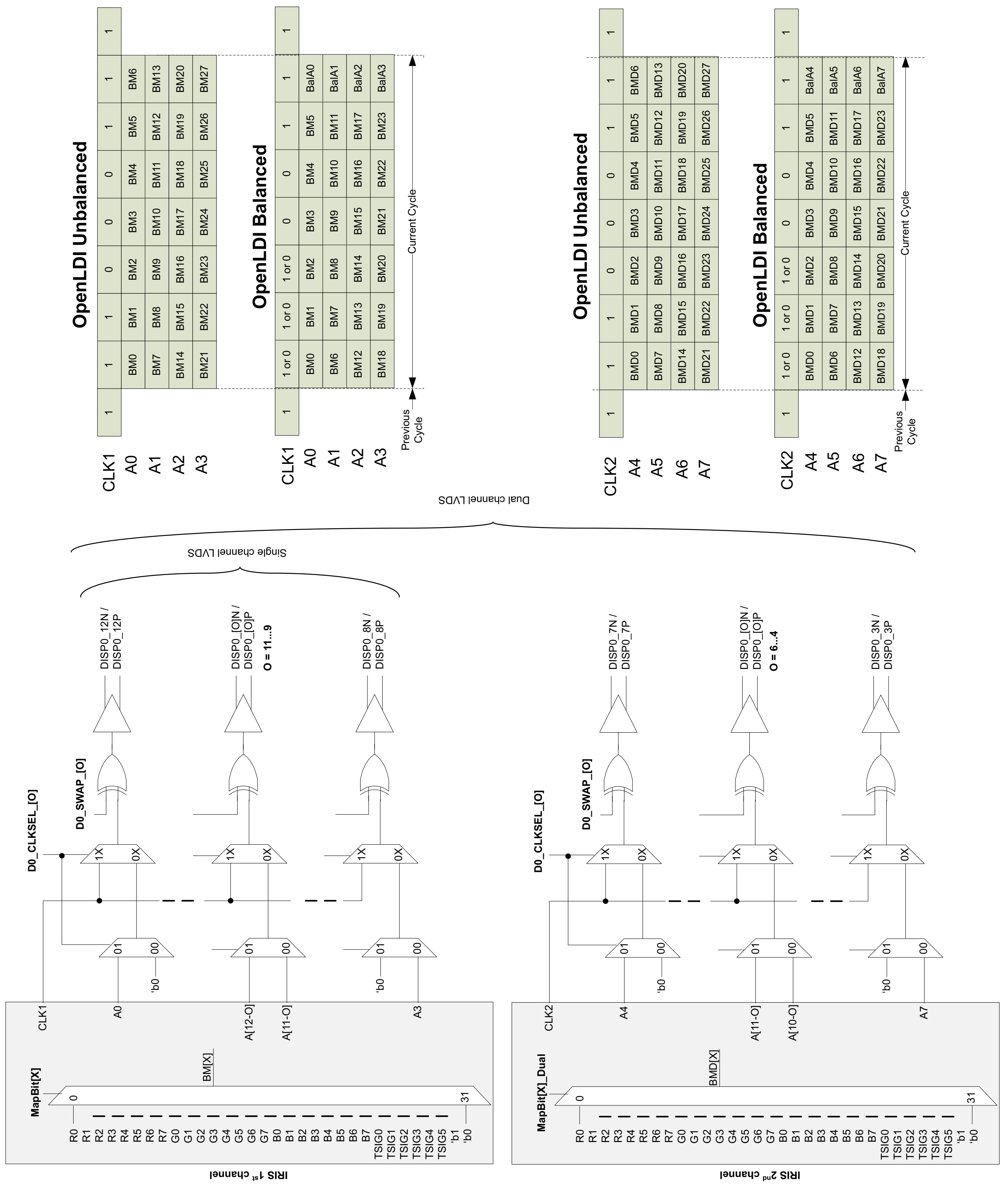


Figure 1-9: LVDS

Chapter 2: Global Control

2.1 General

The Global Control unit is part of the System block. It controls global functions of the MB88F33x ‘Indigo2(-x)’.

2.2 Block Diagram

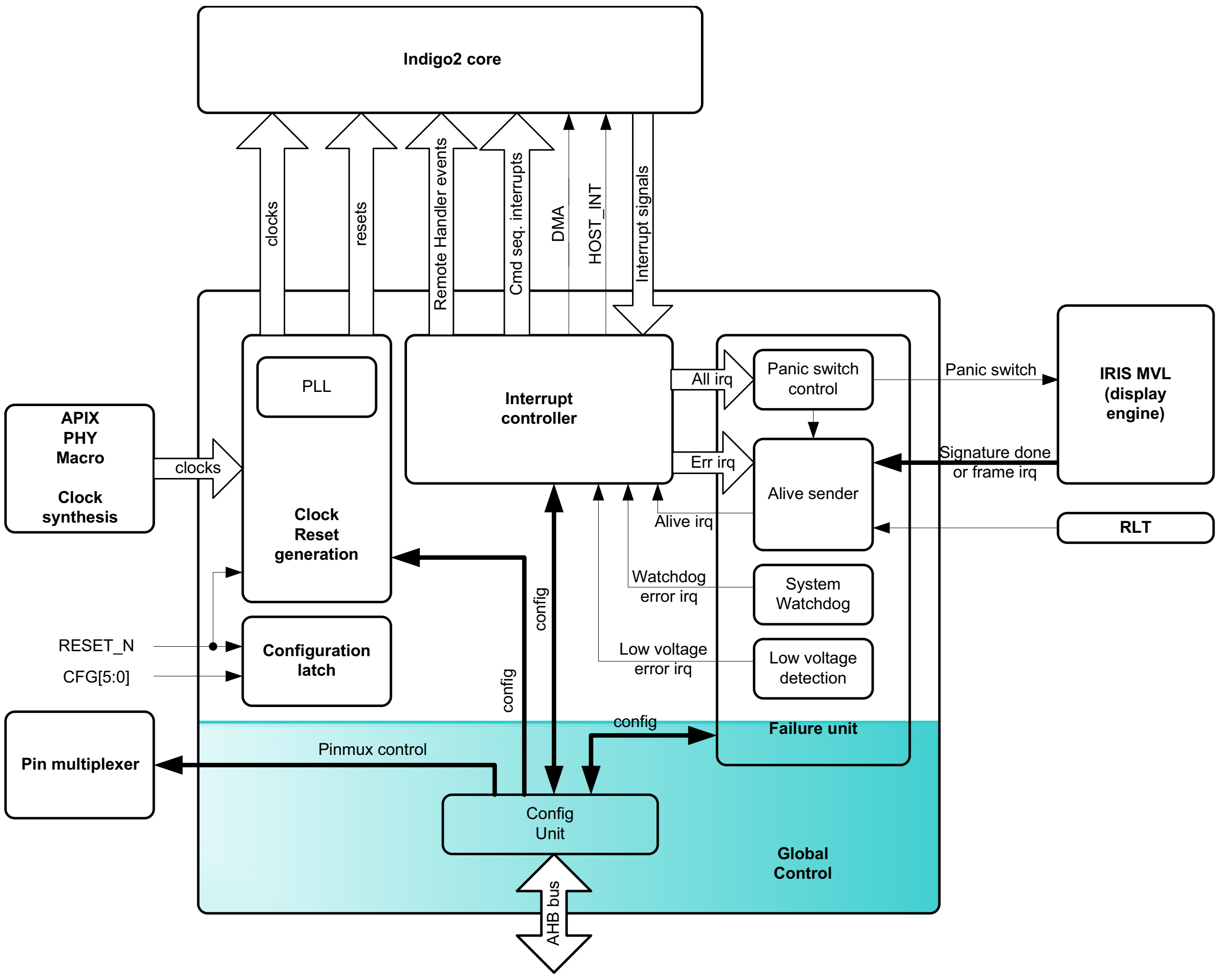


Figure 2-1: Block Diagram

2.3 Global Control Register

2.3.1 Unlocking Global Control Register

Registers in the Global control register space can only be accessed after unlocking this block.Unlocking is done by writing the unlock code:

wr_reg_access GLOBALCONTROL_LOCKUNLOCK { .lockunlock = 0x69b309b8; };

After setting up of the control register the global control register space can be locked by writing the lock code:

wr_reg_access GLOBALCONTROL_LOCKUNLOCK { .lockunlock = 0xa82be775; };

Current lock status can be detected by reading:

rd_reg_access GLOBALCONTROL_LOCKSTATUS {};

2.3.2 Global Control Register Overview

Table 2-1: Registers Overview

Base Address(es)	**Instance no 0: BASEADDR="00000000"**	
Absolute Address	**Register Name**	**Register Description**
BASEADDR + 0x0000	CHIP_ID	CHIP Identification
BASEADDR + 0x0004	CHIP_INFO	CHIP Information
BASEADDR + 0x0008	GC_TEST	Test register
BASEADDR + 0x000C	GC_PROGID	Programming ID
BASEADDR + 0x0010	LockUnlock	Register to lock or unlock write access to registers of this unit with lock property.
BASEADDR + 0x0014	LockStatus	Lock status for write access to registers of this unit with lock property.
BASEADDR + 0x0018	IFC_CTRL 🔒	Interface control
BASEADDR + 0x001C	LVD 🔒	Low voltage detection
BASEADDR + 0x0020	SYSWD_RES 🔒	System Watchdog reset
BASEADDR + 0x0024	SYSWD_CTL 🔒	System Watchdog control
BASEADDR + 0x0028	SYSWD_CNT 🔒	System Watchdog counter value
BASEADDR + 0x002C	SYSWD_WNDW 🔒	System Watchdog counter window
BASEADDR + 0x0030	SYSWD_STS 🔒	System Watchdog counter value
BASEADDR + 0x0034	ALVSND_CTL 🔒	Alive Sender control
BASEADDR + 0x0038	ALVSND_MEN 🔒	Alive Sender Mask enable
BASEADDR + 0x003C	ALVSND_STS 🔒	Alive Sender mask status
BASEADDR + 0x0040	Reserved	Do not modify
BASEADDR + 0x0044	Reserved	Do not modify
BASEADDR + 0x0048	Reserved 🔒	Do not modify
BASEADDR + 0x004C	Reserved 🔒	Do not modify
BASEADDR + 0x0050	Reserved 🔒	Do not modify
BASEADDR + 0x0054	Reserved 🔒	Do not modify
BASEADDR + 0x0058	Reserved 🔒	Do not modify
BASEADDR + 0x0080	CLOCK_SELECTION 🔒	Clock selection Register
BASEADDR + 0x0084	CLOCK_DIV 🔒	Clock divider ratio Register
BASEADDR + 0x0088	PLL_CTRL 🔒	PLL Control
BASEADDR + 0x008C	PLL_PIXCLOCK 🔒	Clock control for PLL Pixel clock
BASEADDR + 0x0090	PLL_CLOCK_DIV 🔒	Clock divider ratio for PLL clock
BASEADDR + 0x0094	Reserved 🔒	Do not modify
BASEADDR + 0x0098	PWR_CTRL 🔒	Power Down Control Reset
BASEADDR + 0x0100	DISP_CTL 🔒	Control of display output.
BASEADDR + 0x0104	DISP0_PN0_CTL 🔒	Control of DISP0 P0/N0 output pad.
BASEADDR + 0x0108	DISP0_PN1_CTL 🔒	Control of DISP0 P1/N1 output pad.
BASEADDR + 0x010C	DISP0_PN2_CTL 🔒	Control of DISP0 P2/N2 output pad.
BASEADDR + 0x0110	DISP0_PN3_CTL 🔒	Control of DISP0 P3/N3 output pad.
BASEADDR + 0x0114	DISP0_PN4_CTL 🔒	Control of DISP0 P4/N4 output pad.
BASEADDR + 0x0118	DISP0_PN5_CTL 🔒	Control of DISP0 P5/N5 output pad.
BASEADDR + 0x011C	DISP0_PN6_CTL 🔒	Control of DISP0 P6/N6 output pad.
BASEADDR + 0x0120	DISP0_PN7_CTL 🔒	Control of DISP0 P7/N7 output pad.
BASEADDR + 0x0124	DISP0_PN8_CTL 🔒	Control of DISP0 P8/N8 output pad.
BASEADDR + 0x0128	DISP0_PN9_CTL 🔒	Control of DISP0 P9/N9 output pad.
BASEADDR + 0x012C	DISP0_PN10_CTL 🔒	Control of DISP0 P10/N10 output pad.
BASEADDR + 0x0130	DISP0_PN11_CTL 🔒	Control of DISP0 P11/N11 output pad.
BASEADDR + 0x0134	DISP0_PN12_CTL 🔒	Control of DISP0 P12/N12 output pad.
BASEADDR + 0x0138	DISP1_PN0_CTL 🔒	Control of DISP1 P0/N0 output pad.

Table 2-1: Registers Overview (Continued)

Base Address(es)	**Instance no 0: BASEADDR="00000000"**	
Absolute Address	**Register Name**	**Register Description**
BASEADDR + 0x013C	DISP1_PN1_CTL 🔒	Control of DISP1 P1/N1 output pad.
BASEADDR + 0x0140	DISP1_PN2_CTL 🔒	Control of DISP1 P2/N2 output pad.
BASEADDR + 0x0144	DISP1_PN3_CTL 🔒	Control of DISP1 P3/N3 output pad.
BASEADDR + 0x0148	DISP1_PN4_CTL 🔒	Control of DISP1 P4/N4 output pad.
BASEADDR + 0x014C	DISP1_PN5_CTL 🔒	Control of DISP1 P5/N5 output pad.
BASEADDR + 0x0150	DISP1_PN6_CTL 🔒	Control of DISP1 P6/N6 output pad.
BASEADDR + 0x0154	DISP1_PN7_CTL 🔒	Control of DISP1 P7/N7 output pad.
BASEADDR + 0x0158	DISP1_PN8_CTL 🔒	Control of DISP1 P8/N8 output pad.
BASEADDR + 0x015C	DISP1_PN9_CTL 🔒	Control of DISP1 P9/N9 output pad.
BASEADDR + 0x0160	DISP1_PN10_CTL 🔒	Control of DISP1 P10/N10 output pad.
BASEADDR + 0x0164	DISP1_PN11_CTL 🔒	Control of DISP1 P11/N11 output pad.
BASEADDR + 0x0168	DISP1_PN12_CTL 🔒	Control of DISP1 P12/N12 output pad.
BASEADDR + 0x016C	TSIG0_3_CTL 🔒	Control of TSIG0-3 output pads.
BASEADDR + 0x0170	TSIG4_7_CTL 🔒	Control of TSIG4-7 output pads.
BASEADDR + 0x0174	TSIG8_11_CTL 🔒	Control of TSIG8-11 output pads.
BASEADDR + 0x0178	DSPINV_CTL 🔒	Control of DSPINV output pad.
BASEADDR + 0x0200	ADC3_CTL 🔒	Control of ADC3 pad
BASEADDR + 0x0204	ADC2_CTL 🔒	Control of ADC2 pad
BASEADDR + 0x0208	ADC1_CTL 🔒	Control of ADC1 pad
BASEADDR + 0x020C	ADC0_CTL 🔒	Control of ADC0 pad
BASEADDR + 0x0210	SMC_1M_0_CTL 🔒	Control of SMC_1M_0 pad
BASEADDR + 0x0214	SMC_1P_0_CTL 🔒	Control of SMC_1P_0 pad
BASEADDR + 0x0218	SMC_2M_0_CTL 🔒	Control of SMC_2M_0 pad
BASEADDR + 0x021C	SMC_2P_0_CTL 🔒	Control of SMC_2P_0 pad
BASEADDR + 0x0220	SMC_1M_1_CTL 🔒	Control of SMC_1M_1 pad
BASEADDR + 0x0224	SMC_1P_1_CTL 🔒	Control of SMC_1P_1 pad
BASEADDR + 0x0228	SMC_2M_1_CTL 🔒	Control of SMC_2M_1 pad
BASEADDR + 0x022C	SMC_2P_1_CTL 🔒	Control of SMC_2P_1 pad
BASEADDR + 0x0230	SMC_1M_2_CTL 🔒	Control of SMC_1M_2 pad
BASEADDR + 0x0234	SMC_1P_2_CTL 🔒	Control of SMC_1P_2 pad
BASEADDR + 0x0238	SMC_2M_2_CTL 🔒	Control of SMC_2M_2 pad
BASEADDR + 0x023C	SMC_2P_2_CTL 🔒	Control of SMC_2P_2 pad
BASEADDR + 0x0240	SMC_1M_3_CTL 🔒	Control of SMC_1M_3 pad
BASEADDR + 0x0244	SMC_1P_3_CTL 🔒	Control of SMC_1P_3 pad
BASEADDR + 0x0248	SMC_2M_3_CTL 🔒	Control of SMC_2M_3 pad
BASEADDR + 0x024C	SMC_2P_3_CTL 🔒	Control of SMC_2P_3 pad
BASEADDR + 0x0250	SMC_1M_4_CTL 🔒	Control of SMC_1M_4 pad
BASEADDR + 0x0254	SMC_1P_4_CTL 🔒	Control of SMC_1P_4 pad
BASEADDR + 0x0258	SMC_2M_4_CTL 🔒	Control of SMC_2M_4 pad
BASEADDR + 0x025C	SMC_2P_4_CTL 🔒	Control of SMC_2P_4 pad
BASEADDR + 0x0260	SMC_1M_5_CTL 🔒	Control of SMC_1M_5 pad
BASEADDR + 0x0264	SMC_1P_5_CTL 🔒	Control of SMC_1P_5 pad
BASEADDR + 0x0268	SMC_2M_5_CTL 🔒	Control of SMC_2M_5 pad
BASEADDR + 0x026C	SMC_2P_5_CTL 🔒	Control of SMC_2P_5 pad
BASEADDR + 0x0270	CFG5_CTL 🔒	Control of CFG5 pad
BASEADDR + 0x0274	CFG4_CTL 🔒	Control of CFG4 pad
BASEADDR + 0x0278	CFG3_CTL 🔒	Control of CFG3 pad
BASEADDR + 0x027C	CFG2_CTL 🔒	Control of CFG2 pad
BASEADDR + 0x0280	CFG1_CTL 🔒	Control of CFG1 pad
BASEADDR + 0x0284	CFG0_CTL 🔒	Control of CFG0 pad

Table 2-1: Registers Overview (Continued)

Base Address(es)	Instance no 0: BASEADDR="00000000"	
Absolute Address	**Register Name**	**Register Description**
BASEADDR + 0x0288	DISP1_0_CTL 🔒	Control of DISP1_0 pad
BASEADDR + 0x028C	DISP1_1_CTL 🔒	Control of DISP1_1 pad
BASEADDR + 0x0290	DISP1_2_CTL 🔒	Control of DISP1_2 pad
BASEADDR + 0x0294	DISP1_3_CTL 🔒	Control of DISP1_3 pad
BASEADDR + 0x0298	DISP1_4_CTL 🔒	Control of DISP1_4 pad
BASEADDR + 0x029C	DISP1_5_CTL 🔒	Control of DISP1_5 pad
BASEADDR + 0x02A0	DISP1_6_CTL 🔒	Control of DISP1_6 pad
BASEADDR + 0x02A4	DISP1_7_CTL 🔒	Control of DISP1_7 pad
BASEADDR + 0x02A8	DISP1_8_CTL 🔒	Control of DISP1_8 pad
BASEADDR + 0x02AC	DISP1_9_CTL 🔒	Control of DISP1_9 pad
BASEADDR + 0x02B0	DISP1_10_CTL 🔒	Control of DISP1_10 pad
BASEADDR + 0x02B4	DISP1_11_CTL 🔒	Control of DISP1_11 pad
BASEADDR + 0x02B8	DISP1_12_CTL 🔒	Control of DISP1_12 pad
BASEADDR + 0x02BC	TSIG11_CTL 🔒	Control of TSIG11 pad
BASEADDR + 0x02C0	TSIG10_CTL 🔒	Control of TSIG10 pad
BASEADDR + 0x02C4	TSIG9_CTL 🔒	Control of TSIG9 pad
BASEADDR + 0x02C8	TSIG8_CTL 🔒	Control of TSIG8 pad
BASEADDR + 0x02CC	TSIG7_CTL 🔒	Control of TSIG7 pad
BASEADDR + 0x02D0	TSIG6_CTL 🔒	Control of TSIG6 pad
BASEADDR + 0x02D4	TSIG5_CTL 🔒	Control of TSIG5 pad
BASEADDR + 0x02D8	TSIG4_CTL 🔒	Control of TSIG4 pad
BASEADDR + 0x02DC	TSIG3_CTL 🔒	Control of TSIG3 pad
BASEADDR + 0x02E0	TSIG2_CTL 🔒	Control of TSIG2 pad
BASEADDR + 0x02E4	TSIG1_CTL 🔒	Control of TSIG1 pad
BASEADDR + 0x02E8	TSIG0_CTL 🔒	Control of TSIG0 pad
BASEADDR + 0x02EC	DISP0_0_CTL 🔒	Control of DISP0_0 pad
BASEADDR + 0x02F0	DISP0_1_CTL 🔒	Control of DISP0_1 pad
BASEADDR + 0x02F4	DISP0_2_CTL 🔒	Control of DISP0_2 pad
BASEADDR + 0x02F8	DISP0_3_CTL 🔒	Control of DISP0_3 pad
BASEADDR + 0x02FC	DISP0_4_CTL 🔒	Control of DISP0_4 pad
BASEADDR + 0x0300	DISP0_5_CTL 🔒	Control of DISP0_5 pad
BASEADDR + 0x0304	DISP0_6_CTL 🔒	Control of DISP0_6 pad
BASEADDR + 0x0308	DISP0_7_CTL 🔒	Control of DISP0_7 pad
BASEADDR + 0x030C	DISP0_8_CTL 🔒	Control of DISP0_8 pad
BASEADDR + 0x0310	DISP0_9_CTL 🔒	Control of DISP0_9 pad
BASEADDR + 0x0314	DISP0_10_CTL 🔒	Control of DISP0_10 pad
BASEADDR + 0x0318	DISP0_11_CTL 🔒	Control of DISP0_11 pad
BASEADDR + 0x031C	DISP0_12_CTL 🔒	Control of DISP0_12 pad
BASEADDR + 0x0320	SG_SGO_CTL 🔒	Control of SG_SGO pad
BASEADDR + 0x0324	SG_SGA_CTL 🔒	Control of SG_SGA pad
BASEADDR + 0x0328	I2C0_SDA_CTL 🔒	Control of I2C0_SDA pad
BASEADDR + 0x032C	I2C0_SCL_CTL 🔒	Control of I2C0_SCL pad
BASEADDR + 0x0330	I2C1_SDA_CTL 🔒	Control of I2C1_SDA pad
BASEADDR + 0x0334	I2C1_SCL_CTL 🔒	Control of I2C1_SCL pad
BASEADDR + 0x0338	ADC9_CTL 🔒	Control of ADC9 pad
BASEADDR + 0x033C	ADC8_CTL 🔒	Control of ADC8 pad
BASEADDR + 0x0340	ADC7_CTL 🔒	Control of ADC7 pad
BASEADDR + 0x0344	ADC6_CTL 🔒	Control of ADC6 pad
BASEADDR + 0x0348	ADC5_CTL 🔒	Control of ADC5 pad
BASEADDR + 0x034C	ADC4_CTL 🔒	Control of ADC4 pad

Table 2-1: Registers Overview (Continued)

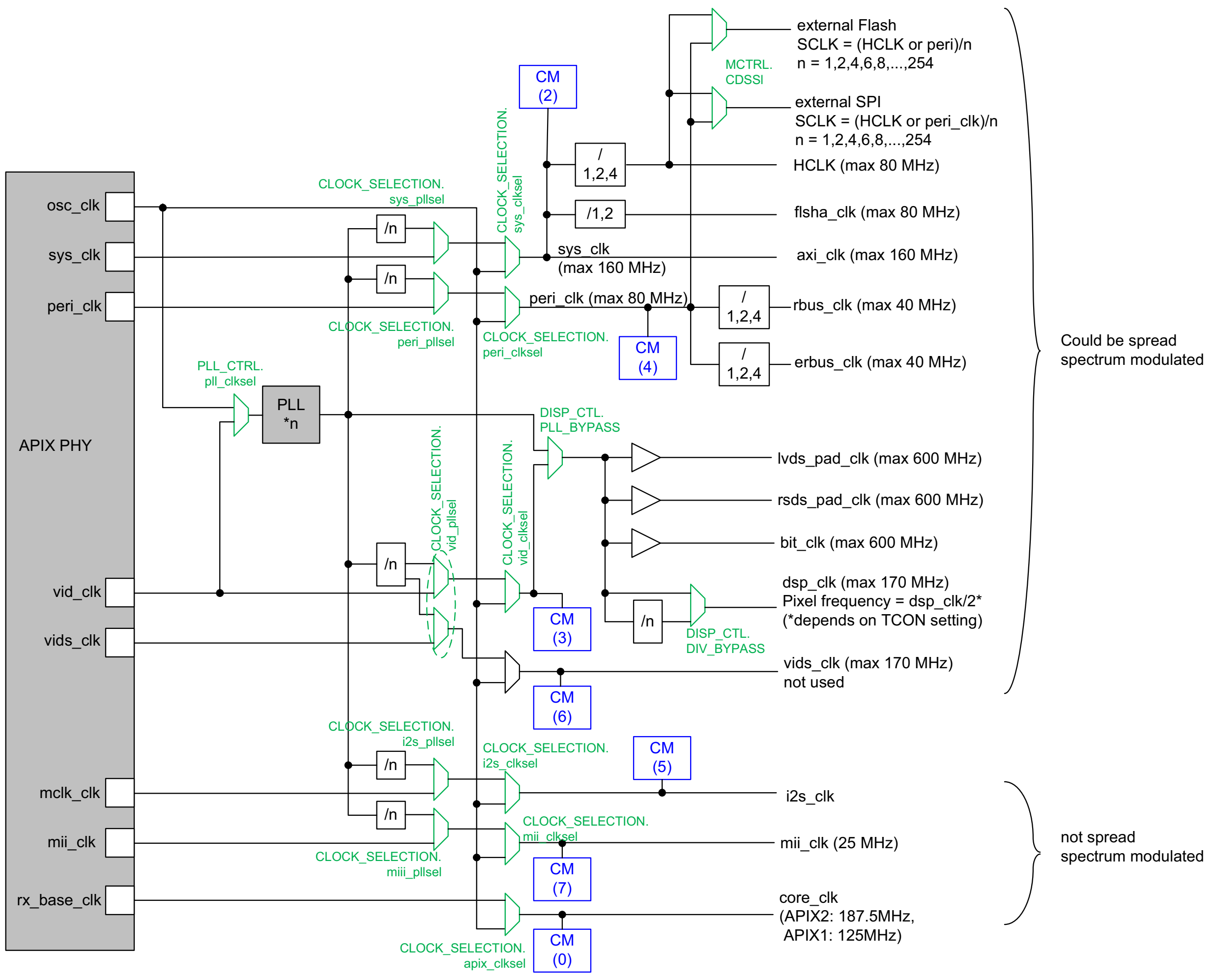
Base Address(es)	**Instance no 0: BASEADDR="00000000"**	
Absolute Address	**Register Name**	**Register Description**
BASEADDR + 0x0400	MODULE_IRQ_STS 🔒	Interrupt status for submodule
BASEADDR + 0x0404	APIX_CLR 🔒	Interrupt clear for APIX interrupts
BASEADDR + 0x0408	APIX_SET 🔒	Interrupt set for APIX interrupts
BASEADDR + 0x040C	APIX_STS 🔒	Interrupt status for APIX interrupts
BASEADDR + 0x0410	ASHELL_RH_CLR 🔒	Interrupt clear for ASHELL_RH interrupts
BASEADDR + 0x0414	ASHELL_RH_SET 🔒	Interrupt set for ASHELL_RH interrupts
BASEADDR + 0x0418	ASHELL_RH_STS 🔒	Interrupt status for ASHELL_RH interrupts
BASEADDR + 0x041C	E2IP_CLR 🔒	Interrupt clear for E2IP interrupts
BASEADDR + 0x0420	E2IP_SET 🔒	Interrupt set for E2IP interrupts
BASEADDR + 0x0424	E2IP_STS 🔒	Interrupt status for E2IP interrupts
BASEADDR + 0x0428	CFF_CTRL_CLR 🔒	Interrupt clear for CFF_CTRL interrupts
BASEADDR + 0x042C	CFF_CTRL_SET 🔒	Interrupt set for CFF_CTRL interrupts
BASEADDR + 0x0430	CFF_CTRL_STS 🔒	Interrupt status for CFF_CTRL interrupts
BASEADDR + 0x0434	CFF_FIFO_CLR 🔒	Interrupt clear for CFF_FIFO interrupts
BASEADDR + 0x0438	CFF_FIFO_SET 🔒	Interrupt set for CFF_FIFO interrupts
BASEADDR + 0x043C	CFF_FIFO_STS 🔒	Interrupt status for CFF_FIFO interrupts
BASEADDR + 0x0440	RLT_STS 🔒	Interrupt status for RLT interrupts
BASEADDR + 0x0444	LIN_STS 🔒	Interrupt status for LIN interrupts
BASEADDR + 0x0448	PPG_STS 🔒	Interrupt status for PPG interrupts
BASEADDR + 0x044C	I2C0_STS 🔒	Interrupt status for I2C0 interrupts
BASEADDR + 0x0450	I2C1_STS 🔒	Interrupt status for I2C1 interrupts
BASEADDR + 0x0454	SGE_CLR 🔒	Interrupt clear for SGE interrupts
BASEADDR + 0x0458	SGE_SET 🔒	Interrupt set for SGE interrupts
BASEADDR + 0x045C	SGE_STS 🔒	Interrupt status for SGE interrupts
BASEADDR + 0x0460	ADC_STS 🔒	Interrupt status for ADC interrupts
BASEADDR + 0x0464	EIRQ_STS 🔒	Interrupt status for EIRQ interrupts
BASEADDR + 0x0468	ESPI_STS 🔒	Interrupt status for ESPI interrupts
BASEADDR + 0x046C	IRIS_CLR 🔒	Interrupt clear for IRIS interrupts
BASEADDR + 0x0470	IRIS_SET 🔒	Interrupt set for IRIS interrupts
BASEADDR + 0x0474	IRIS_STS 🔒	Interrupt status for IRIS interrupts
BASEADDR + 0x0478	CMDSEQ_CLR 🔒	Interrupt clear for CMDSEQ interrupts
BASEADDR + 0x047C	CMDSEQ_SET 🔒	Interrupt set for CMDSEQ interrupts
BASEADDR + 0x0480	CMDSEQ_STS 🔒	Interrupt status for CMDSEQ interrupts
BASEADDR + 0x0484	GC_CLR 🔒	Interrupt clear for GC interrupts
BASEADDR + 0x0488	GC_SET 🔒	Interrupt set for GC interrupts
BASEADDR + 0x048C	GC_STS 🔒	Interrupt status for GC interrupts
BASEADDR + 0x0490	DMAC_STS 🔒	Interrupt status for DMAC interrupts
BASEADDR + 0x0494	FSPI_STS 🔒	Interrupt status for FSPI interrupts
BASEADDR + 0x0498	PRGCRC_STS 🔒	Interrupt status for PRGCRC interrupts
BASEADDR + 0x049C	INTERCONNECT_CLR 🔒	Interrupt clear for INTERCONNECT interrupts
BASEADDR + 0x04A0	INTERCONNECT_SET 🔒	Interrupt set for INTERCONNECT interrupts
BASEADDR + 0x04A4	INTERCONNECT_STS 🔒	Interrupt status for INTERCONNECT interrupts
BASEADDR + 0x04A8	IRQ_CMDSEQ_SEL0 🔒	Interrupt select for command sequencer
BASEADDR + 0x04AC	IRQ_CMDSEQ_SEL1 🔒	Interrupt select for command sequencer
BASEADDR + 0x04B0	CFF_TRG_SEL0 🔒	Trigger select for Config Fifo
BASEADDR + 0x04B4	CFF_TRG_SEL1 🔒	Trigger select for Config Fifo
BASEADDR + 0x04B8	HIRQ_CTL 🔒	Host interrupt control
BASEADDR + 0x04BC	APIX_HIEN 🔒	Host interrupt enable for APIX interrupts
BASEADDR + 0x04C0	ASHELL_RH_HIEN 🔒	Host interrupt enable for ASHELL_RH interrupts
BASEADDR + 0x04C4	E2IP_HIEN 🔒	Host interrupt enable for E2IP interrupts

Table 2-1: Registers Overview (Continued)

Base Address(es)	Instance no 0: BASEADDR="00000000"	
Absolute Address	**Register Name**	**Register Description**
BASEADDR + 0x04C8	CFF_CTRL_HIEN 🔒	Host interrupt enable for CFF_CTRL interrupts
BASEADDR + 0x04CC	CFF_FIFO_HIEN 🔒	Host interrupt enable for CFF_FIFO interrupts
BASEADDR + 0x04D0	RLT_HIEN 🔒	Host interrupt enable for RLT interrupts
BASEADDR + 0x04D4	LIN_HIEN 🔒	Host interrupt enable for LIN interrupts
BASEADDR + 0x04D8	PPG_HIEN 🔒	Host interrupt enable for PPG interrupts
BASEADDR + 0x04DC	I2C0_HIEN 🔒	Host interrupt enable for I2C0 interrupts
BASEADDR + 0x04E0	I2C1_HIEN 🔒	Host interrupt enable for I2C1 interrupts
BASEADDR + 0x04E4	SGE_HIEN 🔒	Host interrupt enable for SGE interrupts
BASEADDR + 0x04E8	ADC_HIEN 🔒	Host interrupt enable for ADC interrupts
BASEADDR + 0x04EC	EIRQ_HIEN 🔒	Host interrupt enable for EIRQ interrupts
BASEADDR + 0x04F0	ESPI_HIEN 🔒	Host interrupt enable for ESPI interrupts
BASEADDR + 0x04F4	IRIS_HIEN 🔒	Host interrupt enable for IRIS interrupts
BASEADDR + 0x04F8	CMDSEQ_HIEN 🔒	Host interrupt enable for CMDSEQ interrupts
BASEADDR + 0x04FC	GC_HIEN 🔒	Host interrupt enable for GC interrupts
BASEADDR + 0x0500	DMAC_HIEN 🔒	Host interrupt enable for DMAC interrupts
BASEADDR + 0x0504	FSPI_HIEN 🔒	Host interrupt enable for FSPI interrupts
BASEADDR + 0x0508	PRGCRC_HIEN 🔒	Host interrupt enable for PRGCRC interrupts
BASEADDR + 0x050C	INTERCONNECT_HIEN 🔒	Host interrupt enable for INTERCONNECT interrupts
BASEADDR + 0x0510	PNCSW_CTL 🔒	Panic Switch control
BASEADDR + 0x0514	APIX_PSEN 🔒	Panic switch enable for APIX interrupts
BASEADDR + 0x0518	ASHELL_RH_PSEN 🔒	Panic switch enable for ASHELL_RH interrupts
BASEADDR + 0x051C	E2IP_PSEN 🔒	Panic switch enable for E2IP interrupts
BASEADDR + 0x0520	CFF_CTRL_PSEN 🔒	Panic switch enable for CFF_CTRL interrupts
BASEADDR + 0x0524	CFF_FIFO_PSEN 🔒	Panic switch enable for CFF_FIFO interrupts
BASEADDR + 0x0528	RLT_PSEN 🔒	Panic switch enable for RLT interrupts
BASEADDR + 0x052C	LIN_PSEN 🔒	Panic switch enable for LIN interrupts
BASEADDR + 0x0530	PPG_PSEN 🔒	Panic switch enable for PPG interrupts
BASEADDR + 0x0534	I2C0_PSEN 🔒	Panic switch enable for I2C0 interrupts
BASEADDR + 0x0538	I2C1_PSEN 🔒	Panic switch enable for I2C1 interrupts
BASEADDR + 0x053C	SGE_PSEN 🔒	Panic switch enable for SGE interrupts
BASEADDR + 0x0540	ADC_PSEN 🔒	Panic switch enable for ADC interrupts
BASEADDR + 0x0544	EIRQ_PSEN 🔒	Panic switch enable for EIRQ interrupts
BASEADDR + 0x0548	ESPI_PSEN 🔒	Panic switch enable for ESPI interrupts
BASEADDR + 0x054C	IRIS_PSEN 🔒	Panic switch enable for IRIS interrupts
BASEADDR + 0x0550	CMDSEQ_PSEN 🔒	Panic switch enable for CMDSEQ interrupts
BASEADDR + 0x0554	GC_PSEN 🔒	Panic switch enable for GC interrupts
BASEADDR + 0x0558	DMAC_PSEN 🔒	Panic switch enable for DMAC interrupts
BASEADDR + 0x055C	FSPI_PSEN 🔒	Panic switch enable for FSPI interrupts
BASEADDR + 0x0560	PRGCRC_PSEN 🔒	Panic switch enable for PRGCRC interrupts
BASEADDR + 0x0564	INTERCONNECT_PSEN 🔒	Panic switch enable for INTERCONNECT interrupts
BASEADDR + 0x0568	DMA_CNTRL 🔒	Control for Interrupt base DMA requests

2.4 Clock Structure

2.4.1 Overview



SMC
PWM_baseclk = rbus_clk/n
n = 1,4,5,6,8,10,12,16
PWM_period = PWM_baseclk * 255/511

RLT (32bit)
count_clock = erbus_clk/n;
(n = 1,2,4,8,16,32)

I2C
bitrate = rbus_clk/(n*12+16+y)
(n = 1...63; y = 0...6)

USART/LIN
baudrate = rbus_clk/(n+1); (n < 2^19)

PWM/PPG
count_clock = rbus_clk/n; (n = 1,4,16,64)
period (8bit mode) = count_clock*p8
p8 = 1...255
period (16bit mode) = count_clock*p16
p16 = 1...65535

Figure 2-2: Clock structure

CM = Clock measurement

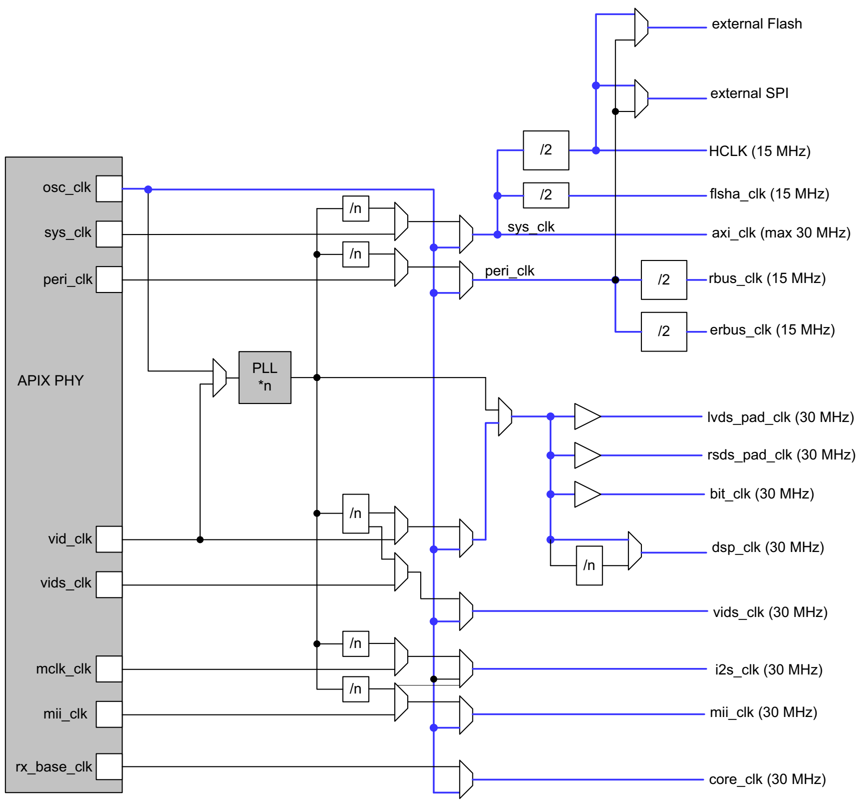
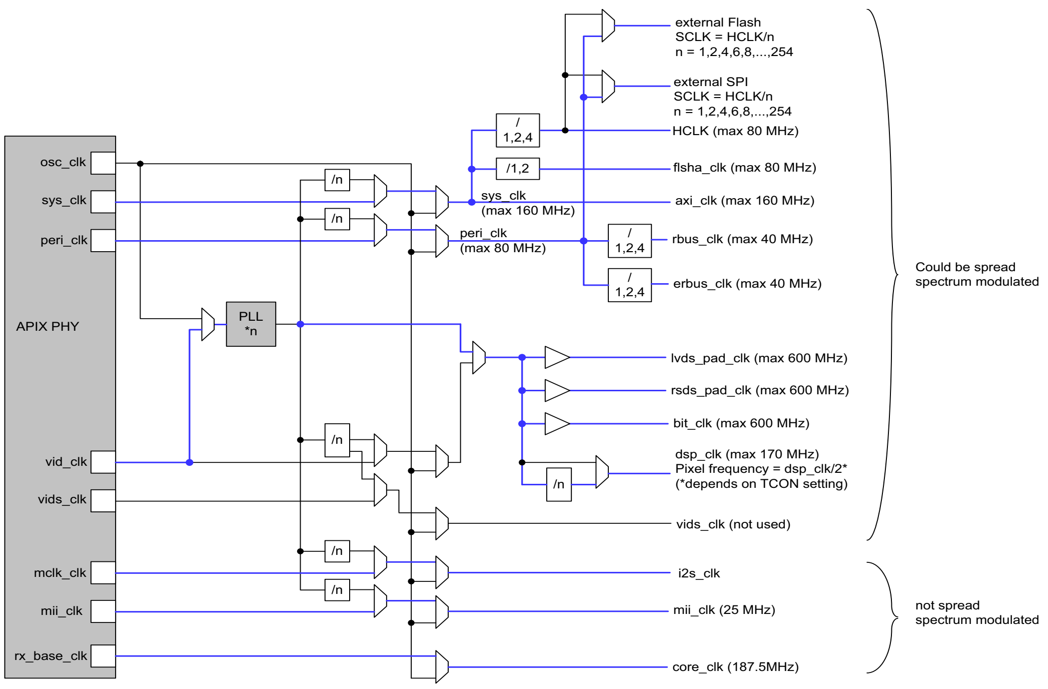


Figure 2-3: Condition after reset



SMC
PWM_baseclk = rbus_clk/n
n = 1,4,5,6,8,10,12,16
PWM_period = PWM_baseclk * 255/511

RLT (32bit)
count_clock = erbus_clk/n;
(n = 1,2,4,8,16,32)

I2C
bitrate = rbus_clk/(n*12+16+y)
(n = 1...63; y = 0...6)

USART/LIN
baudrate = rbus_clk/(n+1); (n < 2^19)

PWM/PPG
count_clock = rbus_clk/n; (n = 1,4,16,64)
period (8bit mode) = count_clock*p8
p8 = 1...255
period (16bit mode) = count_clock*p16
p16 = 1...65535

Figure 2-4: Recommended standard condition

2.4.2 Spread Spectrum Clock Generation

NOTE The maximum spread spectrum value the MB88F33x ‘Indigo2(-x)’ can support is ± 2.5%.

2.4.3 Crystal Oscillator (XTAL)

A 30 MHz crystal oscillator (XTAL) is used for generation of all clocks needed in MB88F33x ‘Indigo2(-x)’. The crystal oscillator is part of the analog APIX PHY unit.

Table 2-2: Oscillator Clock Definition

Clock Name	Frequency Range	Description
osc_clk	30 MHz	Reference Crystal Oscillator Clock. This clock is derived directly from the external XTAL. It is stable (always ON) from the moment the device is powered up AND the external XTAL is working properly and runs independently of the APIX2 PLL. The clock signal is not spread spectrum modulated. The frequency is defined by the external XTAL.

2.4.4 Functional Description

All MB88F33x ‘Indigo2(-x)’ clocks are generated in the APIX2 PHY unit. The frequency and spread spectrum of these clocks can be programmed. A digital clock synthesizer circuit generates these clocks. The clock synthesizer will, on average, generate the programmed frequency, however due to the internal architecture of the clock synthesizer the minimum clock period T_{min} for a programmed clock frequency f_{clock} has to be calculated using:

$$T_{min} = \frac{1}{3000MHz} \cdot \mathrm{INT}\left(\frac{3000MHz}{f_{clock}}\right)$$

Table 2-3: System Clock Definition when using the APIX PHY clock synthesis

Clock Name	Frequency Range	Description
sys_clk	3 MHz to 160 MHz	**AHB, AXI Clock**. This synthesized clock is the master clock for all internal modules. Some internal modules will use divided versions of this clock. This clock can be spread spectrum modulated. The spread spectrum has to be setup in such a way that the next maximum frequency is never exceeded. This clock is stable after APIX2PHY PLL lock is set.
peri_clk	3 MHz to 80 MHz	**Peripherals Clock**. This synthesized clock is the master clock for all peripheral modules. The peripheral modules will use divided versions of this clock. This clock can be spread spectrum modulated. The spread spectrum has to be setup in such a way that the next maximum frequency is never exceeded. This clock is stable after APIX2 PHY PLL lock is set.
vid_clk	3 MHz to 170.0 MHz	**Video Clock**. This synthesized clock is the double video pixel clock. It is used for driving the video output. This clock can be spread spectrum modulated. The spread spectrum has to be set up in such a way that the next maximum frequency is never exceeded. This clock is stable after APIX2 PHY PLL lock is set.
vids_clk	3 MHz to 170.0 MHz	**Shifted Video Clock**. This synthesized clock is a phase shifted version of the video clock. It is used to shift the clock output on the video interface. The minimum shift period is 1.33 ns. One shift step is 333 ps. This clock is stable after APIX2 PHY PLL lock is set.

Several divided internal clocks are generated. The base for these clock dividers are either the synthesized clocks from the APIX PHY or the crystal oscillator clock (osc_clk).

Table 2-4: System Clock Definitions

Clock Name	Frequency Range	Description
axi_clk	3 MHz to 160 MHz	**AXI Bus Clock**. This is the sys_clk.
HCLK	3 MHz to 80 MHz	**AHB Bus Clock**. This is either the axi_clk (if the axi_clk is not more than 83 MHz) or 1/2 or 1/4 of the axi_clk.
flash_clk	3 MHz to 80 MHz	**Flash Clock**. This is either the sys_clk (if the sys_clk is not more than 80 MHz) or 1/2 of the sys_clk.
rbus_clk	3 MHz to 40 MHz	**R-BUS Clock**. This is a divided version (divide by 1,2,4) of the peri_clk. This clock is limited to 40 MHz
erbus_clk	3 MHz to 40 MHz	**eR-BUS Clock**. This is a divided version (divide by 1,2,4) of the peri_clk. This clock is limited to 40 MHz.

There are several possibilities to generate the clocks for the display subsystem. For a detailed description of the display clock setup, refer to the section ‘Clock Setup of Iris-MVL’ in the Hardware Manual of the MB88F33x ‘Indigo2(-x)’.

Clock Name	Frequency Range	Description
bit_clk	3 MHz to 600 MHz	**Bit Clock**. The frequency of bit_clk defines the accuracy of the output signal shifting in TTL and RSDS mode and it is the bit clock in LVDS mode.
dsp_clk	3 MHz to 170 MHz	**Display Clock**. This is the input clock to the IRIS graphic subsystem. This clock is either the double display pixel clock or the pixel clock.

The clocks that belong to the different APIX interfaces come from the APIX RX link. The frequency of these clocks depends on the APIX configuration. These clocks are either synthesized with a digital clock synthesizer or are regenerated with a CDR (clock data recovery) circuit.

Table 2-5: APIX clock definitions

Clock Name	Frequency Range	Description
core_clk	Up to 190 MHz (nominal 187.5 MHz or 125 MHz)	**APIX Core Clock.** This is the clock for the APIX data. It is 187.5 MHz for APIX2 mode and 125 MHz for APIX1 mode. It is not spread spectrum modulated. This clock is generated with a CDR circuit
mii_clk	25 MHz	**MII Clock.** This is the clock of the MII interface. It is not spread spectrum modulated. It is used in the APIX block for the Embedded Ethernet module.
i2s_mclk	54 MHz	**Audio clock** (Master clock). This is the clock of the I2S interface. It is not spread spectrum modulated. This is a synthesized clock.

As with the system master clocks, these clocks can also be generated by the internal PLL or the osc_clk can be used.

2.4.5 Clock Synthesis

For every one of the three clock outputs of the clock synthesis (sys_clk, peri_clk or vid_clk) the frequency can be programmed using this equation:

$$\text{xxx\_clk\_pw} = \frac{(3000\ \text{MHz}) \cdot (2)^6}{f_{\text{xxx\_clk}}}$$

whereby xxx is either sys, peri or vid. The vids_clk has the same frequency as the vid_clk and is shifted by vids_clk_delay * 333 ps.

Example:

A frequency of 40 MHz needs to be generated.

$$\text{xxx\_clk\_pw} = \frac{(3000\ \text{MHz}) \cdot (2)^6}{40\ \text{MHz}}$$

$$\text{xxx\_clk\_pw} = 4800$$

The clock sythesis generates the programmed frequency on average. For this it jumps between two discrete frequencies. The maximum frequency can be calculated by this equation:

$$f_{\text{xxx\_clk}}(\text{MAX}) = \frac{3000\ \text{MHz}}{\text{INT}\left(\frac{\text{xxx\_clk\_pw}}{2^6}\right)}$$

NOTE After reset, the clock synthesis module does not use the reset value of the control register for clock generation in the clock synthesis. Therefore, it is necessary to re-write the reset value or to write an updated value, before using the clock synthesis.

2.4.6 Clock Modulation / Spread Spectrum

The synthesized clocks of the APIX2 clock synthesis can be spread spectrum modulated. A center spread, dual triangle shape is used to modify the pulse width and to archive a spread spectrum characteristic. The modulation frequency is fixed at f_{MOD} = 45.776 kHz, or a period of t_{MOD} = 21.845 µs, respectively.

The MB88F33x 'Indigo2(-x)' can support a maximum spread spectrum value of ± 2.5%.

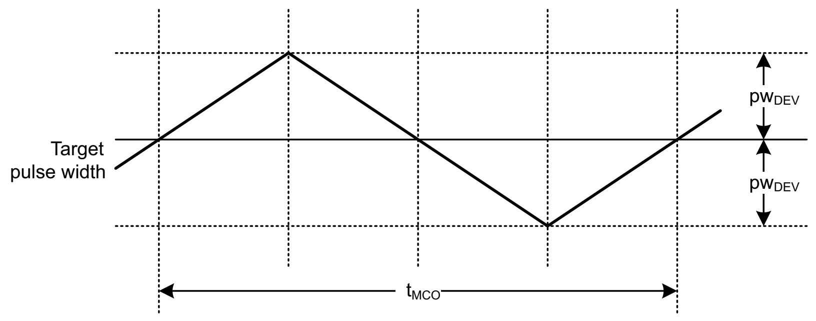


Figure 2-5: Spread spectrum modulation

The spread spectrum modulation can be enabled with xxx_clk_mod_en. The modulation amplitude can be programmed with the xxx_mod_step value. The value of xxx_mod_step can be calculated with this equation:

$$\text{xxx\_mod\_step} = \frac{(\text{modulation\_amplitude}) \cdot (\text{xxx\_clk\_pw})}{16}$$

Example:

For a frequency of 40 MHz a modulation with ± 2% needs to be generated.

$$\text{xxx\_mod\_step} = \frac{0.02 \cdot 4800}{16}$$

$$\text{xxx\_mod\_step} = 6$$

Depending on the modulation amplitude, the center frequency has an error factor compared to the non modulated frequency. The center frequency is slightly higher.

Figure 2-6 shows the expected error in relation to the modulation amplitude.

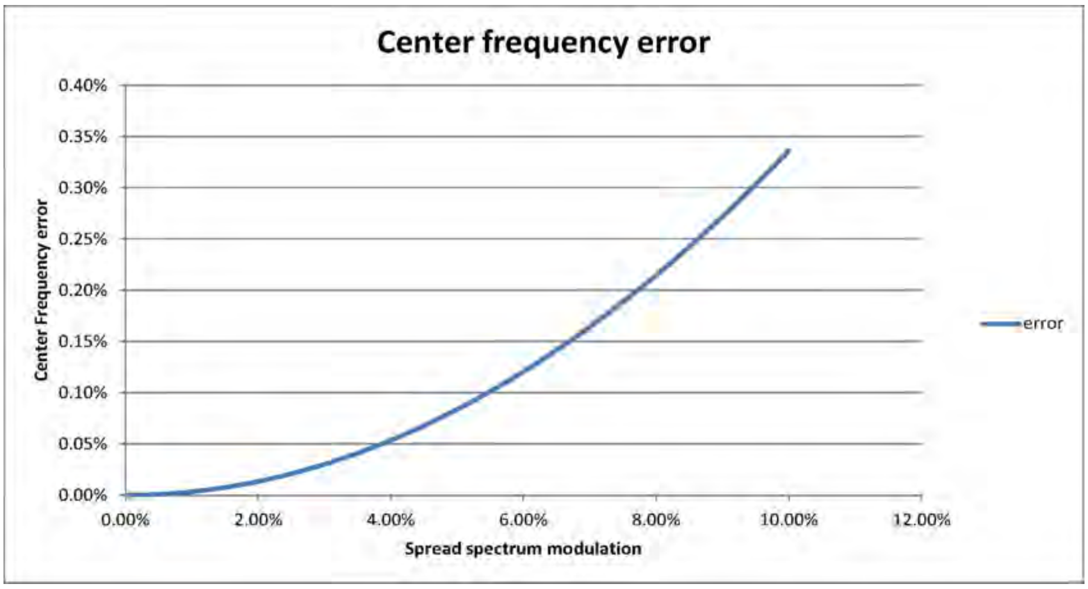


Figure 2-6: Spread Spectrum Frequency Error

The maximum and minimum output frequency can be calculated with these equations:

$$f_{\text{xxx\_clk\_max}} = \frac{3000MHz \cdot 2^6}{\text{xxx\_clk\_pw} - (16 \cdot \text{xxx\_mod\_step})}$$

$$f_{\text{xxx\_clk\_min}} = \frac{3000MHz \cdot 2^6}{\text{xxx\_clk\_pw} + (16 \cdot \text{xxx\_mod\_step})}$$

Based on the equations above the clock synthesis can also be setup for a up- or down- spread operation.

The clock synthesis generates the programmed frequency on average. For this it jumps between discrete frequencies. The absolute maximum frequency which can result with spread spectrum can be calculated by this equation:

$$f_{xxx\_clk\_max}(\mathrm{MAX}) = \frac{3000\ \mathrm{MHz}}{\mathrm{INT}\left(\frac{xxx\_clk\_pw - 16 \cdot xxx\_mod\_step}{2^6}\right)}$$

2.4.7 Internal Display PLL

An internal PLL can be used to generate a high frequent bit_clk. The PLL has to be enabled (PLL_CTRL.pll) and the PLL input is either the oscillator clock or the vid_clk, which is generated in the APIX2PHY (PLL_CTRL.pll_clksel).

The input to the PLL has to be in the range of 20MHz to 50MHz. The output frequency is an integer multiplication of the input frequency and has to be in the range of 300MHz to 600MHz.

The multiplication is set via register PLL_CTRL.pll_idiv. All other control bits of the PLL have to stay in their reset state.

2.5 Reset

The MB88F33x ‘Indigo2(-x)’ can be reset with the external reset input RESET_N. The reset is low active and will reset the entire chip. For timings, please see chapter “7.8 Reset Timing”. The external reset will latch the configuration signals (see section “2.6 Bootstrap Configuration”).

Internally, the reset input is distributed into several sub-blocks. The internal reset signals will not be released until a stable clock is present. It is possible to do a software reset of some of these sub-blocks.

2.5.1 Power On Reset

MB88F33x ‘Indigo2(-x)’ does not have an internal power-on reset circuit.

2.6 Bootstrap Configuration

Two bootstrap configuration pins are latched with a hardware reset. The configuration pins then allow an application to replace some chip internal default values and therefore, to define the system power up configuration.

After power up MB88F33x ‘Indigo2(-x)’ can boot in 4 different operation modes (configurations).

Please refer to section “System Power-up” and “Boot Procedure” for more details about Bootstrap modes.

Table 2-6: Bootstrap register settings

Pin	Function	Comment
CFG_1, CFG_0	Command sequencer boot mode select b00: Disable command sequencer boot sequence b01: Command sequencer starts with boot sequence from internal Flash b10: Power up with default APIX mode (500Mbit/s APIX2 mode) b11: Power up with default APIX mode (500Mbit/s APIX2 mode) an execute boot sequence from internal Flash afterwards. **NOTE** See App Note “USING BOOTMODE 2 & 3” for detailed information about the boot modes.	Internal pull-down
CFG_2	b0:Set to 0 b1:Reserved	Internal pull-down
CFG_3	APIX oscillator mode b0: external crystal oscillator b1: clock input (via XI pin)	Internal pull-down
CFG_4	b0:Set to 0 b1:Reserved	Internal pull-down
CFG_5	Not used Can be read by Command Sequencer and used for selection of different boot sequences	Internal pull-down

2.7 Failure Unit

2.7.1 Panic Switch

The ‘Panic Switch’ is used by the Iris display engine to switch into panic mode. The behavior of the Iris display engine when in panic mode, is programmable. Either a constant color (black) is output, or a predefined image in memory (e.g. ‘NO SIGNAL’) is displayed. In parallel, the panic switch can stop the alive sender, trigger a special panic command sequence and send an APIX interrupt message to the host MCU.

If enabled, the panic switch circuit supervises all the interrupt status signals of the internal HOST_INT interrupt controller. There is a dedicated enable signal for each interrupt status bit. If the interrupt input is enabled, the panic switch is asserted when an interrupt is issued. The panic switch has to be cleared by software writing to a register. The panic switch can be triggered by software for test purposes.

2.7.2 Alive Sender

When enabled, the alive sender sends a periodic signal (similar to an interrupt) to both the APIX AShell and APIX Ethernet remote handlers. The periodic signal can be masked by one or several status or interrupt signals. The mask signals are latched in the alive sender and need to be reset by software if they were issued.

The base for the periodic signal can be either:

- A programmable frame interrupt 0 from the Iris unit (vsync interrupt)
- Any one of the 4 signature measurement interrupts
- The output pulse of reload timer 8

Possible masking signals are:

- System watchdog trigger or low voltage error
- Command Sequencer watchdog error and Command Sequencer error (illegal command)
- Panic Switch
- Any one of the 4 signature error interrupts
- Any one of the 8 external interrupts
- Any one of the 3 Iris sync error interrupts
- Any one of the APIX interface errors
- Configuration FIFO error

2.7.3 Low Voltage Detection (LVD)

The LVD block supervises the core supply voltages and the 5V GPIO supply voltage. If the voltage drops below or climbs above a programmable threshold, an interrupt can be issued. The LVDL part supervises the 1.2 V core voltage and the LVDH part supervises the 5V GPIO voltage. The threshold has a hysteresis to avoid the signal toggling.

Possible settings can be found in Table 2-7 . The threshold has a hysteresis to avoid toggling of the signal.

Table 2-7: Possible settings

Parameter	Min	Typ	Max	Unit	Comment
LVDH	2.0	2.2	2.4	V	SVH setting=0
	2.2	2.4	2.6	V	SVH setting=1
	2.4	2.6	2.8	V	SVH setting=3
	2.5	2.7	2.9	V	SVH setting=2
	3.5	3.7	3.9	V	SVH setting=6
	3.7	3.9	4.1	V	SVH setting=7
	3.9	4.1	4.3	V	SVH setting=5
	4.1	4.3	4.5	V	SVH setting=4
LVDL	0.8	0.9	1.0	V	SVL setting=6
	0.9	1.0	1.1	V	SVL setting=7
	1.0	1.1	1.2	V	SVL setting=5
	1.1	1.2	1.3	V	SVL setting=4

2.7.4 System Watchdog

A system watchdog is implemented to detect a loss of communication to the host. This could occur if the APIX link is down or the host CPU hangs. In the event of such errors, the watchdog will generate an interrupt. This interrupt can be used for the panic switch (See “Panic Switch” on page 18.) or to start a predefined error command sequence (see section “Command Sequencer”).

2.7.4.1 Functional Description

The system watchdog is a 28bit counter, which is decremented when ever the pre-divider counter is zero. For the pre-divider counter the AHB clock is used. The pre-divider start value is programmable and can be from 20 to 215. When the system watchdog counter reaches the zero value an error interrupt is issued. The system watchdog counter and the pre-divider counter can be reset to their start value when writing the wdg_reset bit. When the system watchdog is reset before the system watchdog counter reached a programmable reset window start value also an error interrupt is send to the system. The system watchdog can be disabled and for test it is possible to force a system watchdog error.

2.8 Interrupt Controller

The interrupt controller acts as a collection point for all the possible interrupt sources and status signals in the MB88F33x ‘Indigo2(-x)’ system. It combines the interrupts to one host interrupt which goes to a dedicated pin (HOST_INT). Additionally, it selects 7 possible interrupts for the command sequencer, it can generate DMA requests based on interrupts and it routes all possible interrupts (as an event) to the AShell remote handler and to the ‘Ethernet over APIX’ remote handler.

2.8.1 Interrupt Handling

The different sub-units in the MB88F33x ‘Indigo2(-x)’ generate two types of interrupts (either an edge interrupt or a level interrupt).

For all edge interrupts, the interrupt controller latches the rising edge of the interrupt line and generates a interrupt status signal. This interrupt status signal can be cleared in the interrupt controller.

For all level interrupts, the interrupt controller uses the level interrupt input as an interrupt status. These level interrupt input signals have to be cleared in the dedicated sub-module. For the status input signals, the interrupt controller will generated an edge interrupt, if the status is going high and/or low. This generated edge interrupt is then used like any other edge interrupt in the controller.

The state of all interrupts status signals can be read from the different interrupt controller status register.

2.8.2 HOST_INT Output

Every interrupt status signal can be masked using an enable bit. After masking, all signals are combined to a single HOST_INT signal. Additionally after masking, several groups of interrupts are combined and can be read by the software. This makes it possible to read one register and obtain an overview of which interrupt groups have generated the host interrupt.

2.8.3 Command Sequencer Interrupts

The command sequencer can handle 7 different interrupts. The interrupt controller has the possibility to route every possible interrupt input to any one of the command sequencer interrupts. The command sequencer interrupts do not use the interrupt status signals that are latched in the interrupt controller. Instead, the interrupt signals from the sub-units themselves are used directly.

2.8.4 Remote Handler Events

The remote handlers in the AShell and the ‘Ethernet over APIX’ blocks can handle up to 255 different events. An interrupt message is send to the host MCU (which is connected via the APIX link) for every event. The interrupt controller provides a vector with all the possible interrupt sources to both remote handler units.

2.8.5 DMA Controller Requests

The interrupt controller can generate two different DMA requests for the DMA controller.These DMA requests are generated when the controller receives an rising edge on an interrupt input. The interrupt that is used, is programmable.

2.8.6 Interrupt Table

A list of all possible interrupts can be found in the following table. The different interrupts are cleared in different places.

Table 2-8: Interrupts

ID	Name	Kind	Description
APIX Interrupt Group			
0	APIX_LINK_FUNC	Level	APIX link functional
1	APIX_LINK_ERR	Level	APIX link error
2	APIX_LINK_FATAL	Level	APIX link fatal error
3	APIX_ASHELL_REQ	Level	APIX Ashell request
4	APIX_ASHELL_FUNC	Level	APIX Ashell functional
5	APIX_ASHELL_ERR	Level	APIX Ashell error
6	APIX_ASHELL_FATAL	Level	APIX Ashell fatal error
7	APIX_PIX_ERR	Level	APIX Ashell Pixel error
8	APIX_PIX_FATAL	Level	APIX Ashell Pixel fatal error
9	APIX_PHY_ARS	Rise	APIX PHY recal request
10	APIX_PHY_RES	Fall	APIX PHY reset request
11	APIX_PHY_NC1	Level	APIX PHY interface (not connected)
12	APIX_PHY_NC2	Level	APIX PHY interface (not connected)
13	APIX_HDCP_FUNC	Level	APIX HDCP functional
14	APIX_HDCP_ERR	Level	APIX HDCP error
ASHELL_RH Interrupt Group			
15	ARH_MAIL_REQ	Pulse	Ashell Remote Handler Mailbox request interrupt
16	ARH_MAIL_ACK	Pulse	Ashell Remote Handler Mailbox request done interrupt
17	ARH_PUSH_REQ	Pulse	Ashell Remote Handler Push message request interrupt
18	ARH_PUSH_ACK	Pulse	Ashell Remote Handler Push message request done interrupt
19	ARH_RERR	Pulse	Ashell Remote Handler AHB bus read bus error interrupt
20	ARH_WERR	Pulse	Ashell Remote Handler AHB bus write bus error interrupt
21	ARH_WRLOCK	Pulse	Ashell Remote Handler RX interrupt, receive write message while locked
22	ARH_R_THRESH	Pulse	Ashell Remote Handler RX-fifo threshold reached
23	ARH_R_OVL	Pulse	Ashell Remote Handler RX-fifo overflow (loss of message)
24	ARH_T_THRESH	Pulse	Ashell Remote Handler TX-fifo threshold reached
25	ARH_T_OVL	Pulse	Ashell Remote Handler TX-fifo overflow (loss of message)
26	ARH_T_TOUT	Pulse	Ashell Remote Handler TCTRL timeout (loss of message)
E2IP Interrupt Group			
27	ERH_MAIL_REQ	Pulse	E2IP Remote Handler Mailbox request interrupt
28	ERH_MAIL_ACK	Pulse	E2IP Remote Handler Mailbox request done interrupt
29	ERH_PUSH_REQ	Pulse	E2IP Remote Handler Push message request interrupt
30	ERH_PUSH_ACK	Pulse	E2IP Remote Handler Push message request done interrupt
31	ERH_RERR	Pulse	E2IP Remote Handler AHB bus read bus error interrupt
32	ERH_WERR	Pulse	E2IP Remote Handler AHB bus write bus error interrupt
33	ERH_WRLOCK	Pulse	E2IP Remote Handler RX interrupt, receive write message while locked
34	ERH_R_THRESH	Pulse	E2IPRemoteHandlerRX-fifothreshold reached
35	ERH_R_OVL	Pulse	E2IP Remote Handler RX-fifo overflow (loss of message)
36	ERH_T_THRESH	Pulse	E2IPRemoteHandlerTX-fifothreshold reached
37	ERH_T_OVL	Pulse	E2IP Remote Handler TX-fifo overflow (loss of message)
38	ERH_T_TOUT	Pulse	E2IP Remote Handler TCTRL timeout (loss of message)
39	E2IP_RX_DROP	Pulse	E2IP RX frame dropped
40	E2IP_TX_DROP	Pulse	E2IP TX frame dropped
41	E2IP_RX_OVWR	Pulse	E2IP RX frame dropped, while not already processed
42	E2IP_MAC0_UDT	Pulse	E2IP MAC address of Host 0 updated
43	E2IP_MAC1_UDT	Pulse	E2IP MAC address of Host 1 updated

Table 2-8: Interrupts (Continued)

ID	Name	Kind	Description
CFF_CTRL Interrupt Group			
44	CFF_ALL	Pulse	Combination of all Config FIFO interrupts
45	CFF_RERR	Pulse	Config FIFO AHB Master received ERROR response interrupt
46	CFF_DW7	Pulse	Config FIFO Data written channel 7 interrupt
47	CFF_DW6	Pulse	Config FIFO Data written channel 6 interrupt
48	CFF_DW5	Pulse	Config FIFO Data written channel 5 interrupt
49	CFF_DW4	Pulse	Config FIFO Data written channel 4 interrupt
50	CFF_DW3	Pulse	Config FIFO Data written channel 3 interrupt
51	CFF_DW2	Pulse	Config FIFO Data written channel 2 interrupt
52	CFF_DW1	Pulse	Config FIFO Data written channel 1 interrupt
53	CFF_DW0	Pulse	Config FIFO Data written channel 0 interrupt
CFF_FIFO Interrupt Group			
54	CFF_UFLW7	Pulse	Config FIFO Underflow channel 7 interrupt
55	CFF_OFLW7	Pulse	Config FIFO Overflow channel 7 interrupt
56	CFF_UTHD7	Pulse	Config FIFO Upper Threshold channel 7 interrupt
57	CFF_LTHD7	Pulse	Config FIFO Lower Threshold channel 7 interrupt
58	CFF_UFLW6	Pulse	Config FIFO Underflow channel 6 interrupt
59	CFF_OFLW6	Pulse	Config FIFO Overflow channel 6 interrupt
60	CFF_UTHD6	Pulse	Config FIFO Upper Threshold channel 6 interrupt
61	CFF_LTHD6	Pulse	Config FIFO Lower Threshold channel 6 interrupt
62	CFF_UFLW5	Pulse	Config FIFO Underflow channel 5 interrupt
63	CFF_OFLW5	Pulse	Config FIFO Overflow channel 5 interrupt
64	CFF_UTHD5	Pulse	Config FIFO Upper Threshold channel 5 interrupt
65	CFF_LTHD5	Pulse	Config FIFO Lower Threshold channel 5 interrupt
66	CFF_UFLW4	Pulse	Config FIFO Underflow channel 4 interrupt
67	CFF_OFLW4	Pulse	Config FIFO Overflow channel 4 interrupt
68	CFF_UTHD4	Pulse	Config FIFO Upper Threshold channel 4 interrupt
69	CFF_LTHD4	Pulse	Config FIFO Lower Threshold channel 4 interrupt
70	CFF_UFLW3	Pulse	Config FIFO Underflow channel 3 interrupt
71	CFF_OFLW3	Pulse	Config FIFO Overflow channel 3 interrupt
72	CFF_UTHD3	Pulse	Config FIFO Upper Threshold channel 3 interrupt
73	CFF_LTHD3	Pulse	Config FIFO Lower Threshold channel 3 interrupt
74	CFF_UFLW2	Pulse	Config FIFO Underflow channel 2 interrupt
75	CFF_OFLW2	Pulse	Config FIFO Overflow channel 2 interrupt
76	CFF_UTHD2	Pulse	Config FIFO Upper Threshold channel 2 interrupt
77	CFF_LTHD2	Pulse	Config FIFO Lower Threshold channel 2 interrupt
78	CFF_UFLW1	Pulse	Config FIFO Underflow channel 1 interrupt
79	CFF_OFLW1	Pulse	Config FIFO Overflow channel 1 interrupt
80	CFF_UTHD1	Pulse	Config FIFO Upper Threshold channel 1 interrupt
81	CFF_LTHD1	Pulse	Config FIFO Lower Threshold channel 1 interrupt
82	CFF_UFLW0	Pulse	Config FIFO Underflow channel 0 interrupt
83	CFF_OFLW0	Pulse	Config FIFO Overflow channel 0 interrupt
84	CFF_UTHD0	Pulse	Config FIFO Upper Threshold channel 0 interrupt
85	CFF_LTHD0	Pulse	Config FIFO Lower Threshold channel 0 interrupt
RLT Interrupt Group			
86	RLT0	Level	Reload timer 0 interrupt
87	RLT1	Level	Reload timer 1 interrupt
88	RLT2	Level	Reload timer 2 interrupt
89	RLT3	Level	Reload timer 3 interrupt
90	RLT4	Level	Reload timer 4 interrupt
91	RLT5	Level	Reload timer 5 interrupt
92	RLT6	Level	Reload timer 6 interrupt

Table 2-8: Interrupts (Continued)

ID	Name	Kind	Description
93	RLT7	Level	Reload timer 7 interrupt
94	RLT8	Level	Reload timer 8 interrupt
95	RLT9	Level	Reload timer 9 interrupt
96	RLT10	Level	Reload timer 10 interrupt
97	RLT11	Level	Reload timer 11 interrupt
98	RLT12	Level	Reload timer 12 interrupt
99	RLT13	Level	Reload timer 13 interrupt
100	RLT14	Level	Reload timer 14 interrupt
101	RLT15	Level	Reload timer 15 interrupt
LIN Interrupt Group			
102	LIN_R	Level	LIN Reception interrupt
103	LIN_T	Level	LIN Transmission interrupt
104	LIN_E	Level	LIN Error interrupt
PPG Interrupt Group			
105	PPG00	Level	PPG / PWM module 0 interrupt 0
106	PPG01	Level	PPG / PWM module 0 interrupt 1
107	PPG02	Level	PPG / PWM module 0 interrupt 2
108	PPG03	Level	PPG / PWM module 0 interrupt 3
109	PPG10	Level	PPG / PWM module 1 interrupt 0
110	PPG11	Level	PPG / PWM module 1 interrupt 1
111	PPG12	Level	PPG / PWM module 1 interrupt 2
112	PPG13	Level	PPG / PWM module 1 interrupt 3
113	PPG20	Level	PPG / PWM module 2 interrupt 0
114	PPG21	Level	PPG / PWM module 2 interrupt 1
115	PPG22	Level	PPG / PWM module 2 interrupt 2
116	PPG23	Level	PPG / PWM module 2 interrupt 3
117	PPG30	Level	PPG / PWM module 3 interrupt 0
118	PPG31	Level	PPG / PWM module 3 interrupt 1
119	PPG32	Level	PPG / PWM module 3 interrupt 2
120	PPG33	Level	PPG / PWM module 3 interrupt 3
I2C0 Interrupt Group			
121	I2C0_IRQ	Level	I2C0 Operational interrupt
122	I2C0_ERIRQ	Level	I2C0 Error interrupt
I2C1 Interrupt Group			
123	I2C1_IRQ	Level	I2C1 Operational interrupt
124	I2C1_ERIRQ	Level	I2C1 Error interrupt
SGE Interrupt Group			
125	SGE_IRQ	Level	Sound generator interrupt
126	SGE_RLD	Pulse	Sound generator register reload interrupt
ADC Interrupt Group			
127	ADC_IRQ	Level	ADC Conversion end interrupt
128	ADC2_IRQ	Level	ADC Scan end interrupt
129	ADC_RCOIRQ	Level	ADC Range comparator interrupt
130	ADC_ADPIRQ	Level	ADC pulse detection interrupt
EIRQ Interrupt Group			
131	EIRQ_0	Level	external IRQ pin 0 interrupt
132	EIRQ_1	Level	external IRQ pin 1 interrupt
133	EIRQ_2	Level	external IRQ pin 2 interrupt
134	EIRQ_3	Level	external IRQ pin 3 interrupt
135	EIRQ_4	Level	external IRQ pin 4 interrupt
136	EIRQ_5	Level	external IRQ pin 5 interrupt
137	EIRQ_6	Level	external IRQ pin 6 interrupt

Table 2-8: Interrupts (Continued)

ID	Name	Kind	Description
138	EIRQ_7	Level	external IRQ pin 7 interrupt
ESPI Interrupt Group			
139	ESPI_RX	Level	External device SPI Reception interrupt
140	ESPI_TX	Level	External device SPI Transmission interrupt
141	ESPI_FAULT	Level	External device SPI Fault interrupt
IRIS Interrupt Group			
142	IRS_PE_SC0	Pulse	Iris-MVL pixel engine sequence complete (synchronizer 0)
143	IRS_PE_SC1	Pulse	Iris-MVL pixel engine sequence complete (synchronizer 1)
144	IRS_PE_FC0	Pulse	Iris-MVL pixel engine frame complete (extdst0)
145	IRS_PE_FC1	Pulse	Iris-MVL pixel engine frame complete (extdst1)
146	IRS_LB0_SL	Pulse	Iris-MVL layerblend 0 shadow register loaded
147	IRS_LB1_SL	Pulse	Iris-MVL layerblend 1 shadow loaded
148	IRS_DE_SL	Pulse	Iris-MVL display engine top shadow loaded
149	IRS_DE_SC	Pulse	Iris-MVL display engine sequence complete
150	IRS_FG_P0	Pulse	Iris-MVL frame generator programmable interrupt 0
151	IRS_FG_P1	Pulse	Iris-MVL frame generator programmable interrupt 1
152	IRS_FG_P2	Pulse	Iris-MVL frame generator programmable interrupt 2
153	IRS_FG_P3	Pulse	Iris-MVL frame generator programmable interrupt 3
154	IRS_FG_SL_P	Pulse	Iris-MVL frame generator shadow register loaded (primary input)
155	IRS_FG_SL_S	Pulse	Iris-MVL frame generator shadow register loaded (secondary input)
156	IRS_SIG0_SL	Pulse	Iris-MVL signature unit 0 shadow loaded
157	IRS_SIG0_RDY	Pulse	Iris-MVL signature unit 0 measurement complete
158	IRS_SIG0_ERR	Pulse	Iris-MVL signature unit 0 signature error
159	IRS_SIG1_SL	Pulse	Iris-MVL signature unit 1 shadow loaded
160	IRS_SIG1_RDY	Pulse	Iris-MVL signature unit 1 measurement complete
161	IRS_SIG1_ERR	Pulse	Iris-MVL signature unit 1 signature error
162	IRS_SIG2_SL	Pulse	Iris-MVL signature unit 2 shadow loaded
163	IRS_SIG2_RDY	Pulse	Iris-MVL signature unit 2 measurement complete
164	IRS_SIG2_ERR	Pulse	Iris-MVL signature unit 2 signature error
165	IRS_SIG3_SL	Pulse	Iris-MVL signatureunit 3 shadow loaded
166	IRS_SIG3_RDY	Pulse	Iris-MVL signature unit 3 measurement complete
167	IRS_SIG3_ERR	Pulse	Iris-MVL signature unit 3 signature error
168	IRS_FG_SYNC_P	Rise	Iris-MVL frame generator synchronization stable (primary input)
169	IRS_FG_SYNCERR_P	Fall	Iris-MVL frame generator synchronization loss (primary input)
170	IRS_FG_SYNC_S	Rise	Iris-MVL frame generator synchronization stable (secondary input)
171	IRS_FG_SYNCERR_S	Fall	Iris-MVL frame generator synchronization loss (secondary input)
172	IRS_FC_SYNC	Rise	Iris-MVL frame capture synchronization stable
173	IRS_FC_SYNCERR	Fall	Iris-MVL frame capture synchronization loss
CMDSEQ Interrupt Group			
174	CMDSEQ_WDG	Pulse	Command Sequencer watchdog interrupt (watchdog status)
175	CMDSEQ_SWINT	Pulse	Command Sequencer software interrupt
176	CMDSEQ_LWM	Pulse	Command Sequencer command buffer low watermark interrupt (counter reaches low water mark)
177	CMDSEQ_HWM	Pulse	Command Sequencer command buffer high watermark interrupt (counter reaches high water mark)
178	CMDSEQ_ERROR	Rise	Command Sequencer error interrupt (error on illegal instruction)
179	CMDSEQ_HALT	Rise	Command Sequencer halt interrupt (core is in halt state)
180	CMDSEQ_EMPTY	Rise	Command Sequencer command buffer fifo empty interrupt
181	CMDSEQ_FULL	Rise	Command Sequencer command buffer fifo full interrupt
GC Interrupt Group			
182	GC_ALV	Pulse	Global Control Alive sender IRQ
183	GC_WDG	Pulse	System Watchdog interrupt

Table 2-8: Interrupts (Continued)

ID	Name	Kind	Description
184	LVD_L_R	Rise	Low voltage detection core voltage low thresh old comparator going high interrupt
185	LVD_L_F	Fall	Low voltage detection core voltage low thresh old comparator going low interrupt
186	LVD_H_R	Rise	Lowvoltage detection core voltage high threshold comparator going high interrupt
187	LVD_H_F	Fall	Lowvoltage detection core voltage high threshold comparator going low interrupt
188	PANIC_SWITCH	Level	Panic switch was asserted
189	HIFC	Level	Host interface AHB bus error interrupt
190	FLSH	Level	Flash interface interrupt (ready, hang or single bit error)
DMAC Interrupt Group			
191	DMAC_DIRQ	Level	DMA Controller single ORed output of all the DIRQx generated from each Channel
192	DMAC_DIRQ0	Level	DMA Controller end of DMA transfer channel 0
193	DMAC_DIRQ1	Level	DMA Controller end of DMA transfer channel 1
194	DMAC_EIRQ	Level	DMA Controller single ORed output of all the EIRQx generated from each Channel
195	DMAC_EIRQ0	Level	DMA Controller error DMA channel 0
196	DMAC_EIRQ1	Level	DMA Controller error DMA channel 1
FSPI Interrupt Group			
197	FSPI_RX	Level	External Flash SPI Reception interrupt
198	FSPI_TX	Level	External Flash SPI Transmission interrupt
199	FSPI_FAULT	Level	External Flash SPI Fault interrupt
PRGCRC Interrupt Group			
200	PRGCRC_IRQ	Level	Programmable CRC completion interrupt
INTERCONNECT Interrupt Group			
201	RBUS_BUSERR	Level	RBUS interconnect error (signaled by RBUS error collection unit)
202	ERBUS_BUSERR	Level	eRBUS interconnect error (signaled by eRBUS error collection unit)
203	EXTIRQ_BUSERR	Pulse	External IRQ unit signals AHB interface error
204	ESPI_BUSERR	Pulse	External device SPI unit signals AHB interface error
205	FSPI_BUSERR	Pulse	External Flash SPI unit signals AHB interface error
206	PRGCRC_BUSERR	Pulse	Progammable CRC unit signals AHB interface error
207	IFLASH_BUSERR	Pulse	Internal Flash interface signals AHB interface error
208	IFLASH_TCBUSERR	Pulse	Internal Flash interface signals TC interface error

2.9 Software Interface

All register accesses to global control registers are protected and must be enabled by writing a signature into the register LockUnlock before they can be used.

The base address for the Global Control register is 0x00000000.

The Global Control register block is also mirrored on base addresses 0x0000x000 and 0x0001x000.

For locking of the global control register the lock key = 0xa82be775 has to be written. For unlocking of the global control register the unlock key = 0x69b309b8 has to be written.

The Indigo2 devices have different Chip IDs.

Table 2-9: Chip ID

Name	**Chip ID**	**Description**
MB88F334 ‘Indigo2’	0x88463340	Fully-featured device
MB88F335 ‘Indigo2-S’	0x88463350	Scaled-down device, various features have been stripped out or reduced.
MB88F336 ‘Indigo2-N’	0x88463360	Fully-featured device, but without HDCP functionality.

2.10 System Power-up

After power up, the MB88F33x ‘Indigo2(-x)’ can be booted into one of four different operation modes (configurations).

Table 2-10: System Power Up

Mode	Description
OFF (MODE 0)	Command sequencer in halt state after Power up reset. The command sequencer needs to be restarted with a different boot mode by the host software.
CMDSEQ (MODE 1)	The command sequencer starts with a boot sequence from internal flash memory.
DEFAULT (MODE 2)	The command sequence will execute a default power up sequence. This default sequence will initialize the APIX input with mode (500 Mbit/s APIX2 mode). The rest of the system is configured in a way that it is possible to receive remote handler commands on the APIX interface. This can be used to program a user defined sequence into the flash memory. **Note: Please contact the Fujitsu Application Support Group and ask for the correct parameter values.**
APIX2_500M (MODE 3)	As with default mode, the command sequencer will set up the system in a default mode. After this, the command sequencer will execute a user defined command sequence from the flash memory.

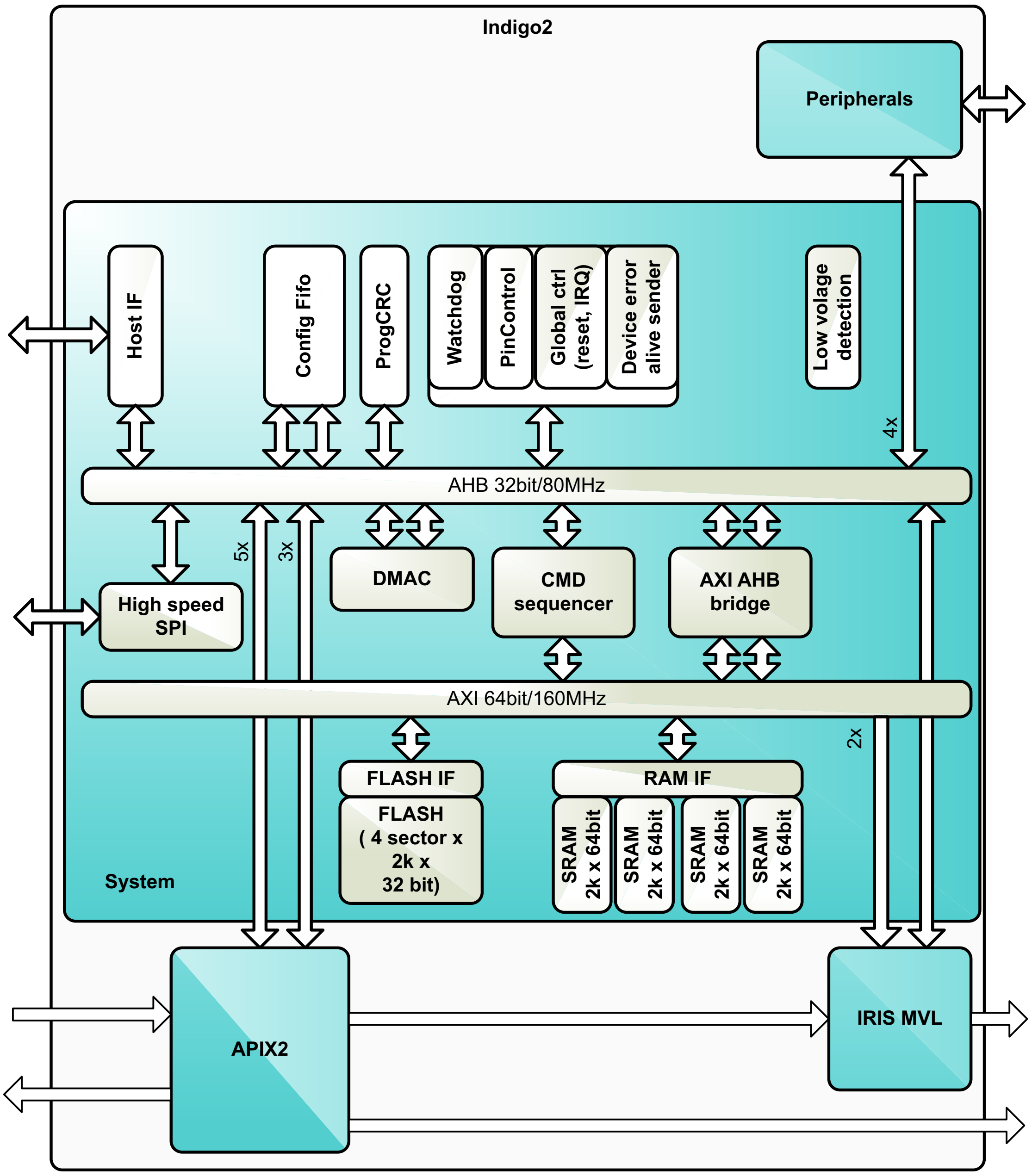
The current operation mode is defined by two bootstrap pins, which are latched during reset (see also Section 2.6). Please refer to Figure 3-23, for more information.

This page intentionally left blank

Chapter 3: System

3.1 General

The following block diagram shows the most important components of the MB88F33x ‘Indigo2(-x)’ device.



System block diagram

3.2 Host Interface

The Host Interface module enables an external host CPU to connect to the internal registers and memories of MB88F33x ‘Indigo2(-x)’ via the HOST Interface protocol (a SPI based interface, further referred as "SPI interface"). This allows a host CPU to read and write to the internal modules.

From a host CPU point of view, this module functions as a slave, whereas internally it functions as a master.

NOTE Please refer to the Application Note “Device Setup and Fujitsu Developer Suite”

3.2.1 Features

Accesses by the host CPU to internal modules can be made using varying address byte lengths in a range of 1 to 4 bytes. Additionally, the data byte length can be arbitrarily set within a range of 1 to 16 bytes. This means that the received number of bytes can be optimized and forwarding can be done efficiently. These settings can be specified by the CMD byte, allowing a highly flexible solution that abstracts the type of host CPU in use and the access objects.

- Supports communication to a host CPU with an SPI interface
- The length of the SPI interface packets is variable to permit the use of variable length addresses and data accesses
- Supports writes/reads to the internal module connected to the AHB (variable, from 1 to 16 bytes)
- Conforms to Freescale Semiconductor's recommended SPI mode (CPOL=0, CPHA=0)
- Host CPU handshaking communication makes software flow control possible
- Interface can be disabled for security reasons.

3.2.2 Block Diagram

Figure 3-1 shows a block diagram of the host interface.

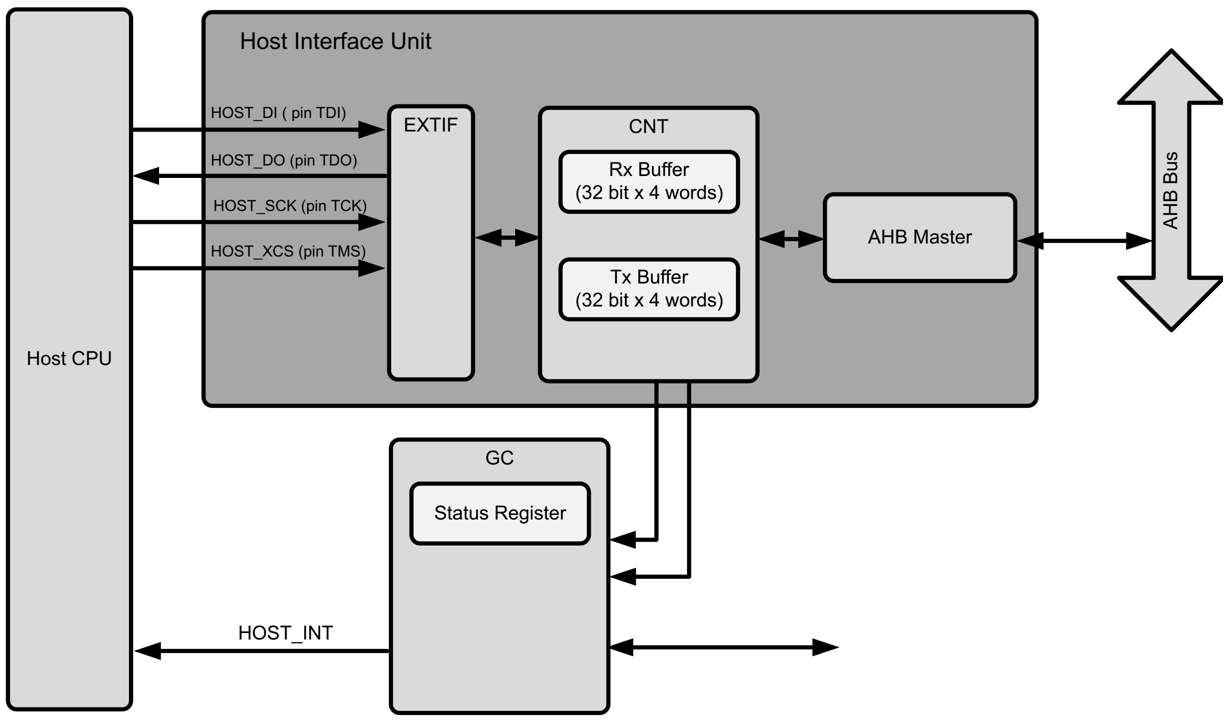


Figure 3-1: Host interface block diagram

3.2.3 Related Pins

Table 3-1: Related pins

Pin Name	Host Interface	Function Description
TDI	HOST_DI	Data input for HOSt SPI Interface
TDO	HOST_DO	Data output for HOST SPI Interface
TCK	HOST_SCK	Shift clock for HOST SPI Interface
TMS	HOST_XCS	Chip select for HOST SPI Interface

3.2.4 Functional Description

The Host interface can be completely disabled with the register IFC_CTRL.

3.2.4.1 SPI Interface

3.2.4.1.1 Write Access

Accesses from the host CPU to this module can arbitrarily use address byte lengths in a range of 1 to 4 bytes, as configured. Also, the data byte length can be arbitrarily set in a range of 1 to 16 bytes.

This module provides a function to notify the host CPU with the result of write processing. It is necessary to send a dummy write CMD after a normal write CMD. The host CPU serial clock is maintained by sending dummy write CMDs. The result of write processing is sent with this clock. The basic format of a write access is shown below.

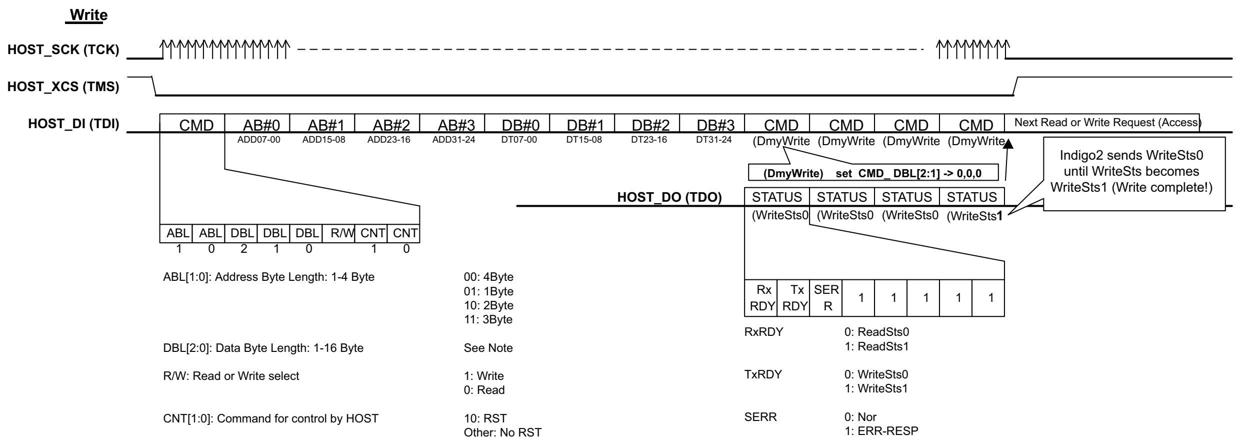


Figure 3-2: Write access

The CMD and STATUS bytes are described as follows:

CMD Byte

ABL: Address Byte Length as shown by using a 2 bit code for 1 to 4 bytes

DBL: Data Byte Length as shown by using 3 bit code for 1 to 16 bytes, see table below.

DBL2	DBL1	DBL0	Data Length
0	0	0	Dummy writes
0	0	1	1
0	1	0	2
0	1	1	4
1	0	0	8
1	0	1	12
1	1	0	16

R/W: Specifies read or write. “1” is a write.

STATUS Byte

The write status is shown by the TxRDY bit of the STATUS byte. When write processing is completed and the next transmission is possible, "1" is shown in the TxRDY bit.

The flow of a write action is shown below.

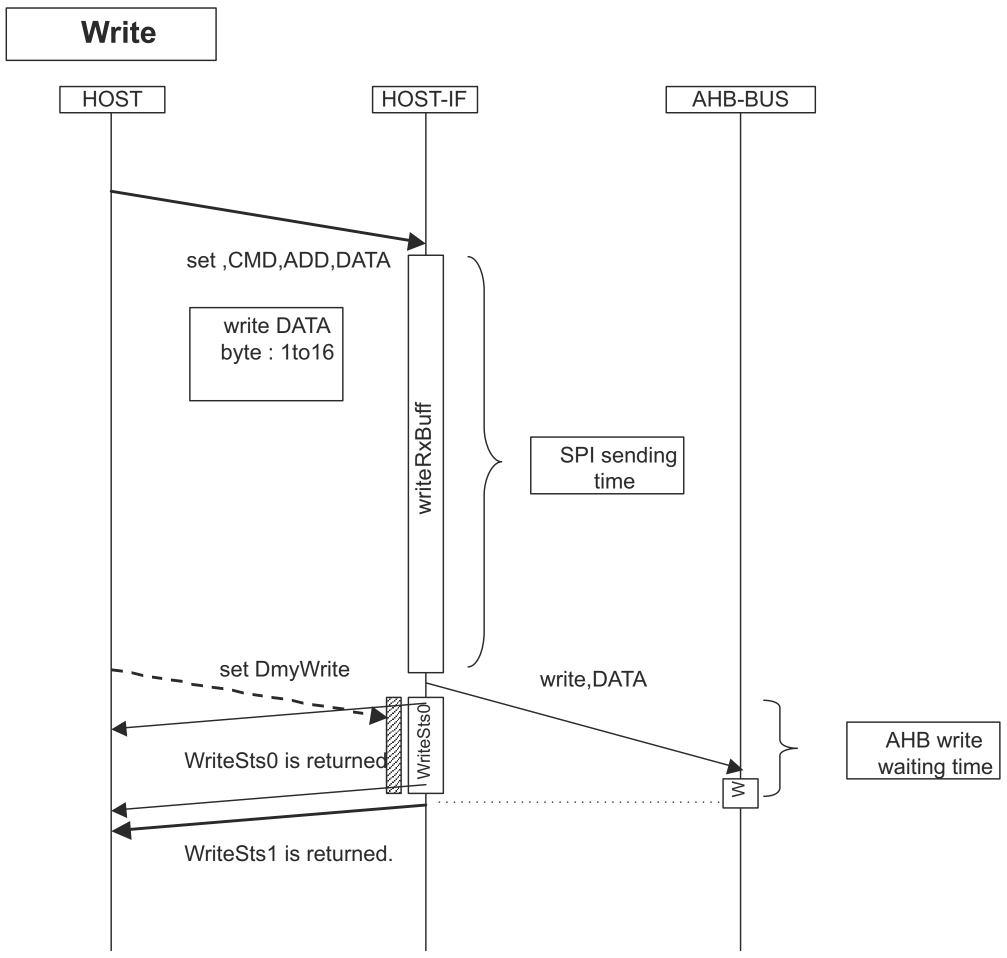


Figure 3-3: Write process flow

3.2.4.1.2 Read Access

Access from the host CPU to this module can be done using an arbitrarily set address byte length in a range of 1 to 4 bytes. In addition, the data byte length can be arbitrarily set in a range of 1 to 16 bytes.

This module adapts its read access actions accordingly by manipulating the wait time of the AHB bus. The wait and ready states for read accesses can be transmitted via a dummy write CMD, which can be used as for a write action too. The basic format of a read access is shown below.

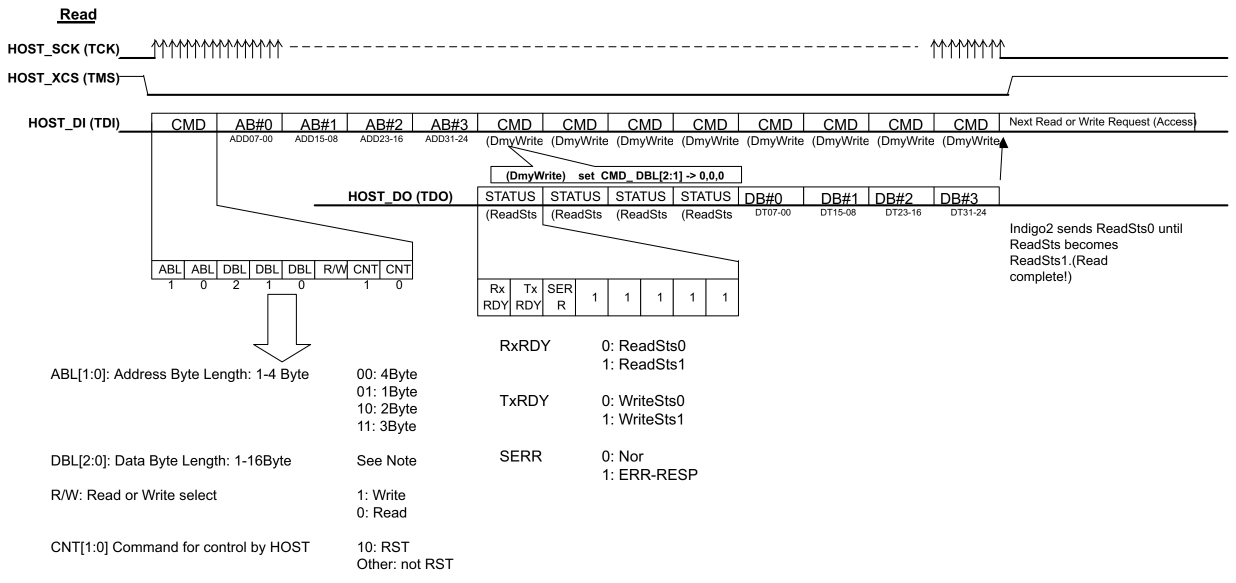


Figure 3-4: Read access

The CMD and STATUS bytes are described as follows:

CMD Byte

ABL: Address Byte Length as shown by using a 2 bit code for 1 to 4 bytes

DBL: Data Byte Length as shown by using 3 bit code for 1 to 16 bytes

DBL2	DBL1	DBL0	Data Length
0	0	0	Dummy Writes
0	0	1	1
0	1	0	2
0	1	1	4
1	0	0	8
1	0	1	12
1	1	0	16

R/W: Specifies read or write. “0” is a read.

STATUS Byte

The read status is shown by the RxRDY bit. When read processing is completed, "1" is shown in the RxRDY bit. The host CPU can retrieve the reading data at the correct time by monitoring the STATUS byte.

The flow of a read action is shown below.

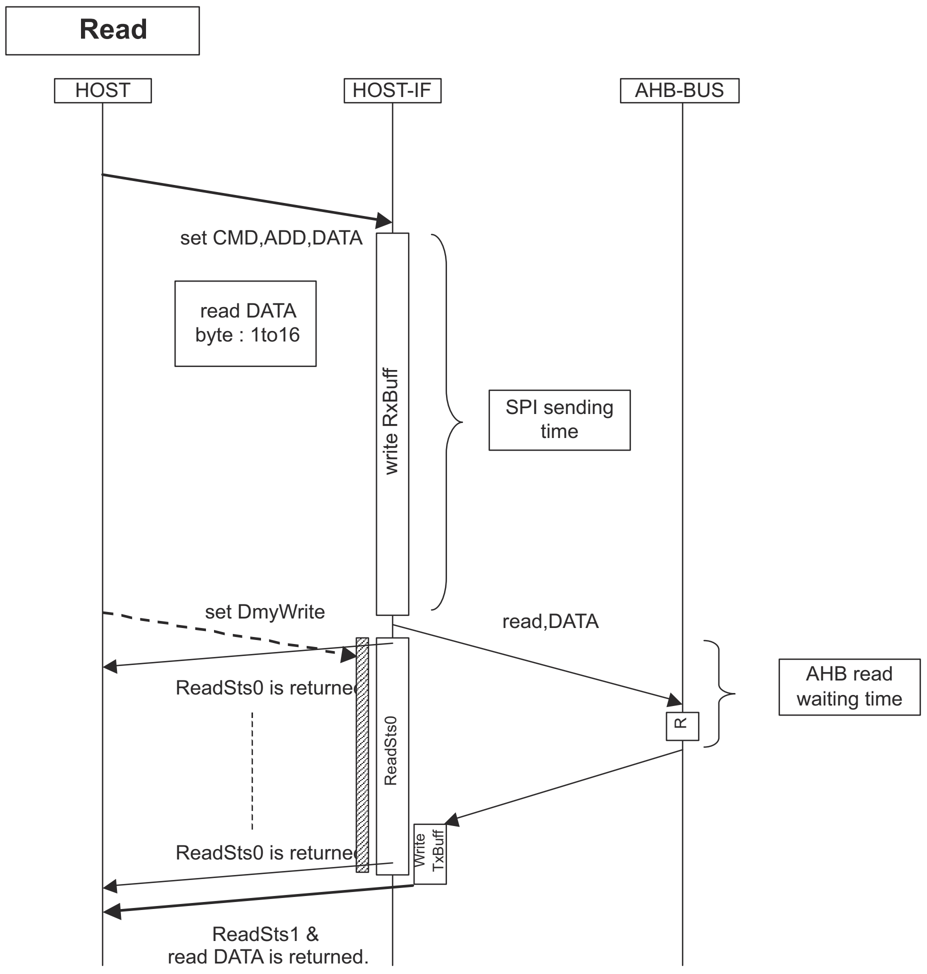


Figure 3-5: Read process flow

3.2.4.2 Interrupt

3.2.4.2.1 AHB Slave Module Access Error Response

An error response from the AHB bus is output to the chip control module, interrupt controller. In addition, an error response is written to the STATUS byte and the host CPU is immediately notified. The RxRDY bit (or TxRDY bit) is set to ‘1’ at the same time. The HOSTIF module itself does not have a register to maintain this information.

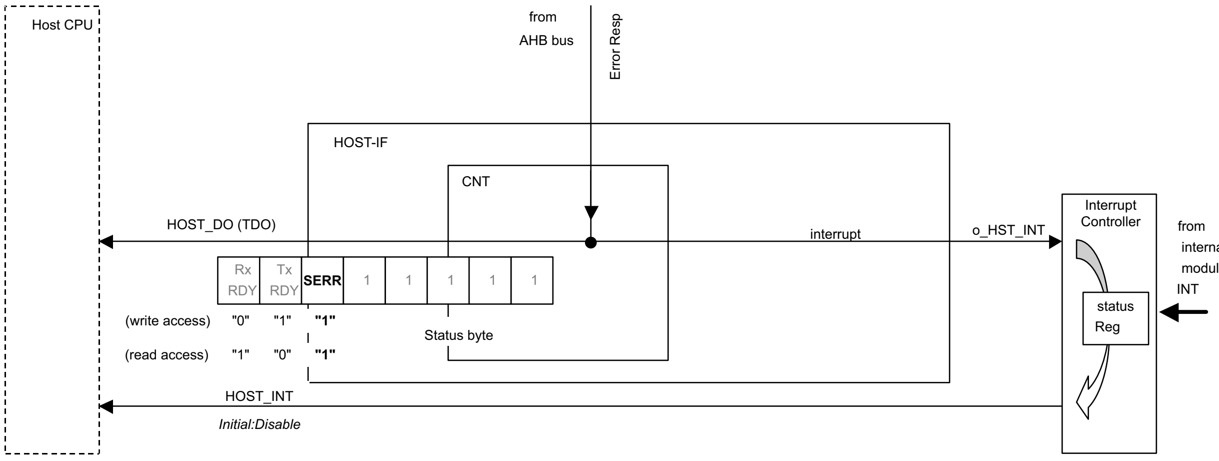


Figure 3-6: Interrupt

When an error response status has been sent to the host CPU, the transaction is completed. If the interrupt setting in the interrupt controller is enabled, an interrupt is generated.

3.2.5 Data Formats

3.2.5.1 Host Interface (Clock Timing and Phase)

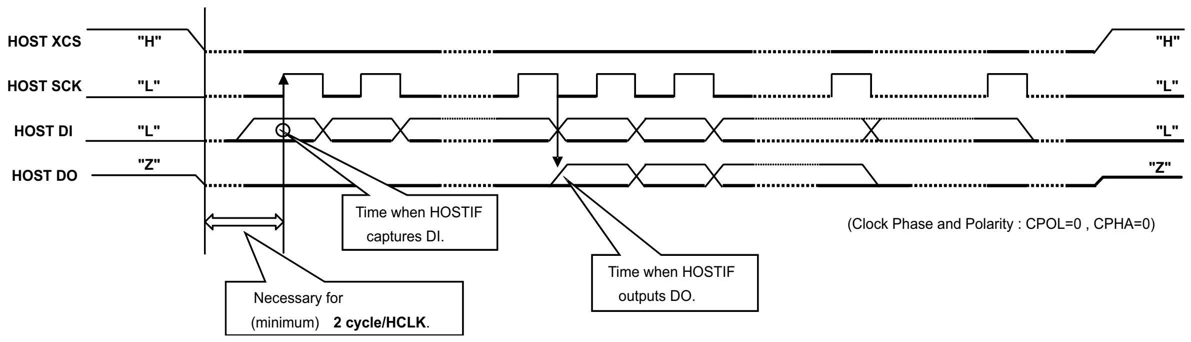

HOST XCS "H"
HOST SCK "L"
HOST DI "L"
HOST DO "Z"
"H"
"L"
"L"
"Z"
Time when HOSTIF captures DI.
Time when HOSTIF outputs DO.
(Clock Phase and Polarity : CPOL=0 , CPHA=0)
Necessary for (minimum) 2 cycle/HCLK.

Figure 3-7: Host Interface (clock timing and phase)

3.2.5.2 Reset Frame

The arrangement of the data byte inputs from the host CPU is a specific one. The byte counter of the EXTIF unit will malfunction if the HOSTIF module is initialized while the host CPU is communicating with the HOSTIF module. In this case, the arrangement of the data bytes would be mistakenly interpreted. It is therefore necessary to use a reset frame if initializing when the HOSTIF module is communicating.

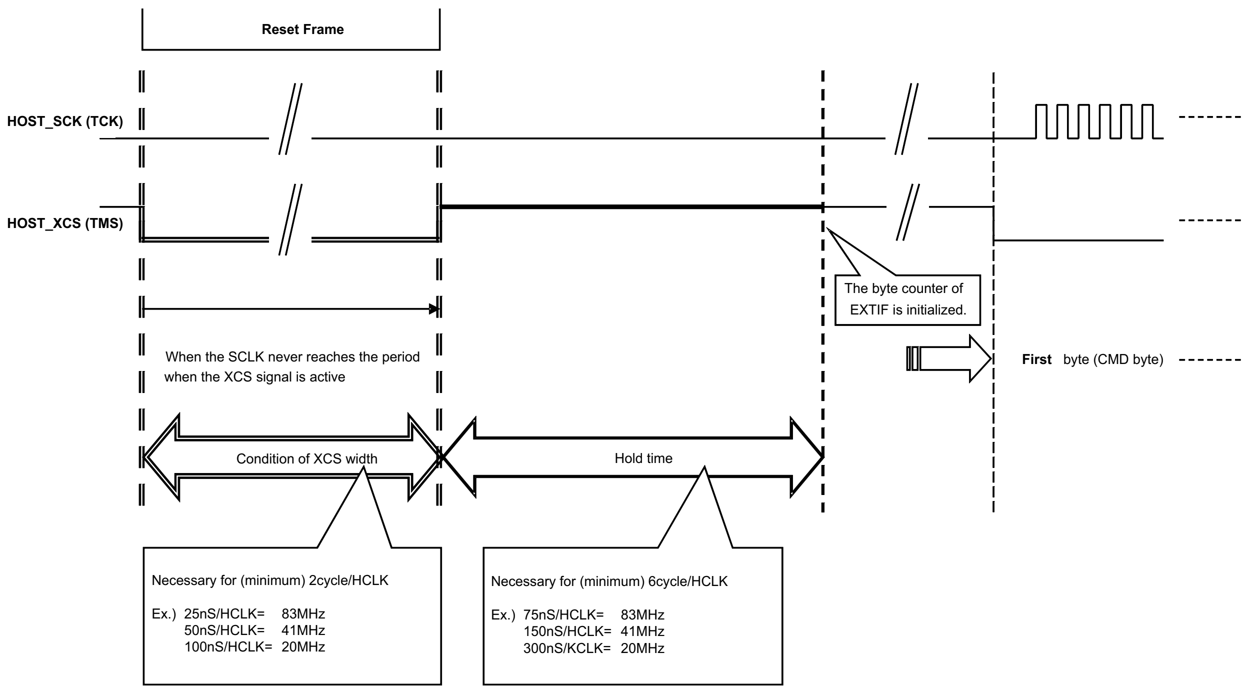


Figure 3-8: Reset Frame

3.2.5.3 Signal Input Format from the Host CPU

The phase relationships of the HOST SCK, HOST XCS, and HOST DI signals is as follows.

The HOSTIF module detects the first '0' of the HOST XCS and stores the data bytes of the specified length. Each byte can be sent using continuous and the non-continuous transmission. The HOSTIF module allows the use of the following three kinds of phase relationships.

3.2.5.3.1 Non-continuous Data Bytes with Non-continuous HOST XCS

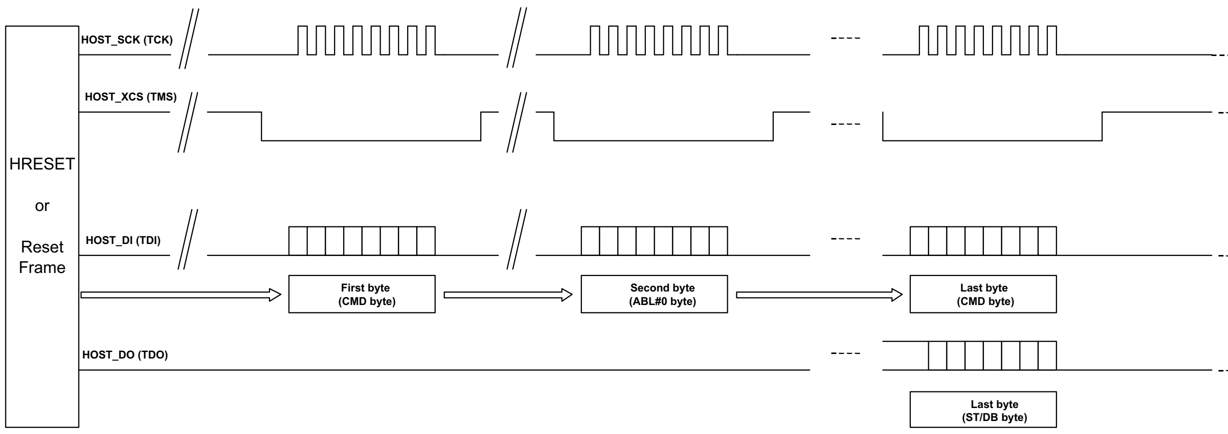


Figure 3-9: Non-continuous data bytes with non-continuous HOST XCS

3.2.5.3.2 Non-continuous Data Bytes with Continuous HOST XCS

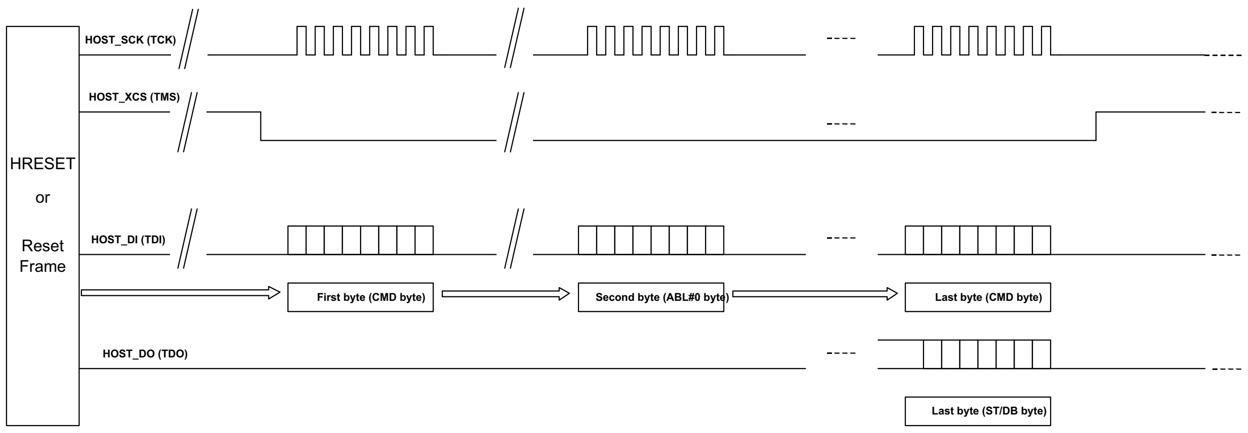


Figure 3-10: Non-continuous data bytes with continuous HOST XCS

3.2.5.3.3 Continuous Data Bytes with Continuous HOST XCS

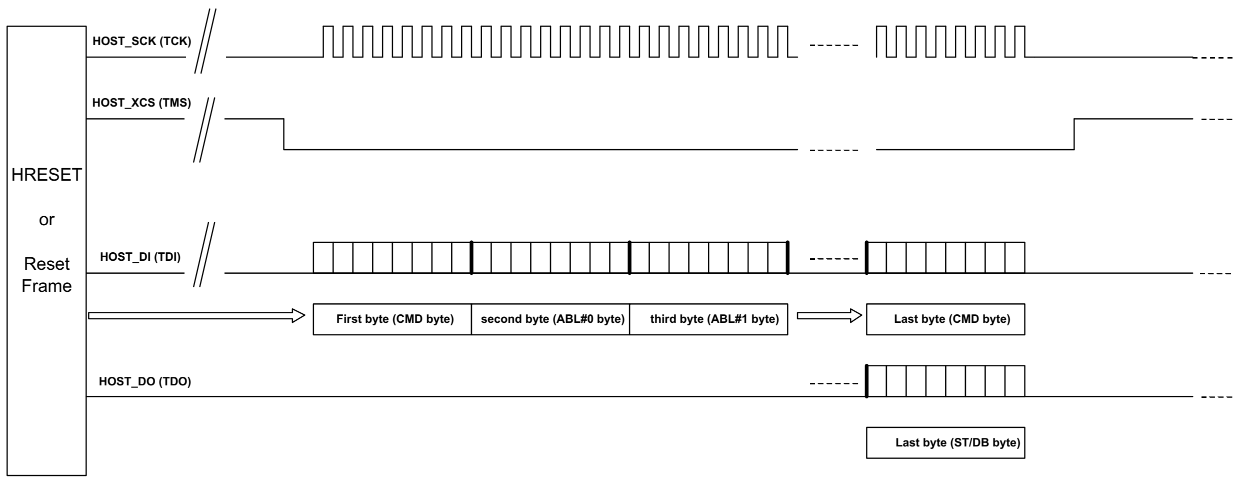


Figure 3-11: Continuous data bytes with continuous HOST XCS

3.2.6 Processing Flow

3.2.6.1 Begin Timing of Protocol Sequence

The protocol sequence sent to the Host Interface module must begin with CMD.

3.2.6.2 Receive Operation and the STATUS Byte

Normal receive operation is confirmed using five bits in the STATUS byte. If the corresponding five bits are all High, the system is in normal receive operation mode. If, for example, the STATUS byte always contains Low bits or always contains High bits, normal receive operation is not functional.

3.2.6.3 Setting the Address

The host CPU can freely select an address byte when accessing the MB88F33x ‘Indigo2(-x)’. If the address is not set, the previous address is maintained and therefore it is not necessary to repeat the address byte with every access. This implements a very effective forwarding mechanism, an example of which is shown below.

Because address information maintained in the host interface module is initialized when the communication using "Reset Frame" is done, the following forwarding cannot be done.

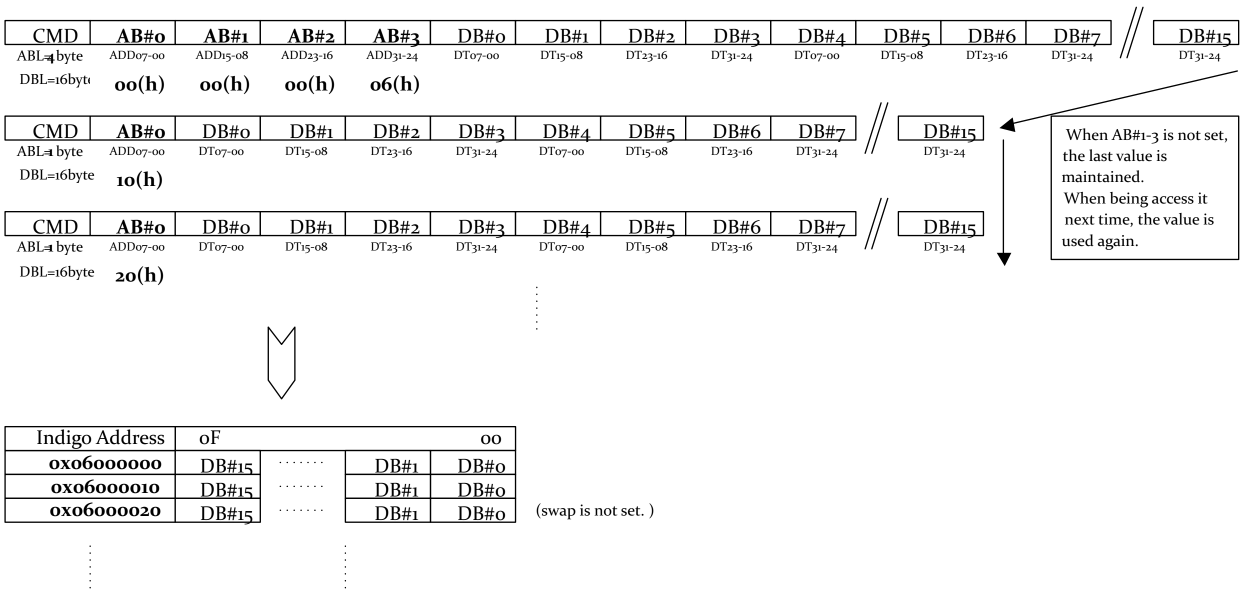


Figure 3-12: Example of setting the address (for write processing)

3.2.6.4 Handling of Irregular Operating Conditions

3.2.6.4.1 The First CMD is a Dummy Command

When the first CMD is a dummy write, the STATUS byte is immediately returned. The status is different depending on whether a READ or WRITE is returned. At this point in time, the transaction is not issued to the AHB. After the STATUS byte replies, this forwarding transaction is completed.

When the first CMD is a dummy Write (R/W bit=WRITE)

TxRDY is sent back for dummy WRITE CMD. Refer to Figure 0

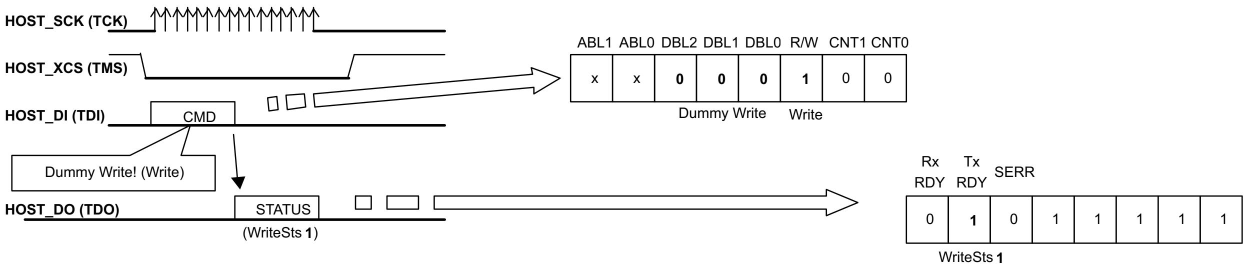


Figure 3-13: Dummy Write (R/W bit=WRITE)

When the first CMD is a dummy Read (R/W bit=Read)

RxRDY is sent back for dummy READ CMD. Refer to Figure 0

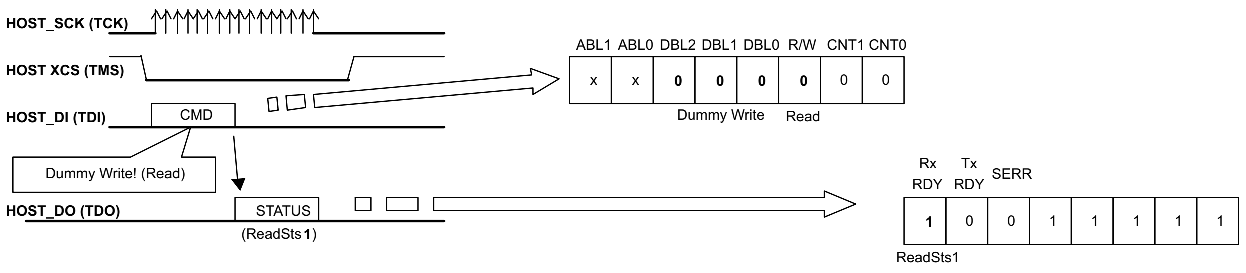


Figure 3-14: Dummy Read (R/W bit= READ)

3.2.6.4.2 The First CMD is a Reset Request

If the first CMD is a dummy RESET command, the HOSTIF module is reset at once although the response has begun. Please send the reset frame after the reset request.

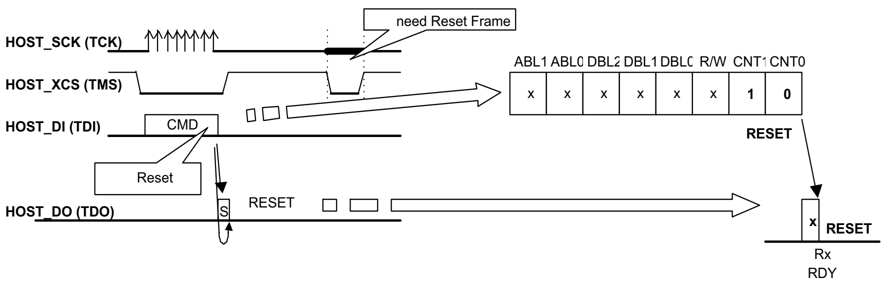


Figure 3-15: Reset Request

NOTE Please refer to the Application Note "Device Setup and Fujitsu Developer Suite" for detailed information.

3.3 Bus Matrix - Address Map

3.3.1 Overview

The bus matrix of the MB88F33x ‘Indigo2(-x)’ has an AXI layer which operates at max. 160 MHz (at 64bit) for accesses to the internal SRAM and flash memory.

The second layer is an AHB layer which operates at max. 80 MHz (at 32bit). The AHB bus clock can be configured to run either with the same frequency as the AXI layer or at 1/2 or 1/4 of the AXI clock speed.

3.3.2 The third layer is a peripheral layer for all peripherals which operate at max. 40 MHz. Address Map

Table 3-2 shows the address map for the MB88F33x ‘Indigo2(-x)’.

Table 3-2: Address map

Name	Base address	Description
GC	0x00000000	Register address for Global Control [1)]
APIX2_PHY	0x00020000	Register address for APIX2 PHY
APIX2_RX	0x00021000	Register address for APIX2 RX
RH_ASHELL	0x00022000	Register address for AShell remote handler
E2IP	0x00023000	Register address E2IP
E2IP remote handler	0x00024000	Register address E2IP remote handler
HDCP	0x00025000	Register address for HDCP Decoder control
FLASH_SPI	0x00026000	Register address for SPI Interface for Flash
PRG_CRC	0x00027000	Register address for Programmable CRC checker
DMAC	0x00028000	Register address for DMA Controller
CMDSEQ	0x0002c000	Register address for Command Sequencer
FLASH_CTL	0x0002D000	Register address for Control for internal Flash
CONFIG_FIFO	0x0002E000	Register address for Configuration Fifo
IRIS_MVL_GC	0x00030000	Register address for Iris MVL global control
IRIS_MVL_PE	0x00030800	Register address for Iris MVL pixel engine top level
IRIS_MVL_FRLD	0x00030c00	Register address for Iris MVL pixel engine Fetch RLD
IRIS_MVL_FSPT	0x00031000	Register address for Iris MVL pixel engine Fetch Sprite
IRIS_MVL_ESRC	0x00031400	Register address for Iris MVL pixel engine ExtSrc (APIX input stream)
IRIS_MVL_EDST0	0x00031800	Register address for Iris MVL pixel engine ExtDst0 (Memory stream)
IRIS_MVL_EDST1	0x00031c00	Register address for Iris MVL pixel engine ExtDst1 (Capture stream)
IRIS_MVL_CLUT0	0x00032000	Register address for Iris MVL pixel engine CLUT
IRIS_MVL_LAYBLD0	0x00032800	Register address for Iris MVL pixel engine Layerblend0 (Foreground)
IRIS_MVL_LAYBLD1	0x00032c00	Register address for Iris MVL pixel engine Layerblend1 (Sprite)
IRIS_MVL_DE	0x00033000	Register address for Iris MVL display engine top level
IRIS_MVL_FG	0x00033400	Register address for Iris MVL display engine Frame Generator
IRIS_MVL_MTX	0x00033800	Register address for Iris MVL display engine Display Matrix
IRIS_MVL_CLUT1	0x00033c00	Register address for Iris MVL display engine CLUT
IRIS_MVL_DITHER	0x00034400	Register address for Iris MVL display engine Dither
IRIS_MVL_TCON	0x00034800	Register address for Iris MVL display engine Timing Controller
IRIS_MVL_SIG0	0x00035000	Register address for Iris MVL display engine Signature unit 0
IRIS_MVL_SIG1	0x00035400	Register address for Iris MVL display engine Signature unit 1

1) The address space of the Global control unit (0x000 - 0xFFF) is mirrored 32 times into the GC address space.

Table 3-2: Address map (Continued)

Name	Base address	Description
IRIS_MVL_SIG2	0x00035800	Register address for Iris MVL display engine Signature unit 2
IRIS_MVL_SIG3	0x00035c00	Register address for Iris MVL display engine Signature unit 3
IRIS_MVL_CE	0x00036800	Register address for Iris MVL Capture Engine
SMC_0	0x00080000	Register address for Stepper Motor Controller 0
SMC_1	0x00080400	Register address for Stepper Motor Controller 1
SMC_2	0x00080800	Register address for Stepper Motor Controller 2
SMC_3	0x00080C00	Register address for Stepper Motor Controller 3
SMC_4	0x00081000	Register address for Stepper Motor Controller 4
SMC_5	0x00081400	Register address for Stepper Motor Controller 5
SMC_TRIG	0x00081C00	Register address for Stepper Motor Controller Trigger
PWM_PPG_0	0x00088000	Register address for Pulse Width Modulator 0
PWM_PPG_1	0x00088400	Register address for Pulse Width Modulator 1
PWM_PPG_2	0x00088800	Register address for Pulse Width Modulator 2
PWM_PPG_3	0x00088C00	Register address for Pulse Width Modulator 3
PWM_PPG_0-3	0x00089000	Register address for Pulse Width Modulator Group 0-3
PWM_PPG_4	0x0008A000	Register address for Pulse Width Modulator 4
PWM_PPG_5	0x0008A400	Register address for Pulse Width Modulator 5
PWM_PPG_6	0x0008A800	Register address for Pulse Width Modulator 6
PWM_PPG_7	0x0008AC00	Register address for Pulse Width Modulator 7
PWM_PPG_4-7	0x0008B000	Register address for Pulse Width Modulator Group 4-7
PWM_PPG_8	0x0008C000	Register address for Pulse Width Modulator 8
PWM_PPG_9	0x0008C400	Register address for Pulse Width Modulator 9
PWM_PPG_10	0x0008C800	Register address for Pulse Width Modulator 10
PWM_PPG_11	0x0008CC00	Register address for Pulse Width Modulator 11
PWM_PPG_8-11	0x0008D000	Register address for Pulse Width Modulator Group 8-11
PWM_PPG_12	0x0008E000	Register address for Pulse Width Modulator 12
PWM_PPG_13	0x0008E400	Register address for Pulse Width Modulator 13
PWM_PPG_14	0x0008E800	Register address for Pulse Width Modulator 14
PWM_PPG_15	0x0008EC00	Register address for Pulse Width Modulator 15
PWM_PPG_12-15	0x0008F000	Register address for Pulse Width Modulator Group 12-15
PWM_GCNR	0x00090000	Register address for Pulse Width Modulator Common Control
I2C_0	0x00094000	Register address for I2C Interface 0
I2C_1	0x00095000	Register address for I2C Interface 1
LIN	0x00096000	Register address for LIN interface
SG	0x00097000	Register address for Sound Generator
ADC	0x00098000	Register address for Analog Digital Converter
RLT_0	0x000A0000	Register address for Reload timer 0
RLT_1	0x000A0800	Register address for Reload timer 1
RLT_2	0x000A1000	Register address for Reload timer 2
RLT_3	0x000A1800	Register address for Reload timer 3
RLT_4	0x000A2000	Register address for Reload timer 4
RLT_5	0x000A2800	Register address for Reload timer 5
RLT_6	0x000A3000	Register address for Reload timer 6
RLT_7	0x000A3800	Register address for Reload timer 7
RLT_8	0x000A4000	Register address for Reload timer 8
RLT_9	0x000A4800	Register address for Reload timer 9
RLT_10	0x000A5000	Register address for Reload timer 10

Table 3-2: Address map (Continued)

Name	Base address	Description
RLT_11	0x000A5800	Register address for Reload timer 11
RLT_12	0x000A6000	Register address for Reload timer 12
RLT_13	0x000A6800	Register address for Reload timer 13
RLT_14	0x000A7000	Register address for Reload timer 14
RLT_15	0x000A7800	Register address for Reload timer 15
GPIO	0x000A8000	Register address for GPIO control
EXT_IRQ	0x000B0000	Register address for External Interrupt Controller
EXT_SPI	0x000B1000	Register address for SPI Interface for external devices
SRAM/VRAM	0x00300000	Memory address for internal Video Memory
FLASH	0x017F2000	Memory address for internal Flash Memory
EXT_FLASH	0x10000000	Memory address for external Flash Memory
EXT_DEVICE	0x20000000	Memory address for external memory mapped Devices

Figure 3-16 on next page shows the address map for the MB88F33x ‘Indigo2(-x)’

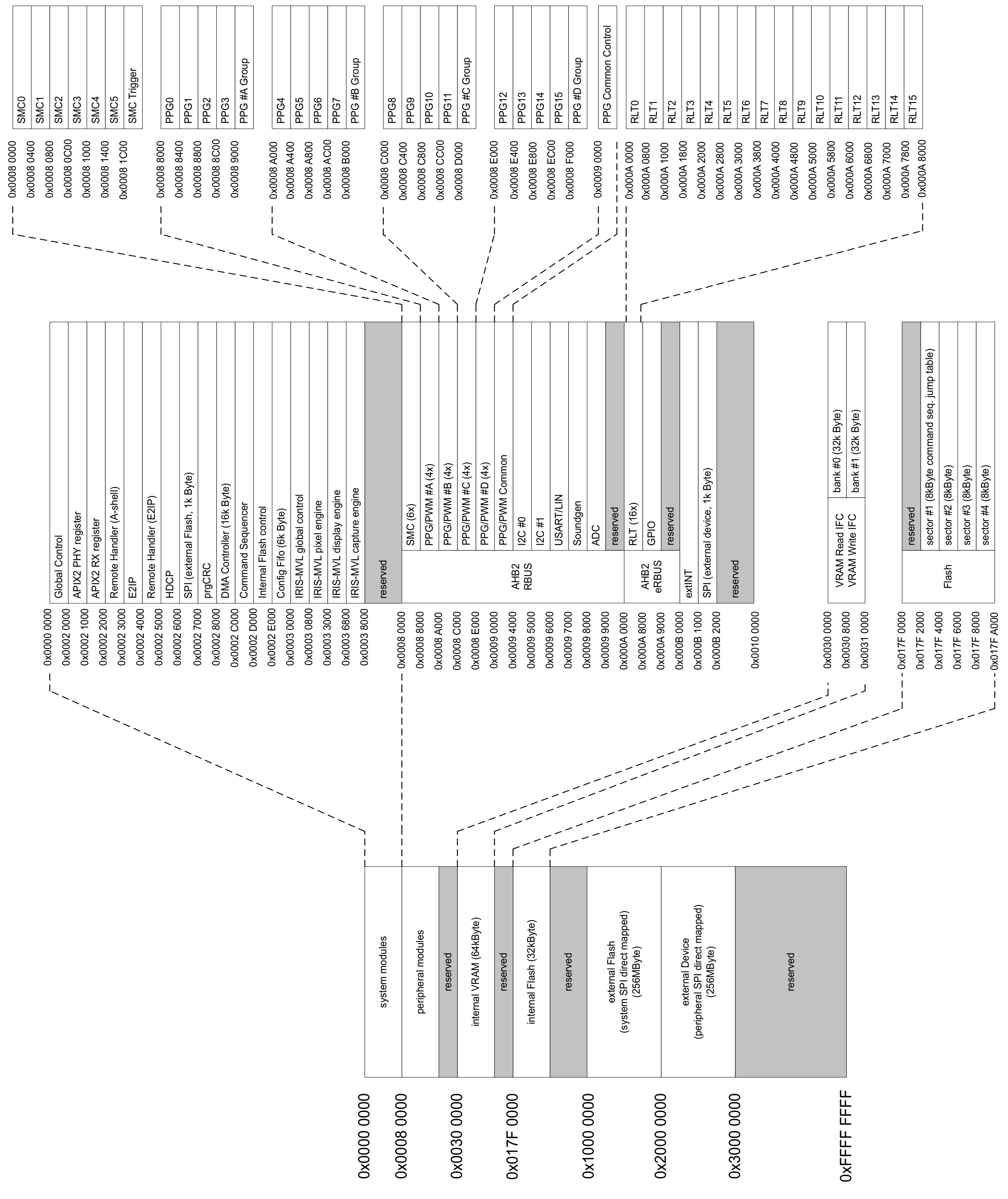


Figure 3-16: Address map

3.4 SRAM Memory

3.4.1 Overview

MB88F33x ‘Indigo2(-x)’ has 64kB embedded SRAM. This SRAM can be used for video data, command sequencer data or for miscellaneous applications.

3.5 Flash Memory

The MB88F33x ‘Indigo2(-x)’ incorporates 32 kB of user flash memory which can be used for command sequences or graphics data. This memory is internally divided into 4 sectors, each with 8kBytes.

NOTE An additional sector (number 0) is reserved for internal use and can not be used for other applications.

The jump table of the command sequencer is stored at the beginning of the first user sector (number 1). The maximum read performance of the flash memory is 32 bit at 80 MHz.

3.6 Flash AXI Interface (TCFLASH)

The Flash AXI Interface allows the internal hosts to read and write data to the internal flash.

3.6.1 Features

- Supports read and write accesses via AXI
- Flash can be of 8/16/32-bit access
- Flash writing is restricted to only 32-bit access if ECC (error control) is enabled; It can be of 8/16/32-bit access, if ECC is disabled.
- Supports unaligned read access. Unaligned write access is not supported and it leads to slave error response.
- Supports ECC-scheme for single bit error correction and double bit error detection (SEC-DED)
- Flash configuration register (TCFCFG_FCPROTKEY) settings are 32-bit key protected for safe operation.

3.6.2 Limitations

- Access to unused address range of Flash will output undefined data and generates slave error response

3.6.3 Block Diagram

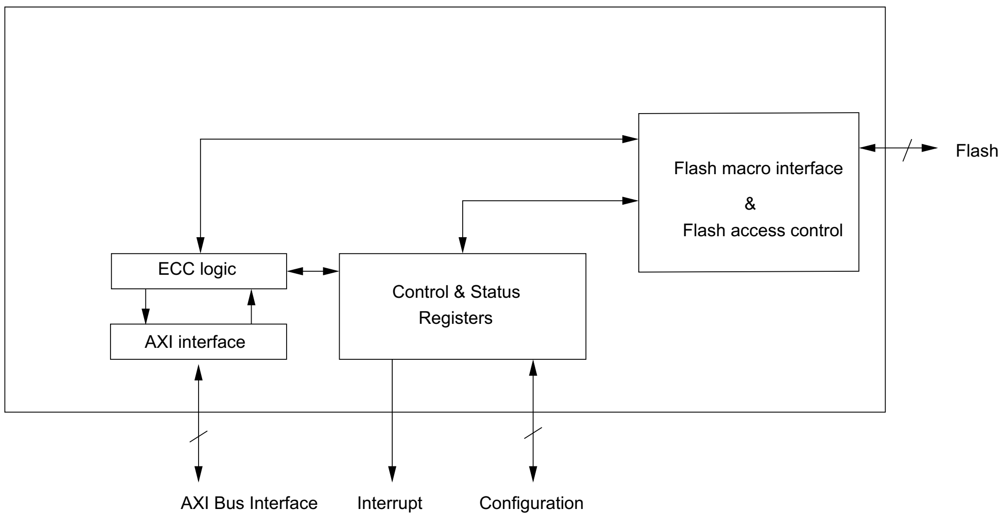


Figure 3-17: TCFLASH interface

3.6.4 Detailed Functional Description

3.6.4.1 Flash Memory

TCFLASH is the standard Flash macro, which is optimized for high read performance. One Flash macro with 4 sectors of 8KB size is available. Below is the list of features supported by Flash macro.

- Use of automatic program algorithm for program/erase
- Sector erase function (any combination of sectors) indicates write command sequence detection
- Detection of completion of writing/erasing by interrupts
- Detection of Hang-up state

3.6.4.2 Flash Memory Operation Mode

TCFLASH can be operated as follows:

User mode: Flash can be accessed by any master in the system via AXI interface. Writing/erasing of Flash is performed by programming access sequences, to trigger its auto algorithm for write, erase, etc. (See Table 3-3). ECC calculation is handled inside Flash interface.

3.6.4.3 Flash Memory Address/Sector Mapping

Address mapping of the Flash sectors is as follows:

Flash memory is mapped to AXI address space.
AXI address mappings are shown in Figure 3-18.

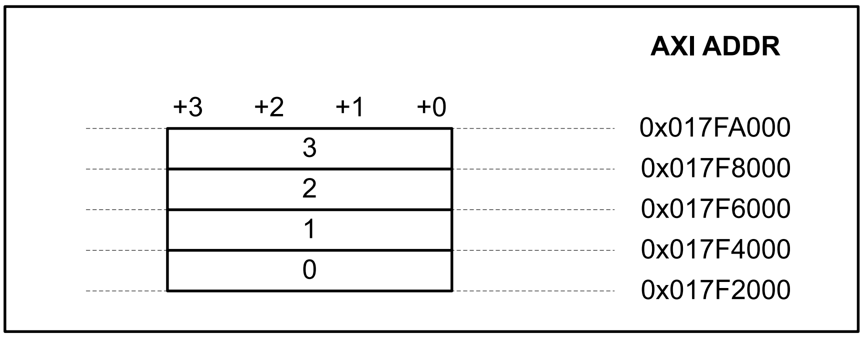


Figure 3-18: Flash Memory Map

3.6.4.4 TCFLASH Programming

TCFLASH program/erase or any master should follow the command sequence table given in Section "3.6.5 Starting the Flash Memory Automatic Algorithm". In order to handle ECC calculation and writing into Flash, Flash write is restricted to only 32-bit mode[1] and should follow a defined sequence as given below.

32-bit write

- Write the unlock code to the Flash Configuration Protection Key Register (TCFCFG_FCPROTKEY = 0xCF61F1A5)
- Set Flash write enable bit (i.e. TCFCFG_FCFGR:WE =1)

Send Flash write command sequence (as given in Table 3-3) with program address and 32-bit data to be written (Always 32-bit data must be transferred to calculate ECC from the whole data)

- With TCFCFG_FSTAT0:WR32F =0, lower 16-bit data is written to program address (PA)
- Repeat the same write command sequence[2]
- At the end of each write sequence TCFCFG_FSTAT0:WR32F[3] bit is toggled automatically
- With TCFCFG_FSTAT0:WR32F =1, upper 16-bit data + ECC is written to incremental address (PA+2)
- Clear Flash write enable bit (i.e. TCFCFG_FCFGR:WE=0)[4]

16/8-bit write

- Write the unlock code to the Flash Configuration Protection Key Register (TCFCFG_FCPROTKEY = 0xCF61F1A5)
- Set Flash write enable bit (i.e. TCFCFG_FCFGR:WE =1)
- Write the unlock code to the Flash Configuration Protection Key Register (TCFCFG_FCPROTKEY = 0xCF61F1A5)
- Disable ECC by setting a configuration register bit (TCFCFG_FECCCTRL:ECCOFF).

Send Flash write command sequence (as given in Table 3-3) with program address and 16/8-bit data to be written

- Clear Flash write enable bit (i.e. TCFCFG_FCFGR:WE=0)[4]

[1] If ECC is enabled, Flash write is restricted to 32-bit mode to allow ECC calculation; If ECC is disabled, Flash write can be of 8/16/32-bit wide. There is no true 32-bit write support by the Flash memory, but the interface accepts this data type through above defined Flash write sequence.

[2] Flash write should repeat the write sequence and the data for the second write has to be the same as for the first to ensure that the correct ECC bits are generated and complete 32-bit data and ECC are written correctly.

[3] Incase of 32-bit write, ECC check is disabled after first write sequence till the second write sequence is completed (i.e. when TCFCFG_FSTAT0:WR32F=1), as ECC data will not be consistent till the 32-bit data and ECC are written completely.

[4] For safety reasons, TCFCFG_FSTAT0:WR32F flag is cleared if Flash write is disabled (i.e. TCFCFG_FCFGR:WE = 0), to avoid temporary disabling of ECC.

NOTE See examples of Flash sequences attached to this PDF.

3.6.4.5 ECC Logic

- During write operation ECC module computes ECC on 32-bit write data in the 2nd write sequence and upper 16-bit data along with ECC is written into Flash.
- During read ECC syndrome is computed for the Flash read data in the same way as the corresponding check bit, except that syndrome calculation includes the check bit as well as the appropriate data bits. Syndrome thus interpreted for no error, single error, double and multiple errors. Multi bit error (i.e. more than two bit errors) can not be reliably detected.
- Also there is an option to disable/switch-off ECC by setting a configuration register bit (TCFCFG_FECCCTRL:ECCOFF).

3.6.4.6 Interrupts

Flash memory interface generates interrupts in following conditions

- Ready interrupt flag (i.e. TCFCFG_FSTAT0:RDYINT) is set when Flash memory RDY output goes high (indicating that Flash program/erase is completed and Flash memory is ready to accept a new command). Interrupt is generated when RDYINT interrupt flag is set and if corresponding Interrupt enable bit (i.e. TCFCFG_FICTRLn:RDYIE) is also set.
- Hangup interrupt flag (i.e. TCFCFG_FSTAT0:HANGINT) is set, when Flash memory HANG output goes high (indicating that Flash memory has detected Hangup-1 condition). Interrupt is generated when HANGINT interrupt flag is set and if corresponding interrupt enable bit (TCFCFG_FICTRLn:HANGIE) is also set.
- SEC interrupt flag (i.e. TCFCFG_FSECIR:SECINT) is set, when ECC Checker logic has detected a Single-bit error in the Flash Read data. Interrupt is generated if SECINT interrupt flag is set and corresponding interrupt enable bit (TCFCFG_FSECIR:SECIE) is also set.

3.6.4.7 Bus Error Response

Flash memory interface generates bus error response on following conditions

- ECC double or multi bit error condition.
- 8/16-bit write access, with ECC enabled
- 8/16-bit write access during the second write sequence for 32-bit write
- Unaligned write access to Flash memory
- Access to unused address space of Flash memory and configuration registers
- Write access to read only registers
- Writing TCFCFG_FECCCTRL register more than once
- Any deviation in Flash configuration unlock sequence

3.6.5 Starting the Flash Memory Automatic Algorithm

Write and erase to Flash memory is performed by launching the Flash memory's own Automatic Algorithms. The following commands are available: Read/Reset, Write, and Sector Erase.

3.6.5.1 Command Sequence Table

Automatic Algorithms for Flash memory write/erase are launched by writing one to six bytes or half-words or words to the Flash memory in succession according to Table 3-3 .

The data of the commands must be written to the low-order byte (except the program data of the write command). The data written to the high-order bytes (in case of half-word or word access) is ignored.

The data width used for the 4th bus write cycle of the write command (program address and program data) determines the write mode of the Flash (byte or half-word or word).

Table 3-3: Command Sequence

Command sequence	Bus write access	1st bus write cycle		2nd bus write cycle		3rd bus write cycle		4th bus write cycle		5th bus write cycle		6th bus write cycle	
		Address	Data	Address	Data	Address	Data	Address	Data	Address	Data	Address	Data
Read/Reset	1	cmd_addr0	F0	-	-	-	-	-	-	-	-	-	-
Write/Pro-gram	4	cmd_addr0	AA	cmd_addr1	55	cmd_addr0	A0	PA	PD	-	-	-	-
Sector Erase	6	cmd_addr0	AA	cmd_addr1	55	cmd_addr0	80	cmd_addr0	AA	cmd_addr1	55	SA	30
PA: Program Address; PD: Program Data SA: Sector Address													

Table 3-4: Command address values

Command Address	Sector Access
cmd_addr0	0x017F_0AA8
cmd_addr1	0x017F_0554

Command sequence	Bus write access	1st bus write cycle		2nd bus write cycle		3rd bus write cycle		4th bus write cycle	
		Address	Data	Address	Data	Address	Data	Address	Data
Write/ Program	4	0x017F_0AA8	AA	0x017F_0554	55	0x017F_0AA8	A0	PA	PD
PA: Program Address; PD: Program Data SA: Sector Address									

NOTE

- The addresses in the tables are the values in the MB88F33x 'Indigo2(-x)' memory map. All addresses and data are represented using hexadecimal notation.
- Flash data bits [31:8] are ignored for command writing (except for the program data PD).

- In case of 4th write cycle of write/program sequence, PA should be actual write/program address location and PD is data*[1] to be written.
- Program Address (PA) should be aligned*[2] at 4 byte grid (for 32-bit write). If TCFCFG_FSTAT0:WR32F =0, lower half-word (PD [15:0]) is written to Flash at address PA+0. If TCFCFG_FSTAT0:WR32F =1, upper half-word and ECC (PD [31:0] & ECC [6:0]) are written to incremental address PA+2
- SA in the table should point the actual sector address on which sector erase or erase suspend or erase resume command is meant to operate. Sector selection is done same as for Program address (PA).
- Writing an illegal address or data, or writing them in the incorrect order, will reset the Flash memory to the read mode.
- It is possible to read data from the Flash between the write cycles of the commands. The command execution starts after the last write cycle.

*1: PD should be 32-bit data if ECC is enabled. PD can be of 8/16/32-bit wide if ECC is disabled

*2: PA should be aligned to 2-byte grid for half-word access and 4-byte grid for word access

NOTE If half word and byte write is used, ECC bits are not programmed. Thus its recommended to use this only when ECC is disabled, to avoid later errors at read back.

3.6.6 Confirming the Automatic Algorithm Execution State

The Flash memory performs the write/erase sequence via automatic algorithms. It thus has hardware for informing the outside world when it has finished internal operations.

3.6.6.1 Hardware Ready Flag

For checking whether automatic writing or erasing is being executed, the TCFCFG_FSTAT0: RDY bit can be read.

Flash memory will transit to command state after completion of automatic algorithm. This must be confirmed by RDY flags before performing the next operation such as data read.

3.6.7 Detailed Explanation of Writing to and Erasing Flash Memory

This section describes each operation procedure of the Flash memory: Read/Reset, Write, and Sector Erase.

3.6.7.1 Detailed Explanation of Flash Memory Write/erase

By issuing a command sequence (see Table 3-3) the Flash memory executes the automatic algorithm to perform Read/Reset, Write, or Sector Erase.

Termination of the automatic algorithm can be determined by polling the RDY flag (TCFCFG_FSTAT0:RDY) or by RDY interrupt. At normal termination, the Flash memory returns to the normal command state. Abnormal termination of automatic algorithm is determined by polling HANG flag (TCFCFG_FSTAT0:HANG) or by HANG interrupt.

Each operation of the Flash memory is described in the following chapters:

- Setting the read/reset state
- Writing data by submitting the Write command sequence
- Erasing (sector)

3.6.8 Setting Read/Reset State

This section describes the procedure for issuing the Read/Reset command to set the Flash memory to the Read/Reset state.

3.6.8.1 Read/Reset Command Sequence

Command sequence	Bus write access	1st bus write cycle	
		Address	Data
Read/Reset	1	cmd_addr0	F0

3.6.8.2 Setting the Flash Memory to the Read/reset State

The read/reset state is the initial state of the Flash memory. When the power is turned on and when a command terminates normally, the Flash memory is set to the read/reset state. In the read/reset state, any command can be input.

The Read/Reset command is not required to read data by a regular read. The Read/Reset command is mainly used to initialize the automatic algorithm when a command does not terminate normally.

The Read/Reset command has no effect in normal operation of the write or erase automatic algorithm. It cannot be used to interrupt an ongoing write or erase command execution. Also resetting the Flash from the sector erase suspended state is not possible.

3.6.9 Writing Data by Submitting the Write Command Sequence

This section describes how to write data to the Flash memory by sending the write command sequence.

3.6.9.1 Write Command Sequence

Command sequence	Bus write access	1st bus write cycle		2nd bus write cycle		3rd bus write cycle		4th bus write cycle		5th bus write cycle		6th bus write cycle	
		Address	Data	Address	Data	Address	Data	Address	Data	Address	Data	Address	Data
Write/Pro-gram	4	cmd_addr0	AA	cmd_addr1	55	cmd_addr0	A0	PA	PD	-	-	-	-

3.6.9.2 Starting the Write Automatic Algorithm

The data write automatic algorithm of the Flash memory can be started by sending the Write command sequence as given above (Refer to Table 3-3 for details). When data write to the target address is completed in the fourth command cycle, the automatic algorithm for writing is started.

- Specifying addresses

Any programming data is written to its target address which will be aligned to the respective target size.

In case of byte and half word data writes, the address will be given directly for the programmed Flash target location. In case of 32 bit data the access must be repeated with the same aligned address. It is internally split for the upper and lower half word of data.

NOTE

- If ECC is on, only 32 bit write should be used. The Flash interface takes care about generating ECC bits and the correct addresses internally.
- If ECC is off, 32, 16 and 8 bit can be used. In case of 32 bit, also here the access needs to be repeated.
- Polling of hardware sequence flag in case of 32 bit should be done as follows: First polling is done on the original write address and second polling has to be done from the write address + 2 because of the upper half word is written in the second step.

- Notes on writing data

Writing cannot return a data bit in the Flash from 0 to 1. When trying to program a bit to 1 which is already set to 0, Flash memory goes to "Hang up 1" state. It is possible to identify whether or not macro has entered hang up 1 state by looking at TCFCFG_FSTAT0:HANG status bit. A bit programmed to 0 can only be set to 1 by an erase operation.

All commands are ignored during execution of the automatic write algorithm. Asserting reset or software triggered Flash reset (i.e. TCFCFG_FCFGR:SWFRST) will trigger Flash hardware reset, which in turn cancels an ongoing write operation and puts the Flash memory to normal command state. In such cases the data of the written addresses will be unpredictable.

3.6.9.2.1 Example for Writing to the Flash Memory

Figure 3-19 show an example of the procedure for writing to the Flash memory.

TCFCFG_FSTAT0:RDY and TCFCFG_FSTAT0:HANG flags can be used to confirm Flash write completion as given in below figure.

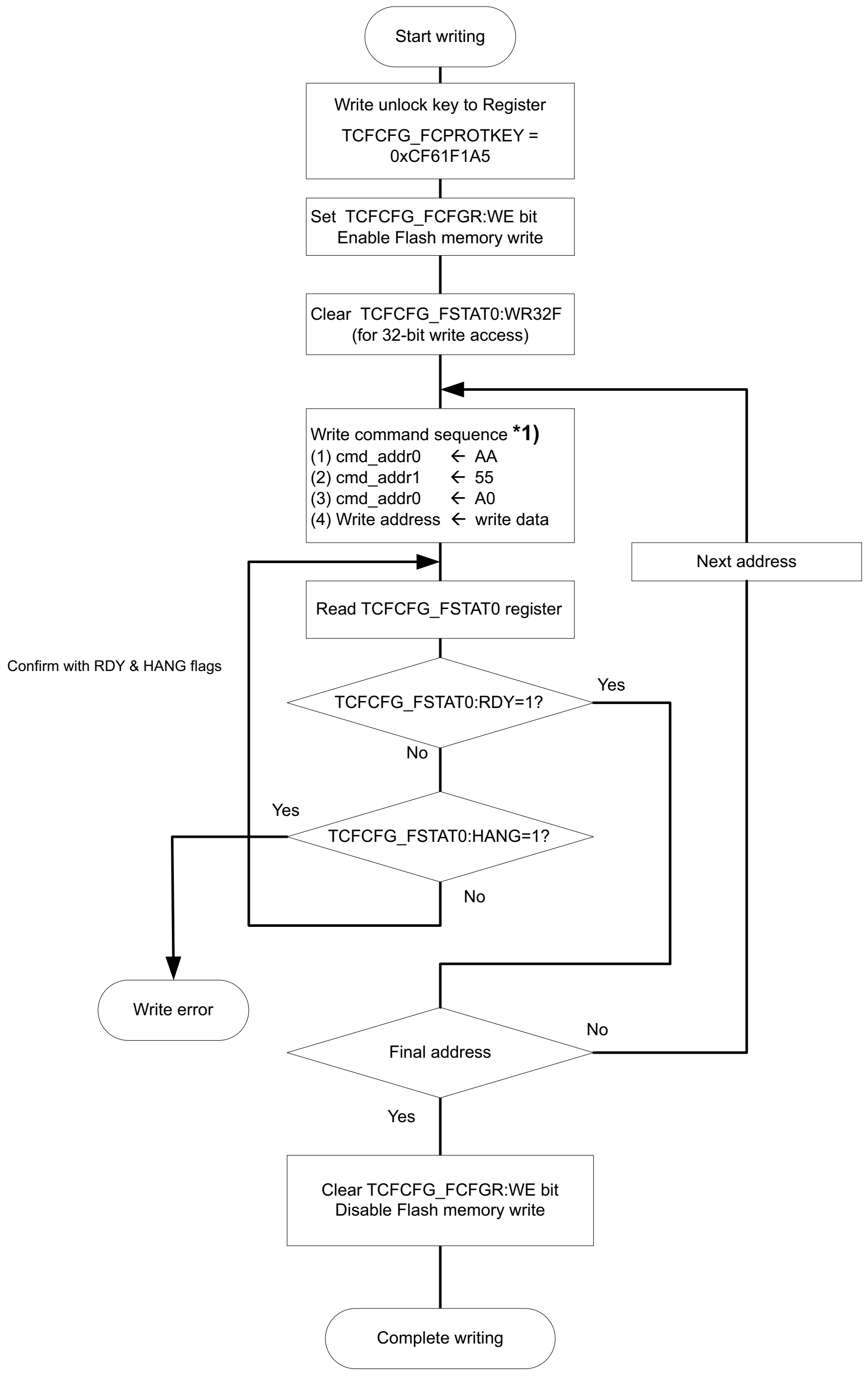

Start writing
Write unlock key to Register
TCFCFG_FCPROTKEY = 0xCF61F1A5
Set TCFCFG_FCFGR:WE bit
Enable Flash memory write
Clear TCFCFG_FSTAT0:WR32F
(for 32-bit write access)
Write command sequence *1)
(1) cmd_addr0 ← AA
(2) cmd_addr1 ← 55
(3) cmd_addr0 ← A0
(4) Write address ← write data
Next address
Read TCFCFG_FSTAT0 register
Confirm with RDY & HANG flags
TCFCFG_FSTAT0:RDY=1?
Yes
No
TCFCFG_FSTAT0:HANG=1?
Yes
No
Write error
Final address
No
Yes
Clear TCFCFG_FCFGR:WE bit
Disable Flash memory write
Complete writing

Figure 3-19: Example of the Flash memory write procedure

NOTE

***1) In case of 32-bit write with ECC**, **each write sequence** in above flow chart **should be repeated** as mentioned in sub section Section 3.6.4.4.(TCFLASH Programming).

Refer to Table 3-4 for values of cmd_addr0 and cmd_addr1.

3.6.10 Erasing User Data (Sector Erase)

This section describes the procedure for issuing the Sector Erase command to erase optional data in the Flash memory. Individual sectors can be erased. Multiple sectors can also be specified at one time.

3.6.10.1 Sector Erase Command Sequence

Command sequence	Bus write access	1st bus write cycle		2nd bus write cycle		3rd bus write cycle		4th bus write cycle		5th bus write cycle		6th bus write cycle	
		Address	Data	Address	Data	Address	Data	Address	Data	Address	Data	Address	Data
Sector Erase	6	cmd_addr 0	AA	cmd_addr 1	55	cmd_addr 0	80	cmd_addr 0	AA	cmd_addr 1	55	SA	30

3.6.10.2 Starting the Sector Erase Automatic Algorithm

Sectors in the Flash memory can be erased by sending the Sector Erase command sequence as given above (Refer to Table 3-3 for details) continuously to the target sector in the Flash memory.

- Specifying sectors

A Sector erase is initiated by submitting the sector erase command (six write operations) to the target sector. After submitting the last command (30h) to the target sector, the sector erase wait time is applied for a minimum of 40us. To erase multiple sectors, write the erase code 30h (sixth cycle of the command sequence) to the next target sector within this wait time. The first 5 cycles of the sector erase command do not have to be written in this case.

During execution of the automatic erase algorithm, the Flash memory automatically writes 0 before all of the cells of target sector are erased. After the completion of sector erase all the memory cells of the target sector will store ’1’.

- Notes on specifying multiple sectors

Erase is started when the sector erase wait period of 40us terminates after the last sector erase code (30h) has been written. Each writing of a sector erase code restarts the sector erase wait period. The sector erase timer flags DQ[3,11] must be checked after submitting each erase code to make sure it has been accepted.

3.6.10.3 Example for Erasing Sectors in the Flash Memory

The RDY flags (see Section 3.6.6.1 on page 25) can be used to determine the state of the automatic algorithm in the Flash memory. Figure 1-5, “Example of the Flash memory single sector erase procedure,” on page 1 - 25 shows the flow for Flash single sector erase.

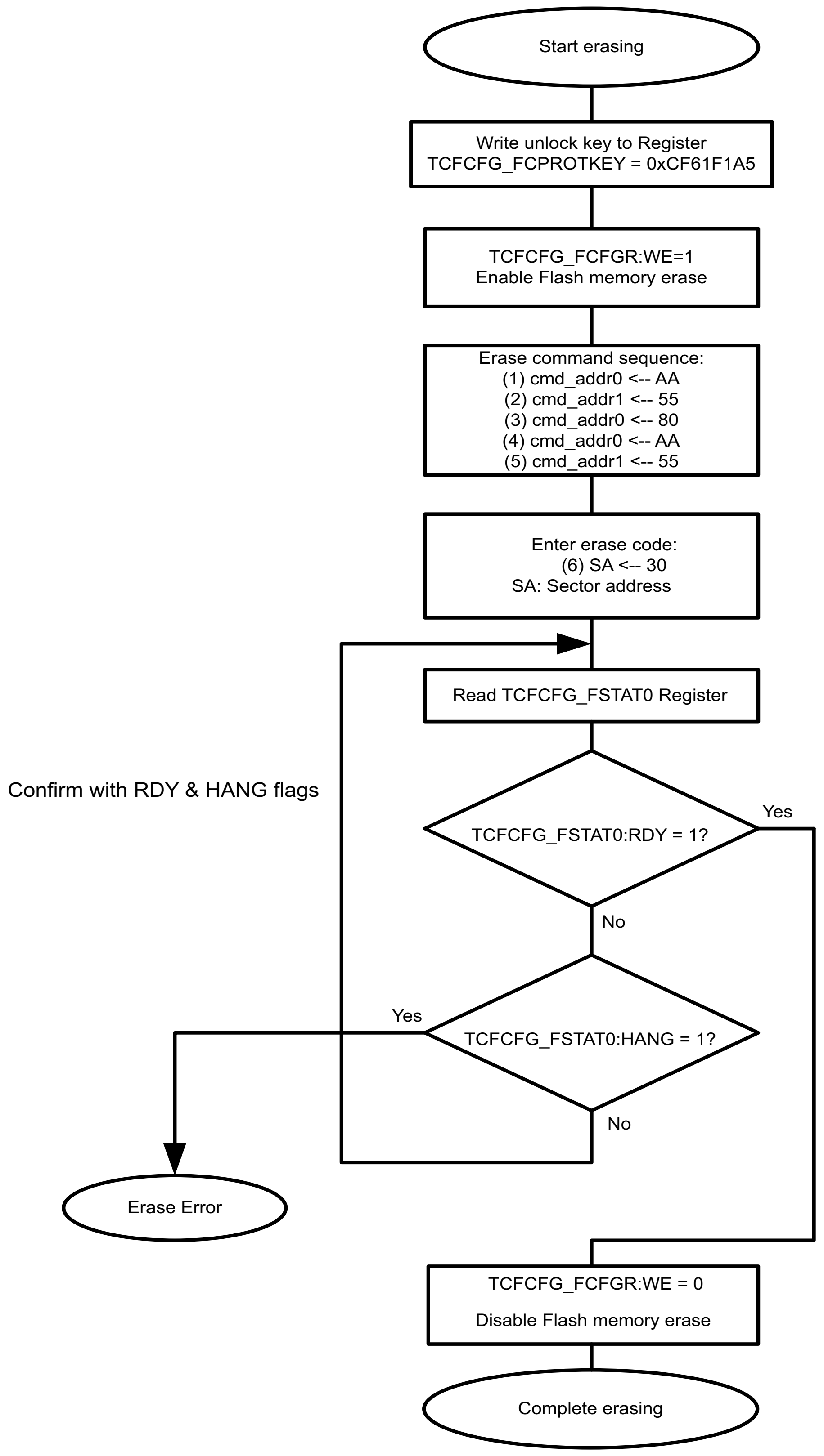


Figure 3-20: Example of Erasing Sectors in the Flash Memory

3.6.11 Notes on Using Flash Memory

This section contains notes on using Flash Memory.

- Input of an external reset

An external reset asserts the Flash hardware reset. Such a reset assertion during Flash writing causes the data being written to be undefined.

Resetting the device once execution of sector erase has begun will corrupt the data in the sector. In that case, restart the erase on this sector and allow it to complete.

- Register Configuration

Since register configuration and Flash memory access are done through different bus interfaces, it is recommended that Flash Configuration Register (TCFCFG_FCFGR) settings are not changed while Flash memory access is in progress.

Also since Flash reset is asserted for a period of minimum 500 ns, during software triggered Flash reset, software should wait till the completion of Flash reset before starting the memory access. Software can read TCFCFG_FSTAT0:RDY bit to know whether Flash is ready for read or next command (RDY output from Flash macro returns to '1' latest after 80 us).

- Program access to Flash memory

While the Flash memory is erased or data is written to the Flash memory, reading data is not possible. Hence, when it is required to read data or code from the Flash memory while the memory is erased or written, the required data or code must be copied to RAM before starting the erase/write operation. This eliminates the need for the program to read the Flash memory while the Flash memory is erased or written. Thus it should be made sure that no code is executed from a Flash memory that is erased/ written.

- Flash Programming and Hardware Sequence Flags
  - It is recommended that software sets TCFCFG_FCFGR:WE bit only before Flash programming and clears this bit after completion of auto-algorithm. TCFCFG_FCFGR:WE bit must remain asserted during polling of HW sequence flags.
  - Since HW sequence flags are not ECC protected, ECC check is disabled when Flash auto algorithm is in progress. Also incase of 32-bit write, ECC check is disabled till both write sequences (lower 16-bit and upper 16-bit+ECC) are completed as ECC data will not be consistent after first write.
  - TCFCFG_FSTAT0:RDY flag or RDYINT interrupt should be used to know auto-algorithm status for a read protected sector.
  - It should be taken care that concurrent accesses from different masters be avoided during Flash programming. Also it is recommended that software to make sure all the pending read accesses (i.e. already issued read accesses) are completed before starting Flash programming/erase operation.
- ECC Error Handling:
  - ECC checking is done on 32-bit data read from the Flash

3.6.12 Internal Flash Control Register Overview

Table 3-5: Registers Overview

Base Address(es)	**Instance no 0: BASEADDR="0002D000"**	
Absolute Address	**Register Name**	**Register Description**
BASEADDR + 0x0000	TCFCFG_FCPROTKEY	Flash Configuration Protection Key Register
BASEADDR + 0x0004	Reserved	Do not modify
BASEADDR + 0x0008	TCFCFG_FCFGR 🔒	Flash Configuration Register
BASEADDR + 0x000C	Reserved	Do not modify
BASEADDR + 0x0010	TCFCFG_FECCCTRL 🔒	Flash ECC Control Register
BASEADDR + 0x0014	Reserved	Do not modify
BASEADDR + 0x0018	Reserved 🔒	Do not modify
BASEADDR + 0x001C	Reserved 🔒	Do not modify
BASEADDR + 0x0020	TCFCFG_FICTRL0	Flash Interrupt Control Register 0
BASEADDR + 0x0024	Reserved	Do not modify
BASEADDR + 0x0028	Reserved	Do not modify
BASEADDR + 0x002C	Reserved	Do not modify
BASEADDR + 0x0030	Reserved	Do not modify
BASEADDR + 0x0034	Reserved	Do not modify
BASEADDR + 0x0038	TCFCFG_FSTAT0	Flash Status Register 0
BASEADDR + 0x003C	Reserved	Do not modify
BASEADDR + 0x0040	Reserved	Do not modify
BASEADDR + 0x0044	Reserved	Do not modify
BASEADDR + 0x0048	Reserved	Do not modify
BASEADDR + 0x004C	Reserved	Do not modify
BASEADDR + 0x0050	TCFCFG_FSECIR	Flash SEC Interrupt Register
BASEADDR + 0x0054	TCFCFG_FECCEAR	Flash ECC Error Address Register
BASEADDR + 0x0058	Reserved	Do not modify
BASEADDR + 0x005C	Reserved	Do not modify
BASEADDR + 0x0060	Reserved	Do not modify
BASEADDR + 0x0064	Reserved	Do not modify
BASEADDR + 0x0068	Reserved	Do not modify
BASEADDR + 0x006C	Reserved	Do not modify
BASEADDR + 0x0070	Reserved	Do not modify
BASEADDR + 0x0074	Reserved	Do not modify
BASEADDR + 0x0078	Reserved	Do not modify
BASEADDR + 0x007C	Reserved	Do not modify

3.7 HS-SPI Interface for External Flash Memory

An external serial flash memory can be connected to the device in order to increase the total amount of flash memory available in the system (internal + external). A high speed SPI interface (HS-SPI) is available for this purpose. The module provides various operating modes for interfacing to serial peripheral flash memory devices that use the de-facto standard SPI protocol.

The interface also supports dual and quad I/O SPI operation modes. Beside connecting to an external flash, the interface can be used to connect other SPI peripherals.

NOTE For more information about the SPI Interface for the external Flash Memory please refer to chapter “6.7 High-speed SPI Interface (HS_SPI)”.

An alternative SPI Interface for external Flash Memory is not possible.

NOTE This functionality is not implemented in MB88F335 ‘Indigo2-S’.

3.7.1 Software Interface

The base address for the External Flash Memory register space is 0x00026000.

The base address for the memory mapped external Flash memory starts at 0x10000000

3.7.1.1 Address Map of the External Flash Memory

The HS_SPI module allocated 256MB of the address space of the MB88F33x ‘Indigo2(-x)’ for memory mapping of the external serial memory devices, using Command Sequencer Mode. An additional 1KB of address space of the MB88F33x ‘Indigo2(-x)’ is reserved for mapping the internal Configuration and Status registers (i.e. CSRs) of HS_SPI.

The address area allocated to HS_SPI is discussed in this section.

Arrangement of HS_SPI Address Space in Memory

Figure 3-21 shows the allocation of HS_SPI address space in the address space of MB88F33x ‘Indigo2(-x)’.

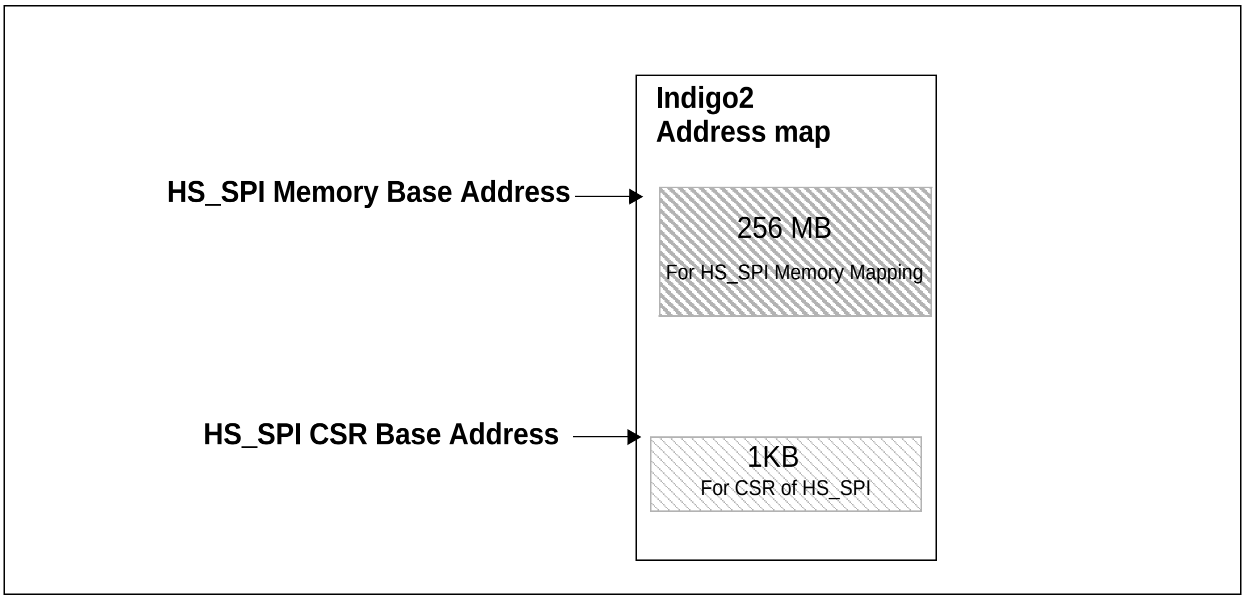


Figure 3-21: Address Map of HS_SPI

- Allocation for Memory Mapped Devices

The 256MB of memory space, starting from “HS_SPI Memory Base Address” is reserved for memory mapping of the external serial devices onto the address space of the MB88F33x ‘Indigo2(-x)’.

This space is used in Command Sequencer mode.

- Allocation for CSRs

The 1KB of memory space, starting from “HS_SPI CSR Base Address” is reserved for memory mapping of the Configuration and Status registers of HS_SPI on to the address space of the MB88F33x ‘Indigo2(-x)’.

3.8 Command Sequencer

3.8.1 Overview

The Command Sequencer unit is responsible for parsing command lists, the distribution of data to the addressed blocks and for the synchronization on certain events.

Command list addresses can be provided by the host CPU or can be fetched directly from local memory.

NOTE For users of MB88F332 ‘Indigo’: The Command Sequencer implemented here is a new version. It is not compatible to the previous version!

3.8.2 Block Diagram

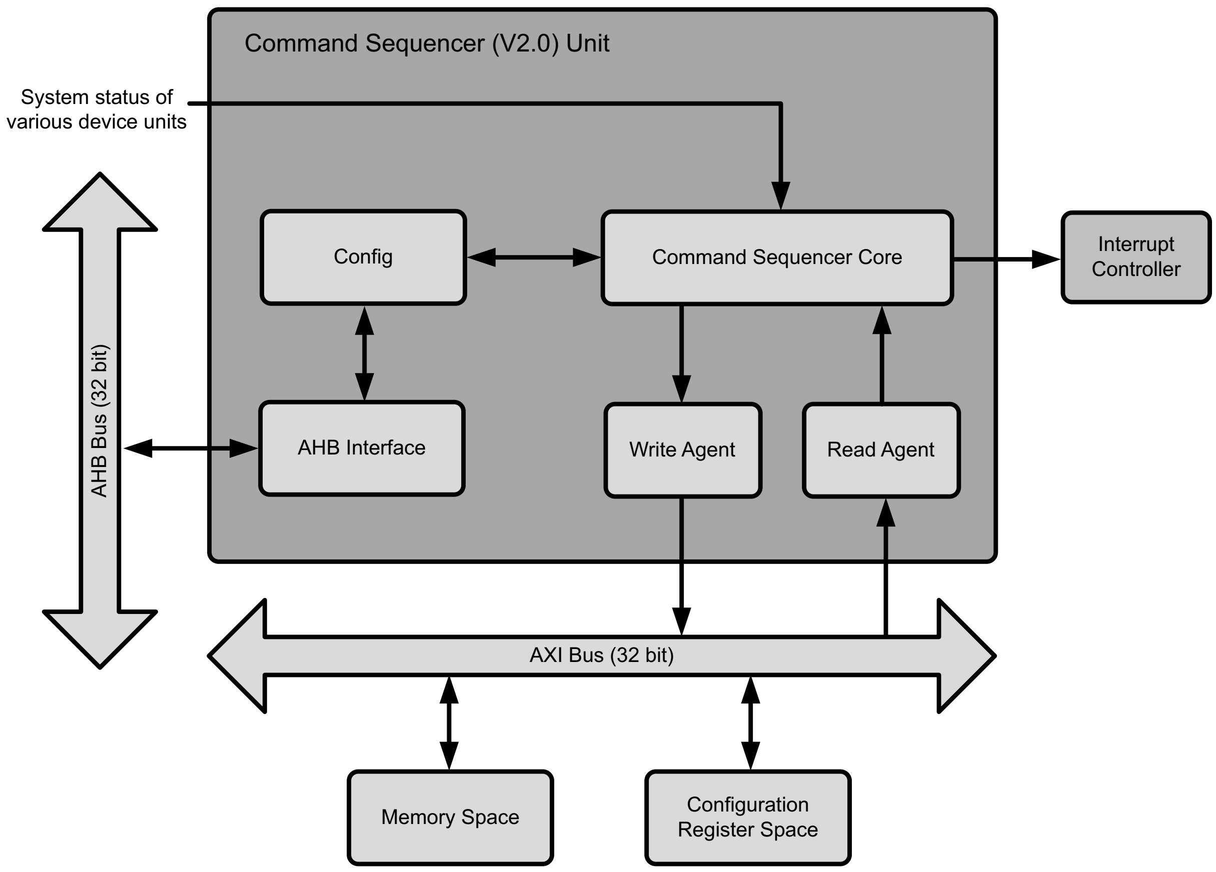


Figure 3-22: Block diagram of the Command Sequencer

3.8.3 Functional Description

3.8.3.1 Boot Sequence

After a reset, the Command Sequencer handles the system power-up sequence of MB88F33x ‘Indigo2(-x)’. This power-up behavior can be modified by external bootstrap resistors (see section “Bootstrap Configuration”). The detailed power-up sequence of MB88F33x ‘Indigo2(-x)’ is found in section “Bootstrap Configuration”.

Please refer to the Application Note “Using Bootmode 2 & 3” for deteiled information about the boot modes.

3.8.4 Processing Algorithm

3.8.4.1 Boot Procedure

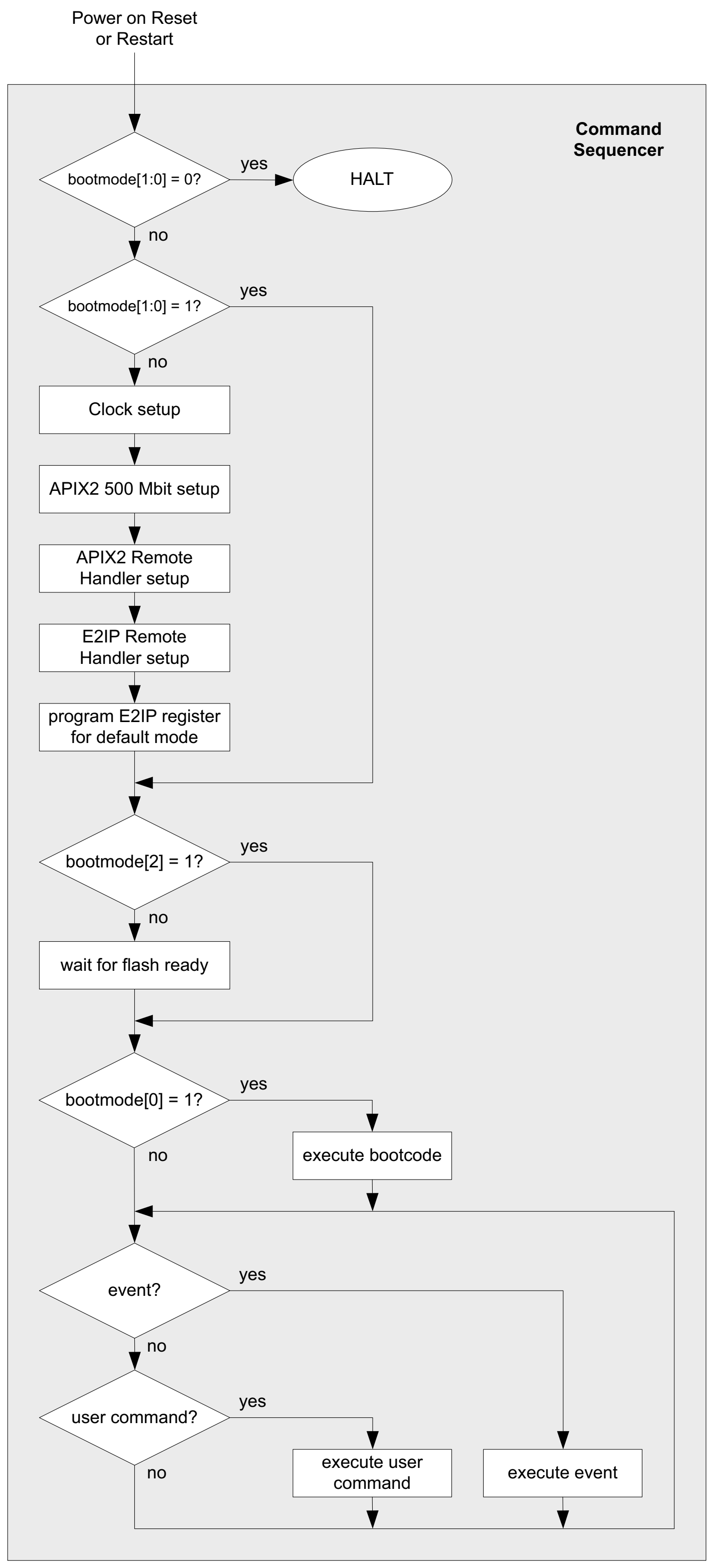


Figure 3-23: Figure 5-3: Indigo2 boot operating sequence

If not disabled, the command sequencer checks for interrupt events after executing the reset power-up sequence. Seven different interrupts can work as an event source. The interrupt source can be selected in the interrupt controller. If the event is raised and this event is enabled, the command sequencer will execute the appropriate event routine. The priority of the events is given by the position within the event input vector (event#0 has highest priority).

Table 3-6: List of functions for the supported bootmodes

Bootmode[1:0]	0	1	2	3
Function				
Do nothing (enter HALT state)	X			
Do APIX setup			X	X
Execute bootcode command list		X		X
Do event handling		X	X	X
Execute user command list if available		X	X	X

Depending on the bootstrap input signals bootmode_i, command sequencer can either start execution after reset or enters halt state (bootmode_i = 0). In this case, the core can be restarted by software later.

Starting with bootmode_i [1] = 1 will initialize and start APIX modules first before executing command lists. When bootmode[0] = 1, command sequencer checks if a command list for initial setup (bootcode) is installed. Reading a value unequal to $FFFFFFFF_H$ at the beginning of the first sector in the internal flash of the LSI means that command list for the bootcode is available. If so, this command list will be executed

3.8.4.2 Event Handling

If an event occurs, the interpreter checks if there is a command list assigned. If so, the command sequencer will start execution at this address before checking for any events or commands in the command buffer.

When executing a command list, it cannot be interrupted, even if the event has a higher priority. In this case, the occurring events will be latched and the event handling starts after running command list has finished.

After all events are handled, the command sequencer will process the user commands or wait for the next event. The jump table for the power-up sequence and the event service routines are stored in the internal flash memory (see section “Flash Memory”).

Flash Access Address Base Address= 0x17f2000	Command List to Execute
Base Address + 00_H	Bootcode
Base Address + 04_H	Event #0
Base Address + 08_H	Event #1
Base Address + $0C_H$	Event #2
Base Address + 10_H	Event #3
Base Address + 14_H	Event #4
Base Address + 18_H	Event #5
Base Address + $1C_H$	Event #6

3.8.4.3 Command Sequencer Watchdog

To prevent the system from hang-up, the command sequencer watchdog functionality can be used.

To setup and start the command sequencer watchdog timer, insert a WDS instruction in the command stream. If all parameters are 0, the command sequencer watchdog is disabled.

If the command sequencer watchdog is enabled, a CMDSEQ_WDG interrupt will be generated when watchdog-counter expires (watchdog counter = 0). Watchdog-counter can be preset by inserting WDR instructions in the command stream.

The 15-bit pre-divider offers a sizeable measurement window. At an operating frequency of 160 MHz the counter granularity varies between 6.25 ns and 204.8 us. Therefore the overall measurement window is between 6.25 ns and 15.27 h.

3.8.4.4 Command Buffer

Command buffer is used as a FIFO for command list addresses.

The command buffer is accessible at any address within the specified address range of FIFOBuffer register.

3.8.4.5 Undefined Instructions

When detecting an undefined instruction code, the command sequencer stops operation and enters error state. This will be signaled by Error0 flag (bit#25) in Status register and by triggering a CMDSEQ_ERROR interrupt.

3.8.5 Control Flow

3.8.6 Command Buffer

Data can be sent to command buffer either using a fixed- or incremented-address within FIFOBuffer address space. The number of available entries can be seen in FIFOSpace field of FIFOStatus register.

When the FIFO runs full, write data will be ignored.

Before writing data to the FIFO, the host should check for having enough space available in the FIFO by reading FIFOSpace or using the high and low-watermark interrupt mechanism. A FIFO high-watermark interrupt (CMDSEQ_HWM) will be generated when the fill counter reaches the FIFOHighWM value and a FIFO low-watermark interrupt (CMDSEQ_LWM) will be generated when the fill counter reaches the FIFOLowWM value afterwards.

FIFOHighWM value has to be greater than FIFOLowWM value.

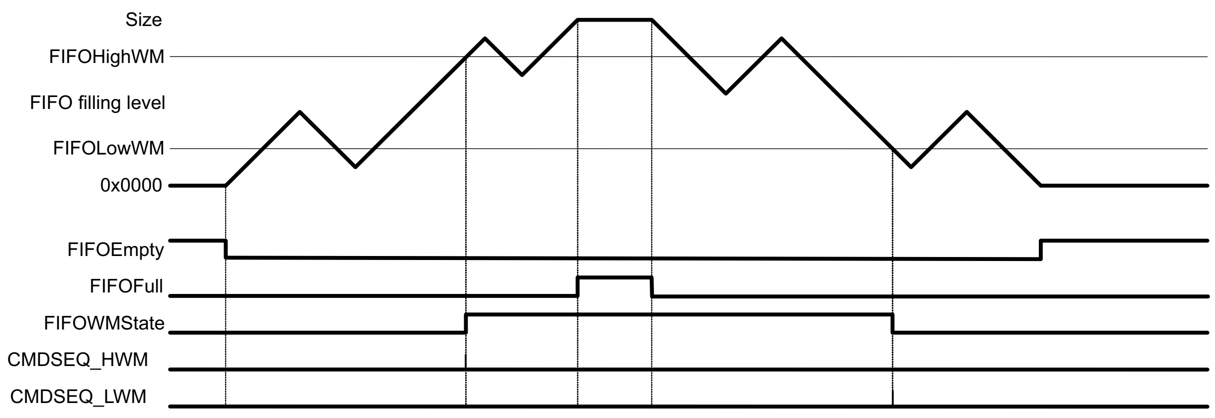


Figure 6-1: Diagram showing FIFO status signals

Command buffer can be cleared by writing a ‘1’ to FIFOClear bit in FIFOControl register. This can be used in any unintended situation to bring the command sequencer in a proper state to start with the next command list.

3.8.7 Setup Command Sequencer Watchdog

- Choose Divider value dependent on the desired measurement window.
- Calculate preset value of the command sequencer watchdog

  Counter register for the maximum time period. This can be done by:
  Counter = T_{max} * f / (2 ^ Divider) - 1
- To start the command sequencer watchdog timer send a WDS instruction with the calculated values.
- Restart the command sequencer watchdog timer by inserting a WDR command in the command stream
- If the command sequencer watchdog expires (Watchdog Counter = ‘0’), the command sequencer watchdog interrupt will be triggered.
- The command sequencer watchdog can be disabled by sending a WDS instruction with Divider and Counter parameters are all ‘0’.

3.8.8 Force Termination of Command List

When implementing polling loops within command lists it is possible that execution will never finish for some reason. In this case, command sequencer can be stopped by writing a ‘1’ to Terminate bit in the Control register. After doing this when executing the next instruction, the command sequencer stops execution and will enter HALT state. If Terminate bit is cleared before entering HALT state it can happen, that execution does not stop.

When command sequencer is in HALT state, the Terminate bit must be cleared before doing a restart.

3.8.8.1 Receiving an Error Response

When the command sequencer receives an error response (SLVERR or DECERR) from the bus system, the core stops execution immediately and enters error state. This will be signaled to the system by triggering a CMDSEQ_ERROR interrupt.

3.8.8.2 Restart when in HALT or ERROR State

When command sequencer is in HALT or ERROR state, the core can be restarted by writing a ‘1’ to the Restart bit in Control register. Command sequencer will start with the full boot procedure.

To prevent from executing some old command lists that maybe still stored in the command buffer, the FIFO should be cleared by writing a ‘1’ to the Clear bit in FIFOControl register.

3.8.9 User Instruction Set

All instructions have to be placed 32-bit aligned in memory space. Bits in instruction words marked as Reserved are not used for decoding but should be written as ‘0’ to prevent from unexpected behavior.

3.8.9.1 Abbreviations

AREG:	Address register for GETINDIRECT/PUTINDIRECT/ARGET instructions
DREG:	Working data register
EREG:	Address used by LABEL/LOOP instructions

3.8.9.2 WAIT – Wait for a Number of Microseconds

This instruction performs a delay. The number of microseconds can be specified by the Count operand.

Due to implementation issues, the overall delay can be larger (up to 3 microseconds) than the specified Count value, but will never be shorter. Note: Wait time is only correct if axi_clk is set to 160MHz.

Operation:

```
for (time = 0; time < Count; )
    wait
```

Syntax:	**Operands:**
WAIT Count	$0 \leq \text{Count} \leq 255$

Opcode:

31 … 24	23 … 16	15 … 8	7 … 0
Reserved	Count	0x00	0xE8

3.8.9.3 SWINT – Generate Interrupt

This instruction triggers a CMDSEQ_SWINT interrupt, which can be used in the interrupt controller.

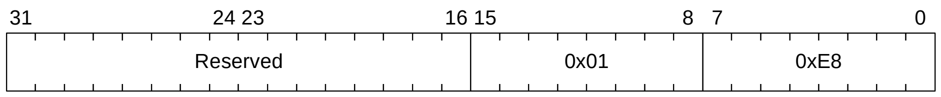
Operation:

Triggers a CMDSEQ_SWINT interrupt

Syntax:Operands:

SWINT None

Opcode:



3.8.9.4 WRITE – Write Data to Buffer

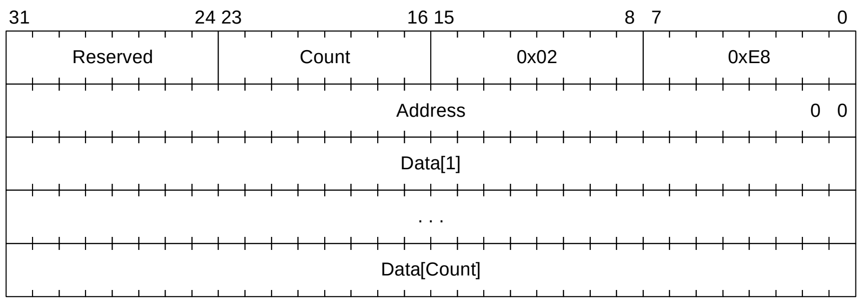
Write list of data to destination buffer.

Operation:

```
for (idx = 1; idx  ≤ Count; idx = idx + 1)
    (Address++) ← Data[idx]
```

Syntax:	**Operands:**
WRITE Count, Address, Data[ ]	1 ≤ Count ≤ 255 0 ≤ Address < 4G Data[ ]

Opcode:



3.8.9.5 OSETREG – Write Data to Buffer with Offset

Write list of data to destination buffer specified by address and offset. When Size is set to 0, byte transfers of data bit 7 down to 0 are performed and if Size is set to 1, word transfers of data bit 15 down to 0 are performed. Destination address has to be aligned to transfer size.

Operation:

```
for (idx = 1; idx ≤ Count; idx = idx + 1)
        (Address+Offset[idx]) ← Data[idx]
```

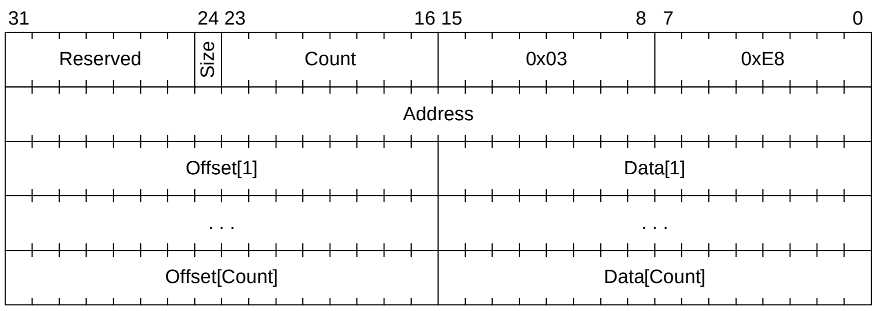
Syntax:

OSETREG Size, Count, Address, Offset[ ], Data[ ]

Operands:

Size = [0, 1]
$1 \leq$ Count ≤ 255
$0 \leq$ Address $< 4G$
Offset[ ]
Data[ ]

Opcode:



3.8.9.6 DRGET – Get DREG Data

Get data from address and store it in local DREG register. Size can take the values 0 to 2 for performing 8 bit, 16 bit and 32 bit transfers. Address has to be aligned to the transfer size.

Operation:

DREG ← (Address)

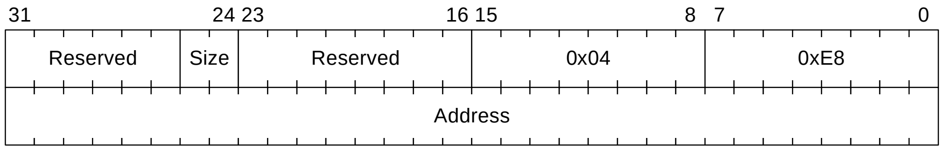
Syntax:

DRGET Size, Address

Operands:

Size = [0, 1, 2]
$0 \leq$ Address $< 4G$

Opcode:



3.8.9.7 DRPUT – Store DREG Data

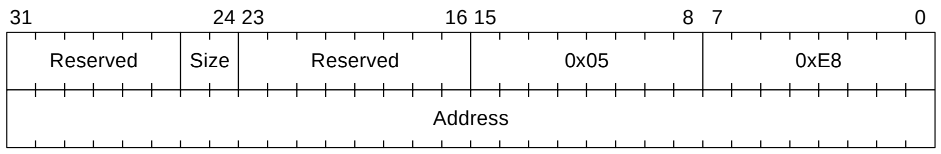
Store data from local DREG register to Address. Size can take the values 0 to 2 for performing 8 bit, 16 bit and 32 bit transfers. Address has to be aligned to the transfer size.

Operation:

(Address) DREG

Syntax:	**Operands:**
DRPUT Size, Address	Size = [0, 1, 2] $0 \leq$ Address < 4G

Opcode:



3.8.9.8 GETINDIRECT – Get DREG Data from AREG Address

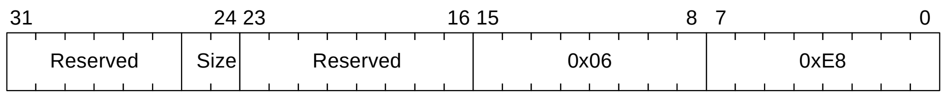
Get data from address in AREG register and store it in local DREG register. Size can take the values 0 to 2 for performing 8 bit, 16 bit and 32 bit transfers. AREG value has to be aligned to the transfer size.

Operation:

DREG $\leftarrow$ (AREG)

Syntax:	**Operands:**
GETINDIRECT Size	Size = [0, 1, 2]

Opcode:



3.8.9.9 PUTINDIRECT – Store DREG Data to AREG Address

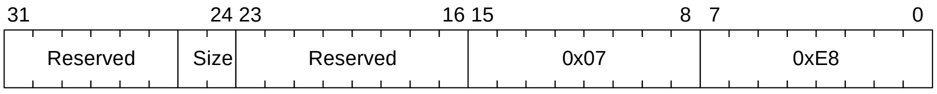
Store data from local DREG register to address in AREG register. Size can take the values 0 to 2 for performing 8 bit, 16 bit and 32 bit transfers. AREG value has to be aligned to the transfer size.

Operation:

(AREG) ← DREG

Syntax:	**Operands:**
PUTINDIRECT Size	Size = [0, 1, 2]

Opcode:



3.8.9.10 CHECK – Check Value of DREG

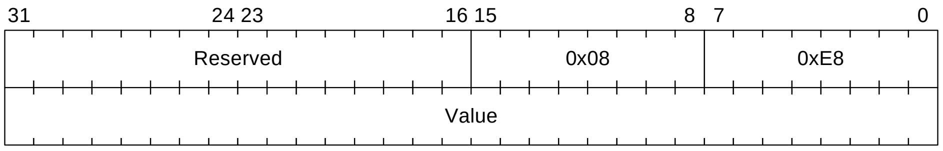
Compare bits of DREG with provided Value and skip next instruction when result is equal.

Operation:

```
if(DREG == Value)
    PC ← PC + sizeof(next instruction)
```

Syntax:	**Operands:**
CHECK Value	0 <= Value < 4G

Opcode:



3.8.9.11 ARGET – Get AREG Data

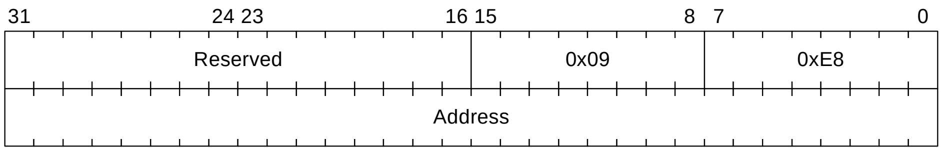
Get data from address and store it in local AREG register.

Operation:

AREG ← (Address)

Syntax:	**Operands:**
ARGET Address	0 ≤ Address < 4G

Opcode:



3.8.9.12 LABEL – Store current address

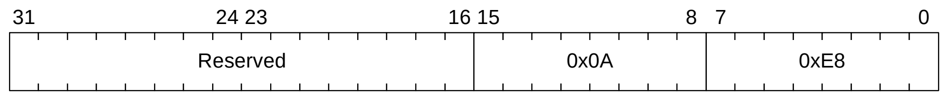
Store current program counter address to EREG register. This can be used for implementation of backward loops.

Operation:

EREG ← PC

Syntax:	**Operands:**
LABEL	None

Opcode:



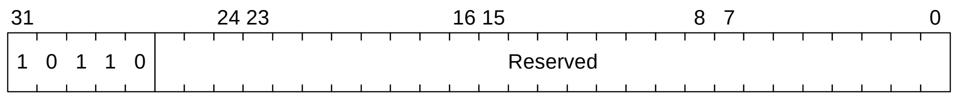
3.8.9.13 LOOP – Jump to label

Continue execution at address stored in EREG register. This can be used for implementation of backward loops.

Operation:
PC ← EREG

Syntax:	**Operands:**
LOOP	None

Opcode:



3.8.9.14 WDR – Command Sequencer Watchdog Reset

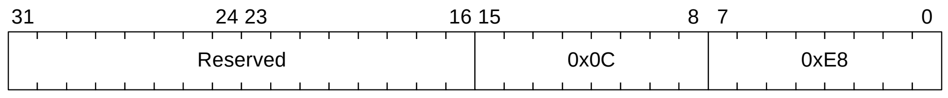
This instruction resets the command sequencer watchdog timer. It must be executed within a limited time given by the watchdog load register and the divider value.

Operation:

Watchdog timer restart

Syntax:	**Operands:**
WDR	None

Opcode:



3.8.9.15 WDS – Command Sequencer Watchdog Setup

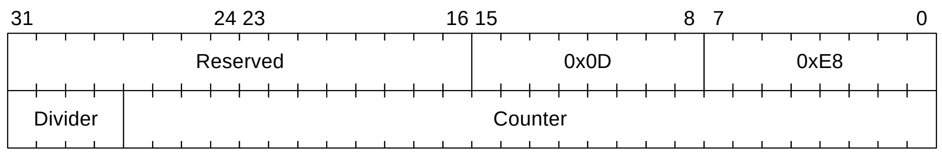
This instruction does the setup of the command sequencer watchdog timer. If Divider and Counter parameters are all ‘0’, the command sequencer watchdog timer will be disabled, otherwise timer will be started with the specified values. Doing a new WDS instruction while timer is running also starts the timer with the new values immediately.

Operation:

```
cnt ← Counter
div ← 2 ^ Divider
if (Counter == 0)
enable ← 0
else
enable ← 1
```

Syntax:	**Operands:**
WDS Divider, Counter	$0 \le$ Divider < 16
	$0 \le$ Counter < 256M

Opcode:



3.8.9.16 JUMP – Jump to Address

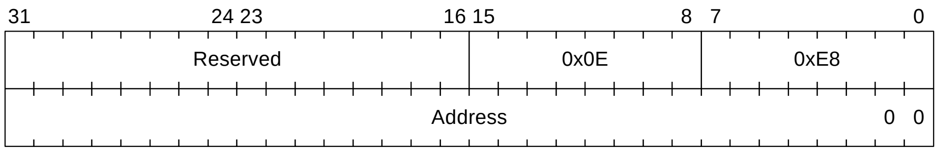
Continue execution at provided Address. This instruction is like a jump and won’t return.

Operation:

PC ← Address

Syntax:	**Operands:**
JUMP Address	0 ≤ Address < 4G

Opcode:



3.8.9.17 END – End of Command List

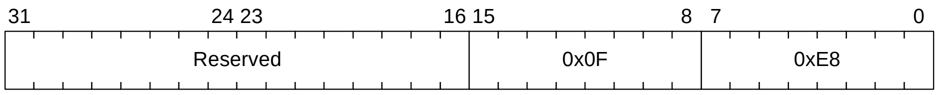
Stop execution of the current command.

Operation:

Return to main loop

Syntax:	**Operands:**
END	None

Opcode:



3.8.9.18 OR – Logical or

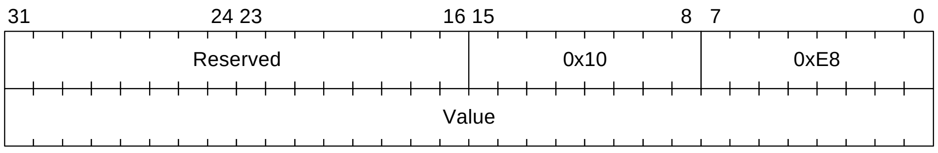
Logical or of DREG content with value.

Operation:

DREG ← DREG | Value

Syntax:	**Operands:**
OR Value	Value = 0 ≤ Value < 4G

Opcode:



3.8.9.19 AND – Logical and

Logical and of DREG content with value.

Operation:

DREG ← DREG & Value

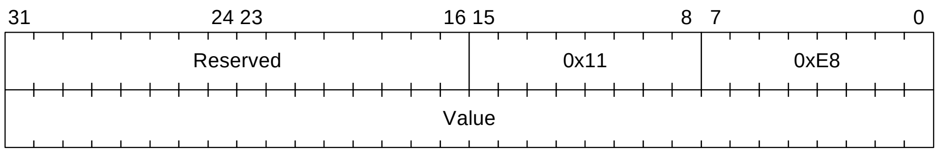
Syntax:

AND Value

Operands:

Value = 0 ≤ Value < 4G

Opcode:



3.8.9.20 ADD – Add Value to DREG

Add value to DREG content.

Operation:

DREG ← DREG + Value

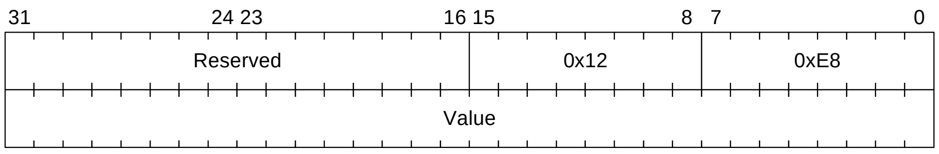
Syntax:

ADD Value

Operands:

Value = 0 ≤ Value < 4G

Opcode:



3.8.9.21 XOR – Logical Exclusive OR

Logical exclusive or of DREG content with value.

Operation:

DREG ← DREG ^ Value

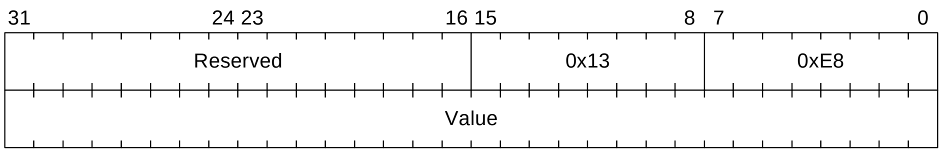
Syntax:

XOR Value

Operands:

0 ≤ Value < 4G

Opcode:



3.8.9.22 SHR – Logical Shift Right

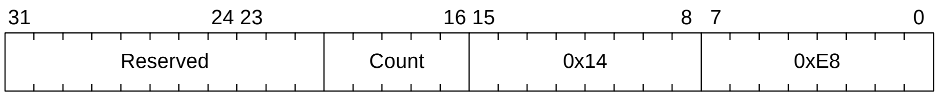
Logical shift right of DREG content.

Operation:

DREG ← DREG >> Count

Syntax:	**Operands:**
SHR Count	$0 \le \text{Count} < 32$

Opcode:



3.8.9.23 SHL – Logical Shift Left

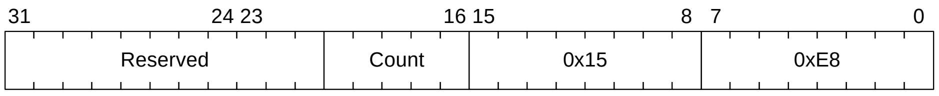
Logical shift left of DREG content.

Operation:

DREG ← DREG << Count

Syntax:	**Operands:**
SHR Count	$0 \le \text{Count} < 32$

Opcode:



3.8.9.24 JMPR – Jump Relative

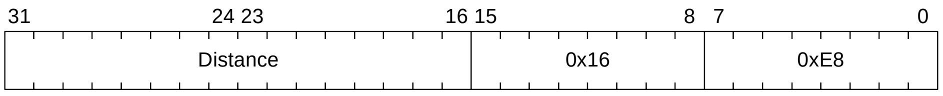
Continue execution at provided distance.

Operation:

PC ← PC + Distance * 4

Syntax:	**Operands:**
JMPR Distance	$-32k \le \text{Distance} < 32k$

Opcode:



3.8.9.25 FILL – Constant Fill

This instruction fills a memory region with a constant value.

Operation:

For (idx = 1; idx ≤ Count; idx = idx + 1)
(Address++) ← Value

Syntax:	**Operands:**
FILL Count, Address, Value	0 ≤ Count < 64k 0 ≤ Address < 4G Value

Opcode:

31 … 24 23 … 16	15 … 8	7 … 0
Count	0x17	0xE8
Address … 0 0		
Value		

3.8.9.26 NOT – Bitwise not

Bitwise logical not of DREG content.

Operation:

DREG ← ~DREG

Syntax:	**Operands:**
NOT	None

Opcode:

31 … 24 23 … 16	15 … 8	7 … 0
Reserved	0x18	0xE8

3.8.9.27 DRLOAD – Load DREG Data

Load value into DREG register.

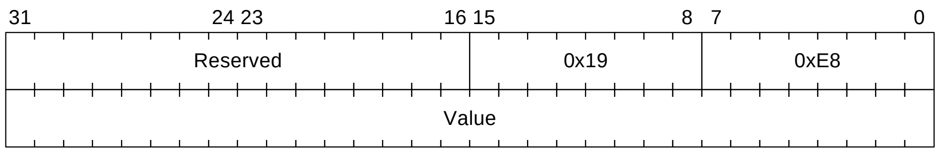
Operation:

DREG ← Value

Syntax: DRLOAD Value

Operands: 0 ≤ Value < 4G

Opcode:



3.8.9.28 DRSAVE – Save DREG Data to Buffer

Save content of DREG register to local buffer. 8 buffers are implemented and can be selected by Index parameter.

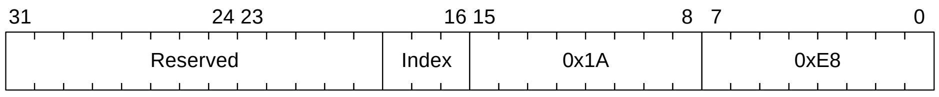
Operation:

DREG → Buffer[Index]

Syntax: DRSAVE Index

Operands: 0 ≤ Index ≤ 7

Opcode:



3.8.9.29 DRRESTORE – Restore DREG Data from Buffer

Restore content of DREG register from local buffer. 8 buffers are implemented and can be selected by Index parameter.

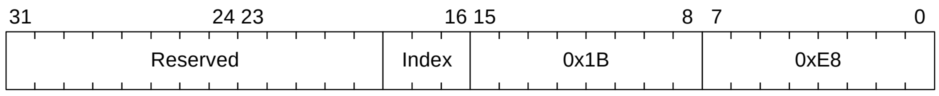
Operation:

DRRESTORE ← Buffer[Index]

Syntax: DRRESTORE Index

Operands: 0 ≤ Index ≤ 7

Opcode:



3.8.10 Command Sequencer Register Overview

Table 3-7: Registers Overview

Base Address(es)	Instance no 0: BASEADDR="0002C000"	
Absolute Address	**Register Name**	**Register Description**
BASEADDR + 0x0000	FIFOBuffer	Command input buffer
BASEADDR + 0x0100	FIFOStatus	Status register
BASEADDR + 0x0104	FIFOControl	Control register
BASEADDR + 0x0108	FIFOWatermarkControl	Watermark Control register
BASEADDR + 0x010C	Reserved	Do not modify
BASEADDR + 0x0110	Reserved	Do not modify
BASEADDR + 0x0200	Reserved	Do not modify
BASEADDR + 0x0204	Status	Status register
BASEADDR + 0x0208	Control	Control register
BASEADDR + 0x020C	Reserved	Do not modify
BASEADDR + 0x0210	Reserved	Do not modify
BASEADDR + 0x0214	Reserved	Do not modify
BASEADDR + 0x0218	Reserved	Do not modify
BASEADDR + 0x021C	Reserved	Do not modify

3.9 Configuration FIFO

The configuration FIFO can be used to decouple a host CPU command-stream from a peripheral command stream. This is necessary, for example to allow the isochronous reconfiguration of special peripherals such as stepper motor controllers if the command stream from the host CPU is not "jitter free", that means communication is disturbed and requires repeated transmissions.

A trigger input is provided for each of the 8 FIFO’s. Each trigger signal starts a transfer from the respective FIFO to the programmed destination address. If multiple FIFOs are triggered at the same time, each trigger request is serviced using a fixed priority scheme. Lower FIFO numbers have higher priority and are serviced first.

3.9.1 Features of the Configuration FIFO

- 8 configurable FIFOs.
- Use of one shared memory.
- The depth of each FIFO is configurable.
- Double buffer mode for reliable data transfers into the FIFO
- FIFO upper and lower threshold interrupts.
- AHB slave interface for FIFO data input, and the register is write/read.
- AHB master interface for FIFO data output (only write is supported).
- Trigger input for each FIFO output.
- Software Trigger for each FIFO output
- Simple local DMA functionality at data output programmable target address different addressing modes (including fixed)
- Meta Commands: dynamic reconfiguration of the target address and addressing modes during a FIFO output transfers
- Enhanced status read back: Read/Write pointers and byte counters, status registers

3.9.2 Block Diagram

Figure 3-24 on page 55 shows configuration FIFO’s block diagram.

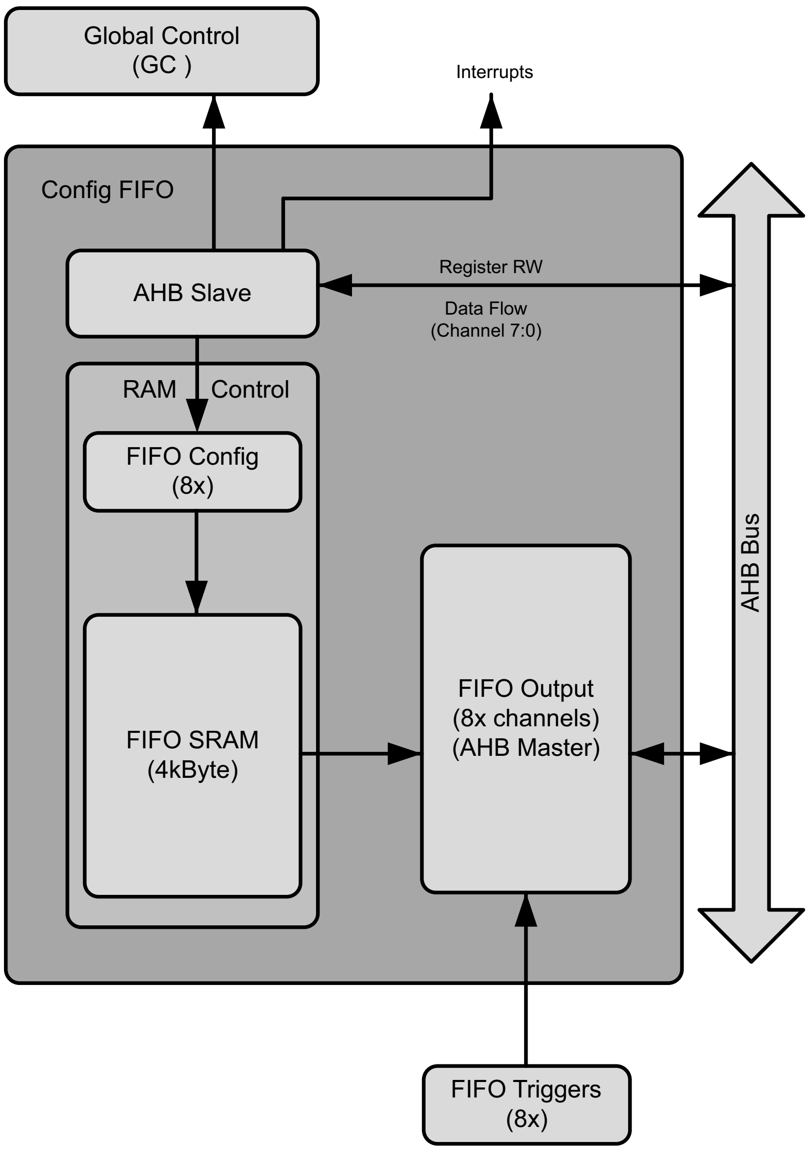


Figure 3-24: Block diagram

3.9.3 Function Description

The Configuration FIFO implements 8 configurable FIFOs, which share one 4k Byte Memory. For each of the 8 FIFOs a trigger input is provided. Each trigger signal starts a transfer from the respective FIFO to the programmed destination address. If multiple FIFOs are triggered at the same time the trigger requests are serviced by a fixed priority scheme. Lower FIFO numbers have higher priority and are serviced first.

3.9.3.1 AHB Slave Interface

The AHB slave interface is used to write and read registers and to input FIFO data. Registers exist for 8 channels. If the FIFO is full, write data transfers are not executed. However, HREADY is asserted. It is necessary to monitor this situation via an interrupt signal.

3.9.3.2 AHB Master Interface

Data is output from a FIFO when this is requested from a channel via a trigger request.

In the case of FFEmptyMode=0, and if the FIFO is emptied, the FFEnO bit of a FFCfg register will be automatically set to 0. If a FIFO is emptied, it is not possible to write a 1 to the FFEnO bit. The application software must first write data to the FIFO and must then set FFEnO back to 1.

If the number of transfers (TransferNumber of a TransferCfg register) is over the FILL Level (FillLevel of a FFStatus register), an AHB master will not transfer, even if a trigger request signal is initiated.

In the following example, if the left side of the equation is too large, an AHB master does not transfer.

(((TransferNumber-1)+1) << transferwidth) > FILL Level

TransferNumber=0 is 64 transfers.

In the case of FFEmptyMode=0, if a trigger request is received and the FILL Level is the same as the number transfers set, then FFEnO is automatically set to 0 after the transfer has completed.

Burst mode for the AHB Master transfer is automatically set depending on the number of transfers.

- In the case of TransferINCR=0 of TransferCfg
  - o_mHBURST is always transferred using SINGLE regardless of the number of transfers.
- In the case of TransferINCR=1 of TransferCfg
  - When TransferNumber is 1, o_mHBURST becomes SINGLE.
  - When TransferNumber is 4, o_mHBURST becomes INCR4.
  - When TransferNumber is 8, o_mHBURST becomes INCR8.
  - When TransferNumber is 16, o_mHBURST becomes INCR16.

As for other values, o_mHBURST becomes INCR (Indefinite length burst). When a transfer address exceeds 1KB, transfer is not executed using INCR4 and INCR8 and INCR16. It is transferred using INCR.

3.9.3.3 FIFO Memory

The FIFO memory is used by the AHB slave interface for FIFO data input. To write to the FIFO three different types of registers exist:

1. Data Registers (FFDataInLx, FFDataInUx)
2. Last Data in Registers (FFEDataInLx, FFEDataInUx)
3. Meta Command Registers (MetaDestAddressx, MetaCfgx)

The first two registers are used for "Double Buffer Mode". The third type is used for dynamic reconfiguration of the AHB destination address and transfer mode.

Double Buffer Mode

Every FIFO supports "Double Buffer Mode" which uses two pointers into the FIFO. The flow is as follows; the software writes data to the input registers (FFDataInL0, FFDataInU0 - in case of FIFO channel 0). The last bytes of data are then written to the "Last Data In Registers" (in the case of FIFO channel 0 these are FFEDataInL0 and FFEDataInU0). This will update the values of the internal write pointers, simultaneously validating the written data. If the last write is not executed to the "Last Data In Registers", the transmitted data will be discarded.

The internal operation of the FIFO is explained in the next figure.

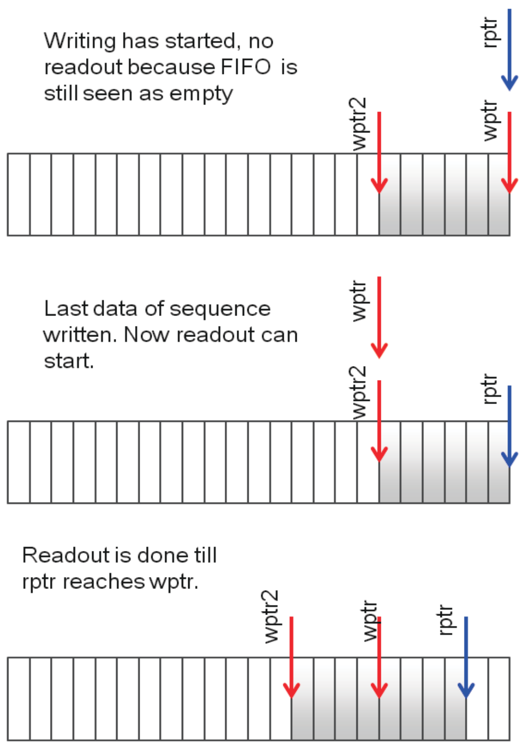

Writing has started, no readout because FIFO is still seen as empty
rptr
wptr2
wptr
Last data of sequence written. Now readout can start.
wptr
wptr2
rptr
Readout is done till rptr reaches wptr.
wptr2
wptr
rptr

Figure 3-25: Writing FIFO in Double Buffer Mode

This mode also supports checking the amount of data written to the FIFO. At the beginning of each transmission, a size register (FFDataSize0 in case of FIFO channel 0) can be set. This register holds the number of **Bytes** that will be written to the FIFO in the subsequent transmission.

At the end of each transmission an internal counter is compared with the size register and if it matches a "Data Written" interrupt is fired. If this interrupt is not received by the transceiver (host CPU), the transmission to the FIFO was not successful and the received data will not be written to the FIFO. In this case the software can run error correction routines.

Meta Commands (Dynamic AHB Reconfiguration)

A special case are the "Meta Commands". These commands can be used to dynamically reconfigure the AHB master destination address and transfer mode on-the-fly during a transmission.

When writing data into a FIFO, these commands can be embedded in the data stream. A Meta Command can be written to the FIFO by sending data to the registers "MetaDestAddressX" and "MetaCfgX". For a description of the registers and their values, please refer to the section “Configuration FIFO Register Overview”.

Temporary Latch Feature

The ‘Temporary Latch Feature’ can be used for all of the above register writes.

If, for example, 2 data words (8 bytes) from the FFDataInL and FFDataInU registers are received, these are written to the FIFO. Writes to the FIFO are done in 2 data word units. The internal FIFO memory is only updated if both registers are written. Otherwise, the FIFO memory remains unchanged.

This mode is referred to as the “temporary latch feature” and is very effective if the FFTempMode bit of a FFCfg register is 1.

In the case of FFTempMode=0, if 1 data block (byte, hword, word) is received, it is written to the FIFO. A receiving address writes data to the address held in FFDataInL. In the case of FFTempMode=0, the application software needs to check the TransferSize setting in the corresponding TransferCfg register. Additionally, it is necessary to control the flow so that the write and read sizes are the same.

In both cases, if the FIFO is full, receive data is not written to the FIFO. In the case of channel 0, the application software must be written in such a way, that it can write to the ensuing addresses:

FFTempMode=0

Table 3-8: Writing the FIFO Memory with Latch Mode disabled

Input Size is BYTE	140h receive → FIFO Write! → 148h receive → FIFO Write! → (Data bit 7–0 is used)
Input Size is HWORD	140h receive → FIFO Write! → 148h receive → FIFO Write! → (Data bit 15–0 is used)
Input Size is WORD	140h receive → FIFO Write! → 148h receive → FIFO Write! → (Data bit 31–0 is used)

FFTempMode=1 (temporary latch feature mode)

Table 3-9: Writing the FIFO Memory with Latch Mode enabled

Input Size is BYTE	148h → 149h → 14Ah → … 14Eh → 14Fh → FIFO Write! → 148h …
Input Size is HWORD	148h → 14Ah → 14Ch → 14Fh → FIFO Write! → 148h …
Input Size is WORD	148h → 14Ch → FIFO Write! → 148h → 14Ch → FIFO Write! → 148h …

NOTE Even for the temporary latch mode, the last data has to be written to the data end registers, e.g., 0x140 and 0x144 (in case of WORD)

Writing 8bit/16bit Register

When a register of an RBUS-peripheral is written, the data in a 32bit configurated ConfigFIFO must be aligned correctly.

Example for a 16bit access to the SMC Register:

Data to be written in the SMC Register (SMC is connected to the RBUS)

addr:	data:
0x00080006	0x80
0x00080008	0x1212
0x0008000A	0x4000
0x0008000C	0x100

Data for the ConfigFIFO has to be aligned as follows:

addr:	data:
0x00080006	0x00800000
0x00080008	0x00001212
0x0008000A	0x40000000
0x0008000C	0x00000100

Important Notes for the FIFO Memory:

- An AHB-master interface is implemented for FIFO data output.
- The shared memory (fixed 4kB) is used by up to 8 channels (programmable).
- The FIFO area of each channel can be arbitrarily changed.
- An area is set with LowerBoundAdr and UpperBoundAdr register of FFB.
- An area is specified using a double word address.
- Do not overlap any channel boundary areas in the settings.
- If areas overlap, FIFO operation may malfunction.
- The maximum total area for the channels is 4kB.
- The maximum size for one channel is 4kByte minus 8Byte (511 x 8Byte).
- A wraparound address is not permitted (e.g. UpperBoundAdr=0x010, LowerBoundAdr=0xf0).

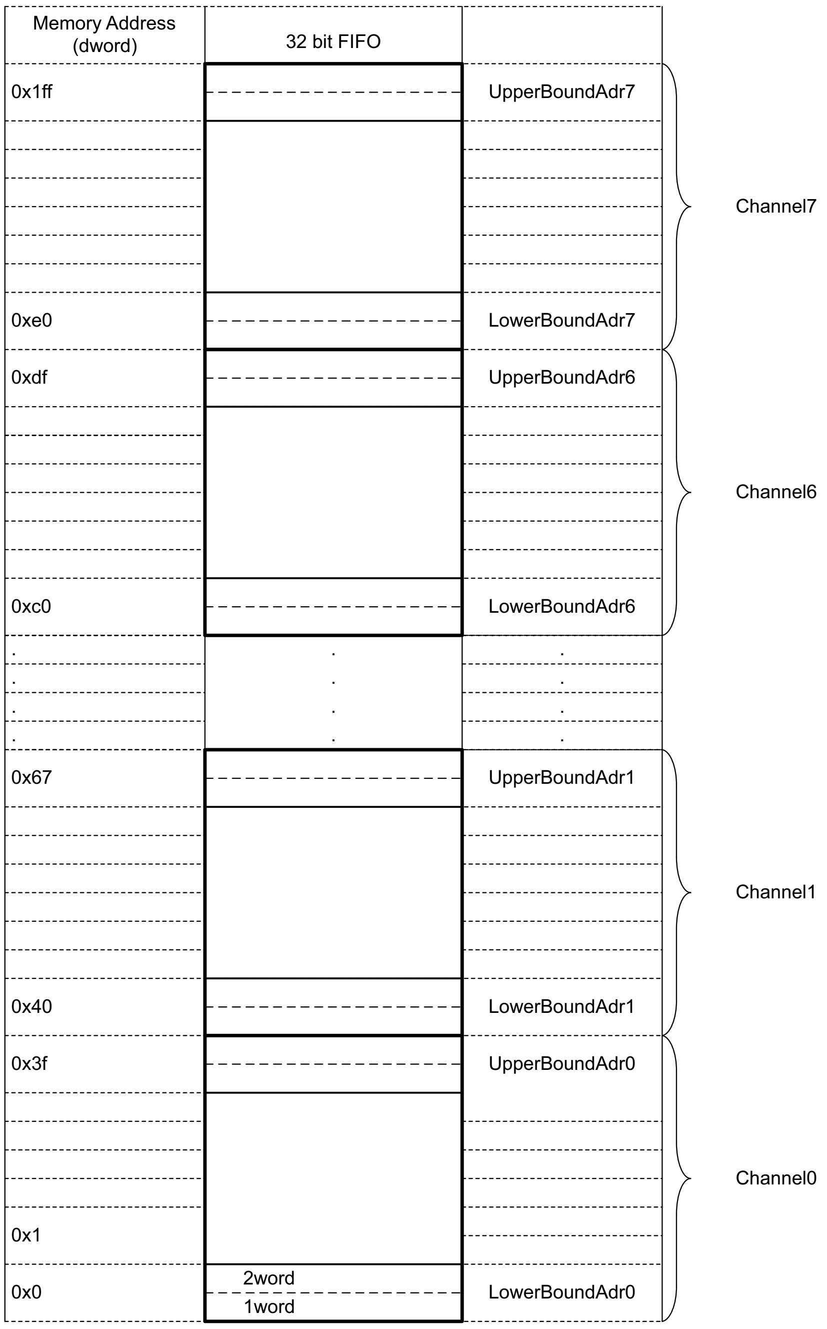


Figure 3-26: FIFO Address setting example

3.9.3.4 Trigger Request

A trigger signal uses positive edge detection. If a trigger signal is a toggle output, the timer preset value is halved and it is necessary to use it as a positive edge output.

Each trigger request is serviced by a fixed priority scheme.

The signal connected with Ch0 (i_TRIGGER[0]) has higher priority by default (initial value).The signal connected with Ch7 (i_TRIGGER[7]) has lowest priority.

If two or more trigger requests occur simultaneously, the channel with the highest priority will be processed. A trigger's priority can be set and changed via the CHPriority register. The value of a CHPriority register becomes a channel of the highest priority.

For example:

CHPriority Register= 010b:

High Priority Ch2 – Ch3 – Ch4 – Ch5 – Ch6 – Ch7 – Ch0 – Ch1 Low Priority.

Important Notes:

ConfigFIFO detects a trigger by the rising edge of i_Trigger. If a trigger is a level interrupt, the ConfigFIFO will only be triggered at the first rising edge. In this case, it is necessary to clear the interrupt again.
Possible ways to clear an interrupt are shown below:
- The software writes a clear
- Automatic hardware clear (e.g. for ADC)
- Clear command in ConfigFIFO

Limitations

– Trigger input. The trigger input for the 8 config fifos can be either any one of the 16 RLT output pulses or any one of the interrupt signals. If a level interrupts is used for triggering the config fifo, this interrupt has to be cleared within the sequence of the config fifo, otherwise no retrigger is possible.

3.9.3.5 Interrupt

The interrupt mechanism has the following output signals:

The following items are interrupts to the Remote Handler and Interrupt Control

The interrupt signal output (o_VAR_INT) is active high due to the following factor:
- AHB master received HRESP error.

The Lower Threshold level interrupt signal output (o_LT_INT[7:0]) is active high due to the following factor:
- Under FIFO (Ch7-0) fill level lower threshold.

The Upper Threshold level interrupt signal output (o_UT_INT[7:0]) is active high due to the following factor:
- Over FIFO (Ch7-0) fill level upper threshold.

The Underflow interrupt signal output (o_UF_INT[7:0]) is active high due to the following factor:
- FIFO (Ch7-0) Underflow (empty).

The Overflow interrupt signal output (o_OF_INT[7:0]) is active high due to the following factor:
- FIFO (Ch7-0) Overflow (full).

The collective output of all the interrupt signals combined (o_ALL_INT).
- Each Channel (Ch0-7) HRESP error + Lower Threshold + Upper Threshold + Underflow + Overflow.

- A DW interrupt is fired in double buffer mode when data is written to the Last Data Register and the Size Register is = 0 OR if the Last Data Register is written and the number of bytes written matches the number predefined in the Size Register (!=0)

Limitations

– All interrupts are always automatically cleared. Software clear is not supported. Interrupt status has to be read from the Global Interrupt Controller.

Important Notes:

Software must ensure that the overflow and underflow interrupts are never issued.

If for some reason an Overflow interrupt is asserted during a write to the FIFO, the current write will be invalidated. This means that the internal pointers and counters will be reset to the last position prior to the current write that caused the interrupt.

If an underflow (the FIFO is empty) occurs, the FIFO has to be re-enabled again after data has been written to it.

3.9.4 Configuration FIFO Register Overview

Table 3-10: Registers Overview

Base Address(es)	Instance no 0: BASEADDR="0002E000"	
Absolute Address	**Register Name**	**Register Description**
BASEADDR + 0x0000	FFISTS	Interrupt status flags.
BASEADDR + 0x0004	FFIEN	Interrupt Enable register. '1' enables.
BASEADDR + 0x0008	Reserved	Do not modify
BASEADDR + 0x000C	Reserved	Do not modify
BASEADDR + 0x0010	CFG_IDLE	The state of all the channels is shown.
BASEADDR + 0x0014	CHPriority	Priority of Trigger Request.
BASEADDR + 0x0100	SW_RT0	Software Reset and Trigger
BASEADDR + 0x0104	FFCfg0	The function of FIFO is set up.
BASEADDR + 0x0108	FFB0	This address sets the boundary address of sharing FIFO. Do not overlap that each channels boundary area setting. When the area overlaps, the operation guarantee cannot be done. When using this channel, it sets up so that it may be set to 'UpperBoundAdr >= LowerBoundAdr'. The maximum size for 'UpperBoundAdr - LowerBoundAdr' is 0x1FE.
BASEADDR + 0x010C	FFT0	Sets the threshhold level for the corresponding FIFO in Bytes.
BASEADDR + 0x0110	DestAddress0	Local AHB-master transfer Destination address
BASEADDR + 0x0114	AdrCfg0	Address generation Configuration
BASEADDR + 0x0118	TransferCfg0	Local AHB master transfer Configuration
BASEADDR + 0x011C	FFStatus0	Status Register
BASEADDR + 0x0120	FFISTS_TH0	Interrupt status flags, a '1' signifies that the corresponding interrupt condition occurred (even if interrupt is disabled), write '1' clears the flag. Even if the factor of Interrupt is cancelled (e.g. not empty), Or if an external clear signal is set to 1, this bit will be cleared automatically. When the rising edge and StatusClear of an interrupt factor happen simultaneously, Status Register gives priority to an interrupt factor.
BASEADDR + 0x0124	FFIEN_TH0	Interrupt Enable register. '1' is enable.
BASEADDR + 0x0128	Reserved	Do not modify
BASEADDR + 0x012C	FFIEN_DW0	Interrupt Enable register. '1' is enable.
BASEADDR + 0x0130	READ_CNT0	32 Bit Counter. Is incremented with every read from the FIFO. This does not correspond with the size of the read (e.g. WORD, HALF WORD or BYTE). For each access regardless of size the counter is incremented by one. An overflow will occur at the boundary of 32 Bit.
BASEADDR + 0x0134	WRITE_CNT0	32 Bit Counter. Is incremented with every write from the FIFO. This does not correspond with the size of the write (e.g. WORD, HALF WORD or BYTE). For each access regardless of size the counter is incremented by one. An overflow will occur at the boundary of 32 Bit.
BASEADDR + 0x0138	INT_READ_ADDR0	Readback register for the read address pointer to the FIFO. NOTE: This is an internal address. It is derived from the lower bound address but translated internally (left shift by three '0'). The value is the next address that will be read from. For an example refer to the chapter 'Readback address pointers'.
BASEADDR + 0x013C	INT_WRITE_ADDR0	Readback register for the current write address pointers to the FIFO. NOTE: These are internal addresses. They are derived from the lower bound address but translated internally (left shift by three '0'). The value is the last address that has been written to. For an example refer to the chapter 'Readback address pointers'.

Table 3-10: Registers Overview

Base Address(es)	Instance no 0: BASEADDR="0002E000"	
Absolute Address	**Register Name**	**Register Description**
BASEADDR + 0x0140	FFEDataInL0	Last FIFO Data In Lower
BASEADDR + 0x0144	FFEDataInU0	Last FIFO Data In Upper
BASEADDR + 0x0148	FFDataInL0	FIFO Data In Lower
BASEADDR + 0x014C	FFDataInU0	FIFO Data In Upper
BASEADDR + 0x0150	FFDataSize0	Number of bytes that will be written to the FIFO
BASEADDR + 0x0154	MetaDestAddress0	Local AHB-master transfer Destination address for Meta Command operation. NOTEs: The first and the last command to in a transmission sequence cannot be a meta command! The Meta address register has to be always written prior to the Meta Config register! For transmissions sequences that contain meta command only indefinite length bursts can be configured (for both fields 'MetaTransferINCR' and 'TransferINCR')!
BASEADDR + 0x0158	MetaCfg0	Local AHB master transfer and address generation Configuration for Meta Command operation. NOTEs: The first and the last command to in a transmission sequence cannot be a meta command! The Meta address register has to be always written prior to the Meta Config register! For transmissions sequences that contain meta command only indefinite length bursts can be configured (for both fields 'MetaTransferINCR' and 'TransferINCR')!
BASEADDR + 0x0180	SW_RT1	Software Reset and Trigger
BASEADDR + 0x0184	FFCfg1	The function of FIFO is set up.
BASEADDR + 0x0188	FFB1	This address sets the boundary address of sharing FIFO. Do not overlap that each channels boundary area setting. When the area overlaps, the operation guarantee cannot be done. When using this channel, it sets up so that it may be set to 'UpperBoundAdr >= LowerBoundAdr'. The maximum size for 'UpperBoundAdr - LowerBoundAdr' is 0x1FE.
BASEADDR + 0x018C	FFT1	Sets the threshhold level for the corresponding FIFO in Bytes.
BASEADDR + 0x0190	DestAddress1	Local AHB-master transfer Destination address
BASEADDR + 0x0194	AdrCfg1	Address generation Configuration
BASEADDR + 0x0198	TransferCfg1	Local AHB master transfer Configuration
BASEADDR + 0x019C	FFStatus1	Status Register
BASEADDR + 0x01A0	FFISTS_TH1	Interrupt status flags, a '1' signifies that the corresponding interrupt condition occurred (even if interrupt is disabled), write '1' clears the flag. Even if the factor of Interrupt is cancelled (e.g. not empty), Or if an external clear signal is set to 1, this bit will be cleared automatically. When the rising edge and StatusClear of an interrupt factor happen simultaneously, Status Register gives priority to an interrupt factor.
BASEADDR + 0x01A4	FFIEN_TH1	Interrupt Enable register. '1' is enable.
BASEADDR + 0x01A8	Reserved	Do not modify
BASEADDR + 0x01AC	FFIEN_DW1	Interrupt Enable register. '1' is enable.
BASEADDR + 0x01B0	READ_CNT1	32 Bit Counter. Is incremented with every read from the FIFO. This does not correspond with the size of the read (e.g. WORD, HALF WORD or BYTE). For each access regardless of size the counter is incremented by one. An overflow will occur at the boundary of 32 Bit.
BASEADDR + 0x01B4	WRITE_CNT1	32 Bit Counter. Is incremented with every write from the FIFO. This does not correspond with the size of the write (e.g. WORD, HALF WORD or BYTE). For each access regardless of size the counter is incremented by one. An overflow will occur at the boundary of 32 Bit.

Table 3-10: Registers Overview

Base Address(es)	Instance no 0: BASEADDR="0002E000"	
Absolute Address	**Register Name**	**Register Description**
BASEADDR + 0x01B8	INT_READ_ADDR1	Readback register for the read address pointer to the FIFO. NOTE: This is an internal address. It is derived from the lower bound address but translated internally (left shift by three '0'). The value is the next address that will be read from. For an example refer to the chapter 'Readback address pointers'.
BASEADDR + 0x01BC	INT_WRITE_ADDR1	Readback register for the current write address pointers to the FIFO. NOTE: These are internal addresses. They are derived from the lower bound address but translated internally (left shift by three '0'). The value is the last address that has been written to. For an example refer to the chapter 'Readback address pointers'.
BASEADDR + 0x01C0	FFEDataInL1	Last FIFO Data In Lower
BASEADDR + 0x01C4	FFEDataInU1	Last FIFO Data In Upper
BASEADDR + 0x01C8	FFDataInL1	FIFO Data In Lower
BASEADDR + 0x01CC	FFDataInU1	FIFO Data In Upper
BASEADDR + 0x01D0	FFDataSize1	Number of bytes that will be written to the FIFO
BASEADDR + 0x01D4	MetaDestAddress1	Local AHB-master transfer Destination address for Meta Command operation. NOTEs: The first and the last command to in a transmission sequence cannot be a meta command! The Meta address register has to be always written prior to the Meta Config register! For transmissions sequences that contain meta command only indefinite length bursts can be configured (for both fields 'MetaTransferINCR' and 'TransferINCR')!
BASEADDR + 0x01D8	MetaCfg1	Local AHB master transfer and address generation Configuration for Meta Command operation. NOTEs: The first and the last command to in a transmission sequence cannot be a meta command! The Meta address register has to be always written prior to the Meta Config register! For transmissions sequences that contain meta command only indefinite length bursts can be configured (for both fields 'MetaTransferINCR' and 'TransferINCR')!
BASEADDR + 0x0200	SW_RT2	Software Reset and Trigger
BASEADDR + 0x0204	FFCfg2	The function of FIFO is set up.
BASEADDR + 0x0208	FFB2	This address sets the boundary address of sharing FIFO. Do not overlap that each channels boundary area setting. When the area overlaps, the operation guarantee cannot be done. When using this channel, it sets up so that it may be set to 'UpperBoundAdr >= LowerBoundAdr'. The maximum size for 'UpperBoundAdr - LowerBoundAdr' is 0x1FE.
BASEADDR + 0x020C	FFT2	Sets the threshhold level for the corresponding FIFO in Bytes.
BASEADDR + 0x0210	DestAddress2	Local AHB-master transfer Destination address
BASEADDR + 0x0214	AdrCfg2	Address generation Configuration
BASEADDR + 0x0218	TransferCfg2	Local AHB master transfer Configuration
BASEADDR + 0x021C	FFStatus2	Status Register
BASEADDR + 0x0220	FFISTS_TH2	Interrupt status flags, a '1' signifies that the corresponding interrupt condition occurred (even if interrupt is disabled), write '1' clears the flag. Even if the factor of Interrupt is cancelled (e.g. not empty), Or if an external clear signal is set to 1, this bit will be cleared automatically. When the rising edge and StatusClear of an interrupt factor happen simultaneously, Status Register gives priority to an interrupt factor.
BASEADDR + 0x0224	FFIEN_TH2	Interrupt Enable register. '1' is enable.
BASEADDR + 0x0228	Reserved	Do not modify

Table 3-10: Registers Overview

Base Address(es)	Instance no 0: BASEADDR="0002E000"	
Absolute Address	**Register Name**	**Register Description**
BASEADDR + 0x022C	FFIEN_DW2	Interrupt Enable register. '1' is enable.
BASEADDR + 0x0230	READ_CNT2	32 Bit Counter. Is incremented with every read from the FIFO. This does not correspond with the size of the read (e.g. WORD, HALF WORD or BYTE). For each access regardless of size the counter is incremented by one. An overflow will occur at the boundary of 32 Bit.
BASEADDR + 0x0234	WRITE_CNT2	32 Bit Counter. Is incremented with every write from the FIFO. This does not correspond with the size of the write (e.g. WORD, HALF WORD or BYTE). For each access regardless of size the counter is incremented by one. An overflow will occur at the boundary of 32 Bit.
BASEADDR + 0x0238	INT_READ_ADDR2	Readback register for the read address pointer to the FIFO. NOTE: This is an internal address. It is derived from the lower bound address but translated internally (left shift by three '0'). The value is the next address that will be read from. For an example refer to the chapter 'Readback address pointers'.
BASEADDR + 0x023C	INT_WRITE_ADDR2	Readback register for the current write address pointers to the FIFO. NOTE: These are internal addresses. They are derived from the lower bound address but translated internally (left shift by three '0'). The value is the last address that has been written to. For an example refer to the chapter 'Readback address pointers'.
BASEADDR + 0x0240	FFEDataInL2	Last FIFO Data In Lower
BASEADDR + 0x0244	FFEDataInU2	Last FIFO Data In Upper
BASEADDR + 0x0248	FFDataInL2	FIFO Data In Lower
BASEADDR + 0x024C	FFDataInU2	FIFO Data In Upper
BASEADDR + 0x0250	FFDataSize2	Number of bytes that will be written to the FIFO
BASEADDR + 0x0254	MetaDestAddress2	Local AHB-master transfer Destination address for Meta Command operation. NOTEs: The first and the last command to in a transmission sequence cannot be a meta command! The Meta address register has to be always written prior to the Meta Config register! For transmissions sequences that contain meta command only indefinite length bursts can be configured (for both fields 'MetaTransferINCR' and 'TransferINCR')!
BASEADDR + 0x0258	MetaCfg2	Local AHB master transfer and address generation Configuration for Meta Command operation. NOTEs: The first and the last command to in a transmission sequence cannot be a meta command! The Meta address register has to be always written prior to the Meta Config register! For transmissions sequences that contain meta command only indefinite length bursts can be configured (for both fields 'MetaTransferINCR' and 'TransferINCR')!
BASEADDR + 0x0280	SW_RT3	Software Reset and Trigger
BASEADDR + 0x0284	FFCfg3	The function of FIFO is set up.
BASEADDR + 0x0288	FFB3	This address sets the boundary address of sharing FIFO. Do not overlap that each channels boundary area setting. When the area overlaps, the operation guarantee cannot be done. When using this channel, it sets up so that it may be set to 'UpperBoundAdr >= LowerBoundAdr'. The maximum size for 'UpperBoundAdr - LowerBoundAdr' is 0x1FE.
BASEADDR + 0x028C	FFT3	Sets the threshhold level for the corresponding FIFO in Bytes.
BASEADDR + 0x0290	DestAddress3	Local AHB-master transfer Destination address
BASEADDR + 0x0294	AdrCfg3	Address generation Configuration

Table 3-10: Registers Overview

Base Address(es)	Instance no 0: BASEADDR="0002E000"	
Absolute Address	**Register Name**	**Register Description**
BASEADDR + 0x0298	TransferCfg3	Local AHB master transfer Configuration
BASEADDR + 0x029C	FFStatus3	Status Register
BASEADDR + 0x02A0	FFISTS_TH3	Interrupt status flags, a '1' signifies that the corresponding interrupt condition occurred (even if interrupt is disabled), write '1' clears the flag. Even if the factor of Interrupt is cancelled (e.g. not empty), Or if an external clear signal is set to 1, this bit will be cleared automatically. When the rising edge and StatusClear of an interrupt factor happen simultaneously, Status Register gives priority to an interrupt factor.
BASEADDR + 0x02A4	FFIEN_TH3	Interrupt Enable register. '1' is enable.
BASEADDR + 0x02A8	Reserved	Do not modify
BASEADDR + 0x02AC	FFIEN_DW3	Interrupt Enable register. '1' is enable.
BASEADDR + 0x02B0	READ_CNT3	32 Bit Counter. Is incremented with every read from the FIFO. This does not correspond with the size of the read (e.g. WORD, HALF WORD or BYTE). For each access regardless of size the counter is incremented by one. An overflow will occur at the boundary of 32 Bit.
BASEADDR + 0x02B4	WRITE_CNT3	32 Bit Counter. Is incremented with every write from the FIFO. This does not correspond with the size of the write (e.g. WORD, HALF WORD or BYTE). For each access regardless of size the counter is incremented by one. An overflow will occur at the boundary of 32 Bit.
BASEADDR + 0x02B8	INT_READ_ADDR3	Readback register for the read address pointer to the FIFO. NOTE: This is an internal address. It is derived from the lower bound address but translated internally (left shift by three '0'). The value is the next address that will be read from. For an example refer to the chapter 'Readback address pointers'.
BASEADDR + 0x02BC	INT_WRITE_ADDR3	Readback register for the current write address pointers to the FIFO. NOTE: These are internal addresses. They are derived from the lower bound address but translated internally (left shift by three '0'). The value is the last address that has been written to. For an example refer to the chapter 'Readback address pointers'.
BASEADDR + 0x02C0	FFEDataInL3	Last FIFO Data In Lower
BASEADDR + 0x02C4	FFEDataInU3	Last FIFO Data In Upper
BASEADDR + 0x02C8	FFDataInL3	FIFO Data In Lower
BASEADDR + 0x02CC	FFDataInU3	FIFO Data In Upper
BASEADDR + 0x02D0	FFDataSize3	Number of bytes that will be written to the FIFO
BASEADDR + 0x02D4	MetaDestAddress3	Local AHB-master transfer Destination address for Meta Command operation. NOTEs: The first and the last command to in a transmission sequence cannot be a meta command! The Meta address register has to be always written prior to the Meta Config register! For transmissions sequences that contain meta command only indefinite length bursts can be configured (for both fields 'MetaTransferINCR' and 'TransferINCR')!
BASEADDR + 0x02D8	MetaCfg3	Local AHB master transfer and address generation Configuration for Meta Command operation. NOTEs: The first and the last command to in a transmission sequence cannot be a meta command! The Meta address register has to be always written prior to the Meta Config register! For transmissions sequences that contain meta command only indefinite length bursts can be configured (for both fields 'MetaTransferINCR' and 'TransferINCR')!
BASEADDR + 0x0300	SW_RT4	Software Reset and Trigger

Table 3-10: Registers Overview

Base Address(es)	Instance no 0: BASEADDR="0002E000"	
Absolute Address	**Register Name**	**Register Description**
BASEADDR + 0x0304	FFCfg4	The function of FIFO is set up.
BASEADDR + 0x0308	FFB4	This address sets the boundary address of sharing FIFO. Do not overlap that each channels boundary area setting. When the area overlaps, the operation guarantee cannot be done. When using this channel, it sets up so that it may be set to 'UpperBoundAdr >= LowerBoundAdr'. The maximum size for 'UpperBoundAdr - LowerBoundAdr' is 0x1FE.
BASEADDR + 0x030C	FFT4	Sets the threshhold level for the corresponding FIFO in Bytes.
BASEADDR + 0x0310	DestAddress4	Local AHB-master transfer Destination address
BASEADDR + 0x0314	AdrCfg4	Address generation Configuration
BASEADDR + 0x0318	TransferCfg4	Local AHB master transfer Configuration
BASEADDR + 0x031C	FFStatus4	Status Register
BASEADDR + 0x0320	FFISTS_TH4	Interrupt status flags, a '1' signifies that the corresponding interrupt condition occurred (even if interrupt is disabled), write '1' clears the flag. Even if the factor of Interrupt is cancelled (e.g. not empty), Or if an external clear signal is set to 1, this bit will be cleared automatically. When the rising edge and StatusClear of an interrupt factor happen simultaneously, Status Register gives priority to an interrupt factor.
BASEADDR + 0x0324	FFIEN_TH4	Interrupt Enable register. '1' is enable.
BASEADDR + 0x0328	Reserved	Do not modify
BASEADDR + 0x032C	FFIEN_DW4	Interrupt Enable register. '1' is enable.
BASEADDR + 0x0330	READ_CNT4	32 Bit Counter. Is incremented with every read from the FIFO. This does not correspond with the size of the read (e.g. WORD, HALF WORD or BYTE). For each access regardless of size the counter is incremented by one. An overflow will occur at the boundary of 32 Bit.
BASEADDR + 0x0334	WRITE_CNT4	32 Bit Counter. Is incremented with every write from the FIFO. This does not correspond with the size of the write (e.g. WORD, HALF WORD or BYTE). For each access regardless of size the counter is incremented by one. An overflow will occur at the boundary of 32 Bit.
BASEADDR + 0x0338	INT_READ_ADDR4	Readback register for the read address pointer to the FIFO. NOTE: This is an internal address. It is derived from the lower bound address but translated internally (left shift by three '0'). The value is the next address that will be read from. For an example refer to the chapter 'Readback address pointers'.
BASEADDR + 0x033C	INT_WRITE_ADDR4	Readback register for the current write address pointers to the FIFO. NOTE: These are internal addresses. They are derived from the lower bound address but translated internally (left shift by three '0'). The value is the last address that has been written to. For an example refer to the chapter 'Readback address pointers'.
BASEADDR + 0x0340	FFEDataInL4	Last FIFO Data In Lower
BASEADDR + 0x0344	FFEDataInU4	Last FIFO Data In Upper
BASEADDR + 0x0348	FFDataInL4	FIFO Data In Lower
BASEADDR + 0x034C	FFDataInU4	FIFO Data In Upper
BASEADDR + 0x0350	FFDataSize4	Number of bytes that will be written to the FIFO

Table 3-10: Registers Overview

Base Address(es)	Instance no 0: BASEADDR="0002E000"	
Absolute Address	**Register Name**	**Register Description**
BASEADDR + 0x0354	MetaDestAddress4	Local AHB-master transfer Destination address for Meta Command operation. NOTEs: The first and the last command to in a transmission sequence cannot be a meta command! The Meta address register has to be always written prior to the Meta Config register! For transmissions sequences that contain meta command only indefinite length bursts can be configured (for both fields 'MetaTransferINCR' and 'TransferINCR')!
BASEADDR + 0x0358	MetaCfg4	Local AHB master transfer and address generation Configuration for Meta Command operation. NOTEs: The first and the last command to in a transmission sequence cannot be a meta command! The Meta address register has to be always written prior to the Meta Config register! For transmissions sequences that contain meta command only indefinite length bursts can be configured (for both fields 'MetaTransferINCR' and 'TransferINCR')!
BASEADDR + 0x0380	SW_RT5	Software Reset and Trigger
BASEADDR + 0x0384	FFCfg5	The function of FIFO is set up.
BASEADDR + 0x0388	FFB5	This address sets the boundary address of sharing FIFO. Do not overlap that each channels boundary area setting. When the area overlaps, the operation guarantee cannot be done. When using this channel, it sets up so that it may be set to 'UpperBoundAdr >= LowerBoundAdr'. The maximum size for 'UpperBoundAdr - LowerBoundAdr' is 0x1FE.
BASEADDR + 0x038C	FFT5	Sets the threshhold level for the corresponding FIFO in Bytes.
BASEADDR + 0x0390	DestAddress5	Local AHB-master transfer Destination address
BASEADDR + 0x0394	AdrCfg5	Address generation Configuration
BASEADDR + 0x0398	TransferCfg5	Local AHB master transfer Configuration
BASEADDR + 0x039C	FFStatus5	Status Register
BASEADDR + 0x03A0	FFISTS_TH5	Interrupt status flags, a '1' signifies that the corresponding interrupt condition occurred (even if interrupt is disabled), write '1' clears the flag. Even if the factor of Interrupt is cancelled (e.g. not empty), Or if an external clear signal is set to 1, this bit will be cleared automatically. When the rising edge and StatusClear of an interrupt factor happen simultaneously, Status Register gives priority to an interrupt factor.
BASEADDR + 0x03A4	FFIEN_TH5	Interrupt Enable register. '1' is enable.
BASEADDR + 0x03A8	Reserved	Do not modify
BASEADDR + 0x03AC	FFIEN_DW5	Interrupt Enable register. '1' is enable.
BASEADDR + 0x03B0	READ_CNT5	32 Bit Counter. Is incremented with every read from the FIFO. This does not correspond with the size of the read (e.g. WORD, HALF WORD or BYTE). For each access regardless of size the counter is incremented by one. An overflow will occur at the boundary of 32 Bit.
BASEADDR + 0x03B4	WRITE_CNT5	32 Bit Counter. Is incremented with every write from the FIFO. This does not correspond with the size of the write (e.g. WORD, HALF WORD or BYTE). For each access regardless of size the counter is incremented by one. An overflow will occur at the boundary of 32 Bit.
BASEADDR + 0x03B8	INT_READ_ADDR5	Readback register for the read address pointer to the FIFO. NOTE: This is an internal address. It is derived from the lower bound address but translated internally (left shift by three '0'). The value is the next address that will be read from. For an example refer to the chapter 'Readback address pointers'.

Table 3-10: Registers Overview

Base Address(es)	Instance no 0: BASEADDR="0002E000"	
Absolute Address	**Register Name**	**Register Description**
BASEADDR + 0x03BC	INT_WRITE_ADDR5	Readback register for the current write address pointers to the FIFO. NOTE: These are internal addresses. They are derived from the lower bound address but translated internally (left shift by three '0'). The value is the last address that has been written to. For an example refer to the chapter 'Readback address pointers'.
BASEADDR + 0x03C0	FFEDataInL5	Last FIFO Data In Lower
BASEADDR + 0x03C4	FFEDataInU5	Last FIFO Data In Upper
BASEADDR + 0x03C8	FFDataInL5	FIFO Data In Lower
BASEADDR + 0x03CC	FFDataInU5	FIFO Data In Upper
BASEADDR + 0x03D0	FFDataSize5	Number of bytes that will be written to the FIFO
BASEADDR + 0x03D4	MetaDestAddress5	Local AHB-master transfer Destination address for Meta Command operation. NOTEs: The first and the last command to in a transmission sequence cannot be a meta command! The Meta address register has to be always written prior to the Meta Config register! For transmissions sequences that contain meta command only indefinite length bursts can be configured (for both fields 'MetaTransferINCR' and 'TransferINCR')!
BASEADDR + 0x03D8	MetaCfg5	Local AHB master transfer and address generation Configuration for Meta Command operation. NOTEs: The first and the last command to in a transmission sequence cannot be a meta command! The Meta address register has to be always written prior to the Meta Config register! For transmissions sequences that contain meta command only indefinite length bursts can be configured (for both fields 'MetaTransferINCR' and 'TransferINCR')!
BASEADDR + 0x0400	SW_RT6	Software Reset and Trigger
BASEADDR + 0x0404	FFCfg6	The function of FIFO is set up.
BASEADDR + 0x0408	FFB6	This address sets the boundary address of sharing FIFO. Do not overlap that each channels boundary area setting. When the area overlaps, the operation guarantee cannot be done. When using this channel, it sets up so that it may be set to 'UpperBoundAdr >= LowerBoundAdr'. The maximum size for 'UpperBoundAdr - LowerBoundAdr' is 0x1FE.
BASEADDR + 0x040C	FFT6	Sets the threshhold level for the corresponding FIFO in Bytes.
BASEADDR + 0x0410	DestAddress6	Local AHB-master transfer Destination address
BASEADDR + 0x0414	AdrCfg6	Address generation Configuration
BASEADDR + 0x0418	TransferCfg6	Local AHB master transfer Configuration
BASEADDR + 0x041C	FFStatus6	Status Register
BASEADDR + 0x0420	FFISTS_TH6	Interrupt status flags, a '1' signifies that the corresponding interrupt condition occurred (even if interrupt is disabled), write '1' clears the flag. Even if the factor of Interrupt is cancelled (e.g. not empty), Or if an external clear signal is set to 1, this bit will be cleared automatically. When the rising edge and StatusClear of an interrupt factor happen simultaneously, Status Register gives priority to an interrupt factor.
BASEADDR + 0x0424	FFIEN_TH6	Interrupt Enable register. '1' is enable.
BASEADDR + 0x0428	Reserved	Do not modify
BASEADDR + 0x042C	FFIEN_DW6	Interrupt Enable register. '1' is enable.
BASEADDR + 0x0430	READ_CNT6	32 Bit Counter. Is incremented with every read from the FIFO. This does not correspond with the size of the read (e.g. WORD, HALF WORD or BYTE). For each access regardless of size the counter is incremented by one. An overflow will occur at the boundary of 32 Bit.

Table 3-10: Registers Overview

Base Address(es)	Instance no 0: BASEADDR="0002E000"	
Absolute Address	**Register Name**	**Register Description**
BASEADDR + 0x0434	WRITE_CNT6	32 Bit Counter. Is incremented with every write from the FIFO. This does not correspond with the size of the write (e.g. WORD, HALF WORD or BYTE). For each access regardless of size the counter is incremented by one. An overflow will occur at the boundary of 32 Bit.
BASEADDR + 0x0438	INT_READ_ADDR6	Readback register for the read address pointer to the FIFO. NOTE: This is an internal address. It is derived from the lower bound address but translated internally (left shift by three '0'). The value is the next address that will be read from. For an example refer to the chapter 'Readback address pointers'.
BASEADDR + 0x043C	INT_WRITE_ADDR6	Readback register for the current write address pointers to the FIFO. NOTE: These are internal addresses. They are derived from the lower bound address but translated internally (left shift by three '0'). The value is the last address that has been written to. For an example refer to the chapter 'Readback address pointers'.
BASEADDR + 0x0440	FFEDataInL6	Last FIFO Data In Lower
BASEADDR + 0x0444	FFEDataInU6	Last FIFO Data In Upper
BASEADDR + 0x0448	FFDataInL6	FIFO Data In Lower
BASEADDR + 0x044C	FFDataInU6	FIFO Data In Upper
BASEADDR + 0x0450	FFDataSize6	Number of bytes that will be written to the FIFO
BASEADDR + 0x0454	MetaDestAddress6	Local AHB-master transfer Destination address for Meta Command operation. NOTEs: The first and the last command to in a transmission sequence cannot be a meta command! The Meta address register has to be always written prior to the Meta Config register! For transmissions sequences that contain meta command only indefinite length bursts can be configured (for both fields 'MetaTransferINCR' and 'TransferINCR')!
BASEADDR + 0x0458	MetaCfg6	Local AHB master transfer and address generation Configuration for Meta Command operation. NOTEs: The first and the last command to in a transmission sequence cannot be a meta command! The Meta address register has to be always written prior to the Meta Config register! For transmissions sequences that contain meta command only indefinite length bursts can be configured (for both fields 'MetaTransferINCR' and 'TransferINCR')!
BASEADDR + 0x0480	SW_RT7	Software Reset and Trigger
BASEADDR + 0x0484	FFCfg7	The function of FIFO is set up.
BASEADDR + 0x0488	FFB7	This address sets the boundary address of sharing FIFO. Do not overlap that each channels boundary area setting. When the area overlaps, the operation guarantee cannot be done. When using this channel, it sets up so that it may be set to 'UpperBoundAdr >= LowerBoundAdr'. The maximum size for 'UpperBoundAdr - LowerBoundAdr' is 0x1FE.
BASEADDR + 0x048C	FFT7	Sets the threshhold level for the corresponding FIFO in Bytes.
BASEADDR + 0x0490	DestAddress7	Local AHB-master transfer Destination address
BASEADDR + 0x0494	AdrCfg7	Address generation Configuration
BASEADDR + 0x0498	TransferCfg7	Local AHB master transfer Configuration
BASEADDR + 0x049C	FFStatus7	Status Register

Table 3-10: Registers Overview

Base Address(es)	Instance no 0: BASEADDR="0002E000"	
Absolute Address	**Register Name**	**Register Description**
BASEADDR + 0x04A0	FFISTS_TH7	Interrupt status flags, a '1' signifies that the corresponding interrupt condition occurred (even if interrupt is disabled), write '1' clears the flag. Even if the factor of Interrupt is cancelled (e.g. not empty), Or if an external clear signal is set to 1, this bit will be cleared automatically. When the rising edge and StatusClear of an interrupt factor happen simultaneously, Status Register gives priority to an interrupt factor.
BASEADDR + 0x04A4	FFIEN_TH7	Interrupt Enable register. '1' is enable.
BASEADDR + 0x04A8	Reserved	Do not modify
BASEADDR + 0x04AC	FFIEN_DW7	Interrupt Enable register. '1' is enable.
BASEADDR + 0x04B0	READ_CNT7	32 Bit Counter. Is incremented with every read from the FIFO. This does not correspond with the size of the read (e.g. WORD, HALF WORD or BYTE). For each access regardless of size the counter is incremented by one. An overflow will occur at the boundary of 32 Bit.
BASEADDR + 0x04B4	WRITE_CNT7	32 Bit Counter. Is incremented with every write from the FIFO. This does not correspond with the size of the write (e.g. WORD, HALF WORD or BYTE). For each access regardless of size the counter is incremented by one. An overflow will occur at the boundary of 32 Bit.
BASEADDR + 0x04B8	INT_READ_ADDR7	Readback register for the read address pointer to the FIFO. NOTE: This is an internal address. It is derived from the lower bound address but translated internally (left shift by three '0'). The value is the next address that will be read from. For an example refer to the chapter 'Readback address pointers'.
BASEADDR + 0x04BC	INT_WRITE_ADDR7	Readback register for the current write address pointers to the FIFO. NOTE: These are internal addresses. They are derived from the lower bound address but translated internally (left shift by three '0'). The value is the last address that has been written to. For an example refer to the chapter 'Readback address pointers'.
BASEADDR + 0x04C0	FFEDataInL7	Last FIFO Data In Lower
BASEADDR + 0x04C4	FFEDataInU7	Last FIFO Data In Upper
BASEADDR + 0x04C8	FFDataInL7	FIFO Data In Lower
BASEADDR + 0x04CC	FFDataInU7	FIFO Data In Upper
BASEADDR + 0x04D0	FFDataSize7	Number of bytes that will be written to the FIFO
BASEADDR + 0x04D4	MetaDestAddress7	Local AHB-master transfer Destination address for Meta Command operation. NOTEs: The first and the last command to in a transmission sequence cannot be a meta command! The Meta address register has to be always written prior to the Meta Config register! For transmissions sequences that contain meta command only indefinite length bursts can be configured (for both fields 'MetaTransferINCR' and 'TransferINCR')!
BASEADDR + 0x04D8	MetaCfg7	Local AHB master transfer and address generation Configuration for Meta Command operation. NOTEs: The first and the last command to in a transmission sequence cannot be a meta command! The Meta address register has to be always written prior to the Meta Config register! For transmissions sequences that contain meta command only indefinite length bursts can be configured (for both fields 'MetaTransferINCR' and 'TransferINCR')!

3.10 DMA Controller

The DMA Controller (DMAC) implements Direct Memory Access (DMA) with little host CPU intervention. The DMAC performs complex data transfers through 2 DMA channels.

3.10.1 Features of the DMA Controller

- Data can be transferred independently over multiple channels.
- A high number of DMA clients can be assigned to the available DMA channels.
- Flexible priority between DMA channels (fixed, dynamic or round-robin)
- DMA transfer request sources
  - Hardware requests
  - Software requests (register write)
- Transfer modes
  - Block transfer, burst transfer and demand transfer
  - Addressing: full 32-bit (incrementing, decrementing or fixed)
  - Data types: 8, 16, 32 or 64-bit wide data

3.10.2 Block Diagram of DMA Controller

Figure 3-27 shows the top level block diagram of the DMA Controller.

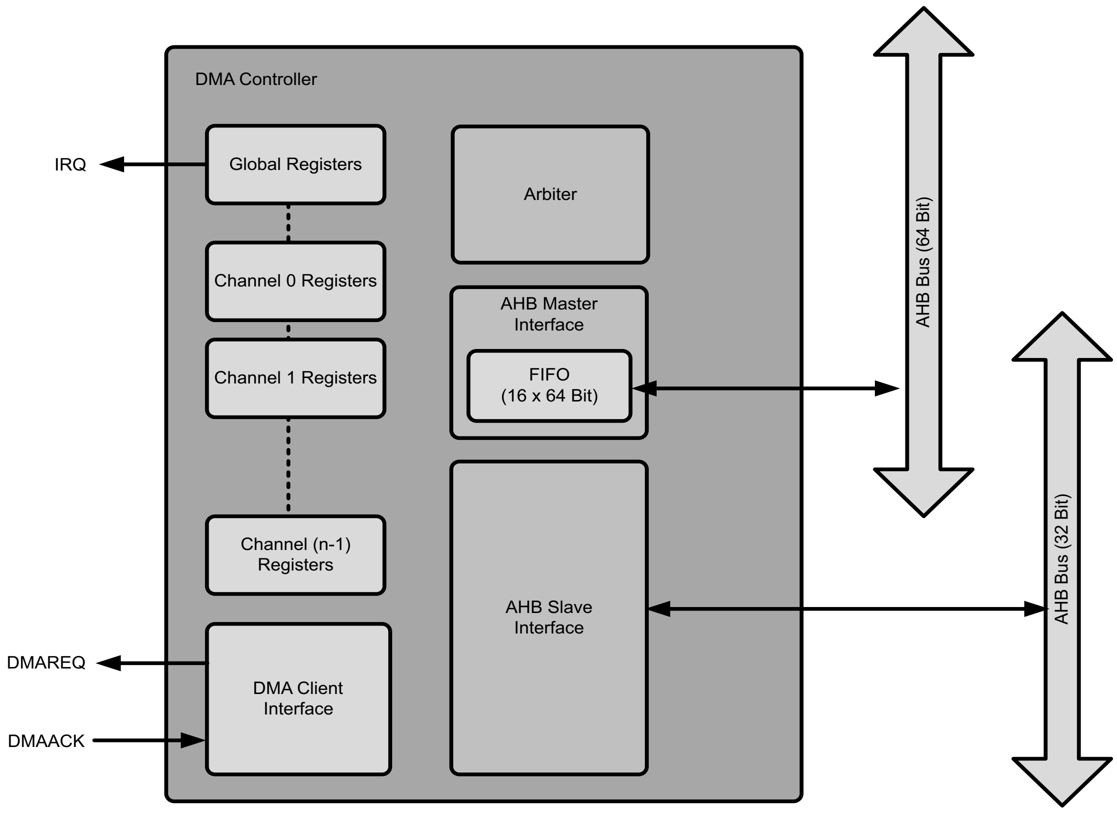


Figure 3-27: DMA Controller Block Diagram

3.10.3 Operation of DMA Controller

The following section describes the operation of the DMA Controller.

3.10.3.1 Features of DMA Controller

- DMA client matrix, routing ’M’ DMA clients to ’N’ DMA Channels.
- DMA trigger
  - Hardware request
  - Software request
- 3 Transfer modes.
  - Block transfer
  - Burst transfer
  - Demand transfer
- 8, 16, 32 or 64 bit wide data transfers.
- Writing to the configuration registers can be done as 8, 16 or 32-bit wide accesses. Illegal accesses result in an error response.
- Incrementing, decrementing or fixed addressing are independent for source and destination.
- Central interrupt flag register for completion and error interrupts.
- Stop status per channel for analysis by ISRs (Interrupt Serve Routines) or debugging.
- Shadow registers for source and destination address.
- 3 channel arbitration schemes:
  - Fixed priority
  - Dynamic priority
  - Round-Robin

3.10.3.2 Global Functions of the DMA Controller

DMAC Enabling or Halting

After a reset, the DMA Controller is disabled. The DMAC is enabled by setting DMA Enable (DMAi_R:DE) to 1. When DMA Enable is set to 1, the individual DMA Channel enable settings (DMAi_An:EB) become effective. Setting DMA Enable to 0 disables the complete DMAC. Setting DMA Enable to 0 while an incomplete transfer is still running on a DMA Channel, will cause the running transfer to be stopped after the current block of data has been transferred and an error interrupt is then issued. DMA channels which have a DMA transfer pending, but are not currently served are disabled and an error interrupt is issued. DMA channels which are enabled but have not started a DMA transfer (the channel was enabled but the transfer request has not yet occurred) are simply disabled without issuing an error interrupt.

The DMA Controller can be halted completely (all DMA Channels) by writing a single bit, DMA Halt (DMAi_R:DH). When this bit is set to 1, all DMA Channels are requested to halt and not to perform DMA transfers until this bit has been cleared. After DMA Halt has been cleared, the halted DMA transfers continue from the point at which they were halted. If DMA Halt is set to 1 while a transfer is running, the halt occurs when the current block of data of the running transfer has been transferred.

NOTE Depending on the clock ratio of the DMAC/Source clock domain and DMAC/Destination clock domain, a considerable space of time can elapse between the time when the halt request was set and when the DMAC is actually halted.

The global disable and halt request conditions of the DMAC are indicated by DMA Stop Flag (DMAi_R:DS). DMA Stop Flag is 0 if none of the disable or halt request conditions below is true.

- DMAi_R:DE bit is set to 0
- DMAi_R:DH bit is set to 1

If any of the above conditions is true, DMA Stop Flag is 1 indicating that DMA transfers of all channels are requested to halt or stop.

3.10.4 DMA Channels

3.10.4.1 Modes of Operation

The DMA Channels can operate in 3 modes:

- Block transfer mode
- Burst transfer mode
- Demand transfer mode

The mode must be set with bits Mode Select (DMAi_Bn:MS[1:0]). After reset, the channel is set to Block transfer mode.

3.10.4.2 Block Transfer Mode

In block transfer mode, the DMA Client will request the transfers of a specified number of blocks of data. The number of blocks to be transferred is specified with *Transfer Count* (DMAi_An:TC[15:0]). Each block of data is transferred in one arbitration phase of the DMA Arbiter. Either a hardware request or software request is needed for each block to transfer via DMA transfer. The DMA Client or the software continues to give requests until the DMAC has transferred the specified number of blocks and thus, the DMA transfer is completed. The DMA transfer will be successfully completed if all the data blocks were transferred without errors, or it will be unsuccessfully completed if an error condition occurred.

After each transferred block of data, the DMA Arbiter does another arbitration and proceeds with the next requesting channel with the highest priority. The arbitration depends on the selected arbitration scheme and the set priorities of the requesting channels (see 3.10.6 for the details about the arbitration).

3.10.4.2.1 DMA Transfer Requests

Hardware Request

For a channel hardware request, Input Select (DMAi_An:IS) need to be set to ‘01’. The DMA client which is routed through the DMA Client Matrix to the channel will give the trigger by asserting DREQ. Refer to 3.10.5 for the function and configuration of the DMA Client Matrix.

NOTE Although the DMA request is triggered by a hardware client, a software IRQ clear is required after a transfer.

Software Request

For a software request, Input Select need to be set to ‘00’ and Software Trigger (DMAi_An:ST) must be set to 1. Software Trigger shall be set to 1 if the channel is ready to receive a software trigger, which is indicated with Software Trigger Ready flag (DMAi_Bn:SR), and no error condition is pending (DMAi_Bn:SS[2:0] is ‘000’ or ‘101’). If Software Trigger is tried to set to 1 while Software Trigger Ready is 0, it is ignored. Software Trigger is automatically cleared by hardware once the trigger is accepted or an error condition occurred. The error conditions can be:

- DMA Channel is disabled immediately after setting Software Trigger
- The master interface receives an error response and the CPU sets the Software Trigger before receiving an error interrupt caused by the AHB error

NOTE After changing the source and destination address (DMAi_A0.EB low, write new addresses, DMAi_A0.EB high), please wait 5 AHB-cycles before the transfer can be triggered by software, otherwise the software trigger just vanishes without any effect.

3.10.4.2.2 Block of Data

A block of data is determined by the setting of Block Count (DMAi_An:BC[3:0]) and Transfer Width (DMAi_Bn:TW[1:0]). The DMA Controller will make BC (Block Count) + 1 data transfers from the source address range starting at Source Address (DMAi_SAn:SA) to the destination address range starting at Destination Address (DMAi_DAn:DA). If BC is set to 0, a single data transfer from Source Address (DMAi_SAn:SA) to Destination Address (DMAi_DAn:DA) will be done. The settings of Block Count, Beat Limit (DMAi_An:BL[1:0], Alternate (DMAi_An:AL), and Transfer Width define how the AHB Master Interface issues the BC + 1 data transfers. The following table shows all possible combinations between BC, BL, and AL. Only these three influence the sequence of AHB transfers whereas TW only affects the data size which will be transported.

NOTE When DMA operation is done during a running Indigo2 display the maximum burst length should be limited to SINGLE to avoid underruns in the display buffers.

Table 3-11: Block Count / Beat Limit / Alternate combinations

Block Count	Beat Limit	Alternate	Resulting sequence of AHB transfers
0	SINGLE	0	1x SINGLE RD + 1x SINGLE WR
1	SINGLE	0	2x SINGLE RD + 2x SINGLE WR
2	SINGLE	0	3x SINGLE RD + 3x SINGLE WR
3	SINGLE	0	4x SINGLE RD + 4x SINGLE WR
4	SINGLE	0	5x SINGLE RD + 5x SINGLE WR
5	SINGLE	0	6x SINGLE RD + 6x SINGLE WR
6	SINGLE	0	7x SINGLE RD + 7x SINGLE WR
7	SINGLE	0	8x SINGLE RD + 8x SINGLE WR
8	SINGLE	0	9x SINGLE RD + 9x SINGLE WR
9	SINGLE	0	10x SINGLE RD + 10x SINGLE WR
10	SINGLE	0	11x SINGLE RD + 11x SINGLE WR
11	SINGLE	0	12x SINGLE RD + 12x SINGLE WR
12	SINGLE	0	13x SINGLE RD + 13x SINGLE WR
13	SINGLE	0	14x SINGLE RD + 14x SINGLE WR
14	SINGLE	0	15x SINGLE RD + 15x SINGLE WR
15	SINGLE	0	16x SINGLE RD + 16x SINGLE WR
0	SINGLE	1	1x (1x SINGLE RD + 1x SINGLE WR)
1	SINGLE	1	2x (1x SINGLE RD + 1x SINGLE WR)
2	SINGLE	1	3x (1x SINGLE RD + 1x SINGLE WR)
3	SINGLE	1	4x (1x SINGLE RD + 1x SINGLE WR)

Table 3-11: Block Count / Beat Limit / Alternate combinations (Continued)

Block Count	Beat Limit	Alternate	Resulting sequence of AHB transfers
4	SINGLE	1	5x (1x SINGLE RD + 1x SINGLE WR)
5	SINGLE	1	6x (1x SINGLE RD + 1x SINGLE WR)
6	SINGLE	1	7x (1x SINGLE RD + 1x SINGLE WR)
7	SINGLE	1	8x (1x SINGLE RD + 1x SINGLE WR)
8	SINGLE	1	9x (1x SINGLE RD + 1x SINGLE WR)
9	SINGLE	1	10x (1x SINGLE RD + 1x SINGLE WR)
10	SINGLE	1	11x (1x SINGLE RD + 1x SINGLE WR)
11	SINGLE	1	12x (1x SINGLE RD + 1x SINGLE WR)
12	SINGLE	1	13x (1x SINGLE RD + 1x SINGLE WR)
13	SINGLE	1	14x (1x SINGLE RD + 1x SINGLE WR)
14	SINGLE	1	15x (1x SINGLE RD + 1x SINGLE WR)
15	SINGLE	1	16x (1x SINGLE RD + 1x SINGLE WR)
0	INCR4	0	1x SINGLE RD + 1x SINGLE WR
1	INCR4	0	2x SINGLE RD + 2x SINGLE WR
2	INCR4	0	3x SINGLE RD + 3x SINGLE WR
3	INCR4	0	1x 4_BEAT RD + 1x 4_BEAT WR
4	INCR4	0	1x 4_BEAT RD + 1x SINGLE RD + 1x 4_BEAT WR + 1x SINGLE WR
5	INCR4	0	1x 4_BEAT RD + 2x SINGLE RD + 1x 4_BEAT WR + 2x SINGLE WR
6	INCR4	0	1x 4_BEAT RD + 3x SINGLE RD + 1x 4_BEAT WR + 3x SINGLE WR
7	INCR4	0	2x 4_BEAT RD + 2x 4_BEAT WR
8	INCR4	0	2x 4_BEAT RD + 1x SINGLE RD + 2x 4_BEAT WR + 1x SINGLE WR
9	INCR4	0	2x 4_BEAT RD + 2x SINGLE RD + 2x 4_BEAT WR + 2x SINGLE WR
10	INCR4	0	2x 4_BEAT RD + 3x SINGLE RD + 2x 4_BEAT WR + 3x SINGLE WR
11	INCR4	0	3x 4_BEAT RD + 3x 4_BEAT WR
12	INCR4	0	3x 4_BEAT RD + 1x SINGLE RD + 3x 4_BEAT WR + 1x SINGLE WR
13	INCR4	0	3x 4_BEAT RD + 2x SINGLE RD + 3x 4_BEAT WR + 2x SINGLE WR
14	INCR4	0	3x 4_BEAT RD + 3x SINGLE RD + 3x 4_BEAT WR + 3x SINGLE WR
15	INCR4	0	4x 4_BEAT RD + 4x 4_BEAT WR
0	INCR4	1	1x SINGLE RD + 1x SINGLE WR
1	INCR4	1	2x (1x SINGLE RD + 1x SINGLE WR)
2	INCR4	1	3x (1x SINGLE RD + 1x SINGLE WR)
3	INCR4	1	1x (1x 4_BEAT RD + 1x 4_BEAT WR)
4	INCR4	1	1x (1x 4_BEAT RD + 1x 4_BEAT WR) + 1x SINGLE RD + 1x SINGLE WR
5	INCR4	1	1x (1x 4_BEAT RD + 1x 4_BEAT WR) + 2x (1x SINGLE RD + 1x SINGLE WR)
6	INCR4	1	1x (1x 4_BEAT RD + 1x 4_BEAT WR) + 3x (1x SINGLE RD + 1x SINGLE WR)
7	INCR4	1	2x (1x 4_BEAT RD + 1x 4_BEAT WR)
8	INCR4	1	2x (1x 4_BEAT RD + 1x 4_BEAT WR) + 1x SINGLE RD + 1x SINGLE WR
9	INCR4	1	2x (1x 4_BEAT RD + 1x 4_BEAT WR) + 2x (1x SINGLE RD + 1x SINGLE WR)
10	INCR4	1	2x (1x 4_BEAT RD + 1x 4_BEAT WR) + 3x (1x SINGLE RD + 1x SINGLE WR)
11	INCR4	1	3x (1x 4_BEAT RD + 1x 4_BEAT WR)
12	INCR4	1	3x (1x 4_BEAT RD + 1x 4_BEAT WR) + 1x SINGLE RD + 1x SINGLE WR
13	INCR4	1	3x (1x 4_BEAT RD + 1x 4_BEAT WR) + 2x (1x SINGLE RD + 1x SINGLE WR)
14	INCR4	1	3x (1x 4_BEAT RD + 1x 4_BEAT WR) + 3x (1x SINGLE RD + 1x SINGLE WR)
15	INCR4	1	4x (1x 4_BEAT RD + 1x 4_BEAT WR)
0	INCR8	0	1x SINGLE RD + 1x SINGLE WR
1	INCR8	0	2x SINGLE RD + 2x SINGLE WR
2	INCR8	0	3x SINGLE RD + 3x SINGLE WR
3	INCR8	0	1x 4_BEAT RD + 1x 4_BEAT WR
4	INCR8	0	1x 4_BEAT RD + 1x SINGLE RD + 1x 4_BEAT WR + 1x SINGLE WR
5	INCR8	0	1x 4_BEAT RD + 2x SINGLE RD + 1x 4_BEAT WR + 2x SINGLE WR
6	INCR8	0	1x 4_BEAT RD + 3x SINGLE RD + 1x 4_BEAT WR + 3x SINGLE WR

Table 3-11: Block Count / Beat Limit / Alternate combinations (Continued)

Block Count	Beat Limit	Alternate	Resulting sequence of AHB transfers
7	INCR8	0	1x 8_BEAT RD + 1x 8_BEAT WR
8	INCR8	0	1x 8_BEAT RD + 1x SINGLE RD + 1x 8_BEAT WR + 1x SINGLE WR
9	INCR8	0	1x 8_BEAT RD + 2x SINGLE RD + 1x 8_BEAT WR + 2x SINGLE WR
10	INCR8	0	1x 8_BEAT RD + 3x SINGLE RD + 1x 8_BEAT WR + 3x SINGLE WR
11	INCR8	0	1x 8_BEAT RD + 1x 4_BEAT RD + 1x 8_BEAT WR + 1x 4_BEAT WR
12	INCR8	0	1x 8_BEAT RD + 1x 4_BEAT RD + 1x SINGLE RD + 1x 8_BEAT WR + 1x 4_BEAT WR + 1x SINGLE WR
13	INCR8	0	1x 8_BEAT RD + 1x 4_BEAT RD + 2x SINGLE RD + 1x 8_BEAT WR + 1x 4_BEAT WR + 2x SINGLE WR
14	INCR8	0	1x 8_BEAT RD + 1x 4_BEAT RD + 3x SINGLE RD + 1x 8_BEAT WR + 1x 4_BEAT WR + 3x SINGLE WR
15	INCR8	0	2x 8_BEAT RD + 2x 8_BEAT WR
0	INCR8	1	1x SINGLE RD + 1x SINGLE WR
1	INCR8	1	2x (1x SINGLE RD + 1x SINGLE WR)
2	INCR8	1	3x (1x SINGLE RD + 1x SINGLE WR)
3	INCR8	1	1x (1x 4_BEAT RD + 1x 4_BEAT WR)
4	INCR8	1	1x (1x 4_BEAT RD + 1x 4_BEAT WR) + 1x SINGLE RD + 1x SINGLE WR
5	INCR8	1	1x (1x 4_BEAT RD + 1x 4_BEAT WR) + 2x (1x SINGLE RD + 1x SINGLE WR)
6	INCR8	1	1x (1x 4_BEAT RD + 1x 4_BEAT WR) + 3x (1x SINGLE RD + 1x SINGLE WR)
7	INCR8	1	1x (1x 8_BEAT RD + 1x 8_BEAT WR)
8	INCR8	1	1x (1x 8_BEAT RD + 1x 8_BEAT WR) + 1x SINGLE RD + 1x SINGLE WR
9	INCR8	1	1x (1x 8_BEAT RD + 1x 8_BEAT WR) + 2x (1x SINGLE RD + 1x SINGLE WR)
10	INCR8	1	1x (1x 8_BEAT RD + 1x 8_BEAT WR) + 3x (1x SINGLE RD + 1x SINGLE WR)
11	INCR8	1	1x (1x 8_BEAT RD + 1x 8_BEAT WR) + 1x (1x 4_BEAT RD + 1x 4_BEAT WR)
12	INCR8	1	1x (1x 8_BEAT RD + 1x 8_BEAT WR) + 1x (1x 4_BEAT RD + 1x 4_BEAT WR) + 1x SINGLE RD + 1x SINGLE WR
13	INCR8	1	1x (1x 8_BEAT RD + 1x 8_BEAT WR) + 1x (1x 4_BEAT RD + 1x 4_BEAT WR) + 2x (1x SINGLE RD + 1x SINGLE WR)
14	INCR8	1	1x (1x 8_BEAT RD + 1x 8_BEAT WR) + 1x (1x 4_BEAT RD + 1x 4_BEAT WR) + 3x (1x SINGLE RD + 1x SINGLE WR)
15	INCR8	1	2x (1x 8_BEAT RD + 1x 8_BEAT WR)
0	INCR16	0	1x SINGLE RD + 1x SINGLE WR
1	INCR16	0	2x SINGLE RD + 2x SINGLE WR
2	INCR16	0	3x SINGLE RD + 3x SINGLE WR
3	INCR16	0	1x 4_BEAT RD + 1x 4_BEAT WR
4	INCR16	0	1x 4_BEAT RD + 1x SINGLE RD + 1x 4_BEAT WR + 1x SINGLE WR
5	INCR16	0	1x 4_BEAT RD + 2x SINGLE RD + 1x 4_BEAT WR + 2x SINGLE WR
6	INCR16	0	1x 4_BEAT RD + 3x SINGLE RD + 1x 4_BEAT WR + 3x SINGLE WR
7	INCR16	0	1x 8_BEAT RD + 1x 8_BEAT WR
8	INCR16	0	1x 8_BEAT RD + 1x SINGLE RD + 1x 8_BEAT WR + 1x SINGLE WR
9	INCR16	0	1x 8_BEAT RD + 2x SINGLE RD + 1x 8_BEAT WR + 2x SINGLE WR
10	INCR16	0	1x 8_BEAT RD + 3x SINGLE RD + 1x 8_BEAT WR + 3x SINGLE WR
11	INCR16	0	1x 8_BEAT RD + 1x 4_BEAT RD + 1x 8_BEAT WR + 1x 4_BEAT WR
12	INCR16	0	1x 8_BEAT RD + 1x 4_BEAT RD + 1x SINGLE RD + 1x 8_BEAT WR + 1x 4_BEAT WR + 1x SINGLE WR
13	INCR16	0	1x 8_BEAT RD + 1x 4_BEAT RD + 2x SINGLE RD + 1x 8_BEAT WR + 1x 4_BEAT WR + 2x SINGLE WR
14	INCR16	0	1x 8_BEAT RD + 1x 4_BEAT RD + 3x SINGLE RD + 1x 8_BEAT WR + 1x 4_BEAT WR + 3x SINGLE WR
15	INCR16	0	1x 16_BEAT RD + 1x 16_BEAT WR
0	INCR16	1	1x SINGLE RD + 1x SINGLE WR
1	INCR16	1	2x (1x SINGLE RD + 1x SINGLE WR)
2	INCR16	1	3x (1x SINGLE RD + 1x SINGLE WR)

Table 3-11: Block Count / Beat Limit / Alternate combinations (Continued)

Block Count	Beat Limit	Alternate	Resulting sequence of AHB transfers
3	INCR16	1	1x (1x 4_BEAT RD + 1x 4_BEAT WR)
4	INCR16	1	1x (1x 4_BEAT RD + 1x 4_BEAT WR) + 1x SINGLE RD + 1x SINGLE WR
5	INCR16	1	1x (1x 4_BEAT RD + 1x 4_BEAT WR) + 2x (1x SINGLE RD + 1x SINGLE WR)
6	INCR16	1	1x (1x 4_BEAT RD + 1x 4_BEAT WR) + 3x (1x SINGLE RD + 1x SINGLE WR)
7	INCR16	1	1x (1x 8_BEAT RD + 1x 8_BEAT WR)
8	INCR16	1	1x (1x 8_BEAT RD + 1x 8_BEAT WR) + 1x SINGLE RD + 1x SINGLE WR
9	INCR16	1	1x (1x 8_BEAT RD + 1x 8_BEAT WR) + 2x (1x SINGLE RD + 1x SINGLE WR)
10	INCR16	1	1x (1x 8_BEAT RD + 1x 8_BEAT WR) + 3x (1x SINGLE RD + 1x SINGLE WR)
11	INCR16	1	1x (1x 8_BEAT RD + 1x 8_BEAT WR) + 1x (1x 4_BEAT RD + 1x 4_BEAT WR)
12	INCR16	1	1x (1x 8_BEAT RD + 1x 8_BEAT WR) + 1x (1x 4_BEAT RD + 1x 4_BEAT WR) + 1x SINGLE RD + 1x SINGLE WR
13	INCR16	1	1x (1x 8_BEAT RD + 1x 8_BEAT WR) + 1x (1x 4_BEAT RD + 1x 4_BEAT WR) + 2x (1x SINGLE RD + 1x SINGLE WR)
14	INCR16	1	1x (1x 8_BEAT RD + 1x 8_BEAT WR) + 1x (1x 4_BEAT RD + 1x 4_BEAT WR) + 3x (1x SINGLE RD + 1x SINGLE WR)
15	INCR16	1	1x 16_BEAT RD + 1x 16_BEAT WR

NOTE n_BEAT RD can be an ’n’ beat incremental burst (INCRn) or it can be ‘n’ times a single (SINGLE) data transfer. n_BEAT RD will be ‘n’ times a SINGLE transfer if one or more of the following conditions is met:

1. Fixed Source Address (DMAi_Dn:FS) is set to 1.
2. Decrement Source Address (DMAi_Dn:DES) is set to 1.
3. The 1kByte AHB address boundary will be crossed by the read block transfer.

n_BEAT WR can be an ‘n’ beat incremental burst (INCRn) or it can be ‘n’ times a single (SINGLE) data transfer. n_BEAT WR will be ‘n’ times a SINGLE transfer if one or more of the following conditions is met:

1. Fixed Destination Address (DMAi_Dn:FD) is set to 1.
2. Decrement Destination Address (DMAi_Dn:DED) is set to 1.
3. The 1kByte AHB address boundary will be crossed by the write block transfer.

After each successful read data transfer, Source Address Shadow (DMAi_SASHDWn:SASHDW) will be either incremented, decremented or remains unaltered. The behavior is determined by the settings of Fixed Source Address, Decrement Source Address, or Fixed Block Source Address. Destination Address Shadow (DMAi_DASHDWn:DASHDW) exhibits the same behavior with respect to the settings of Fixed Destination Address, Decrement Destination Address, or Fixed Block Destination Address and will be updated after each successful write data transfer. The tables below list the possible combinations and the resulting action.

Table 3-12: Source Address Shadow update behavior

FS	DES	FBS	Description of SASHDW update behavior
0	0	0	SASHDW is incremented at each successful read data transfer. Size of address increment depends on Transfer Width.
0	1	0	SASHDW is decremented at each successful read data transfer. Size of address decrement depends on Transfer Width.
0	X	1	SASHDW is incremented at each successful read data transfer. SASHDW is updated with the value stored in DMAi_SAn at the end of a block.
1	X	X	SASHDW remains constant.

Table 3-13: Destination Address Shadow update behavior

FD	DED	FBD	Description of DASHDW update behavior
0	0	0	DASHDW is incremented at each successful write data transfer. Size of address increment depends on Transfer Width.
0	1	0	DASHDW is decremented at each successful write data transfer. Size of address decrement depends on Transfer Width.
0	X	1	DASHDW is incremented at each successful write data transfer. DASHDW is updated with the value stored in DMAi_DAn at the end of a block.
1	X	X	DASHDW remains constant.

Figure 3-28 illustrates this behavior exemplarily for *Source Address Shadow*.

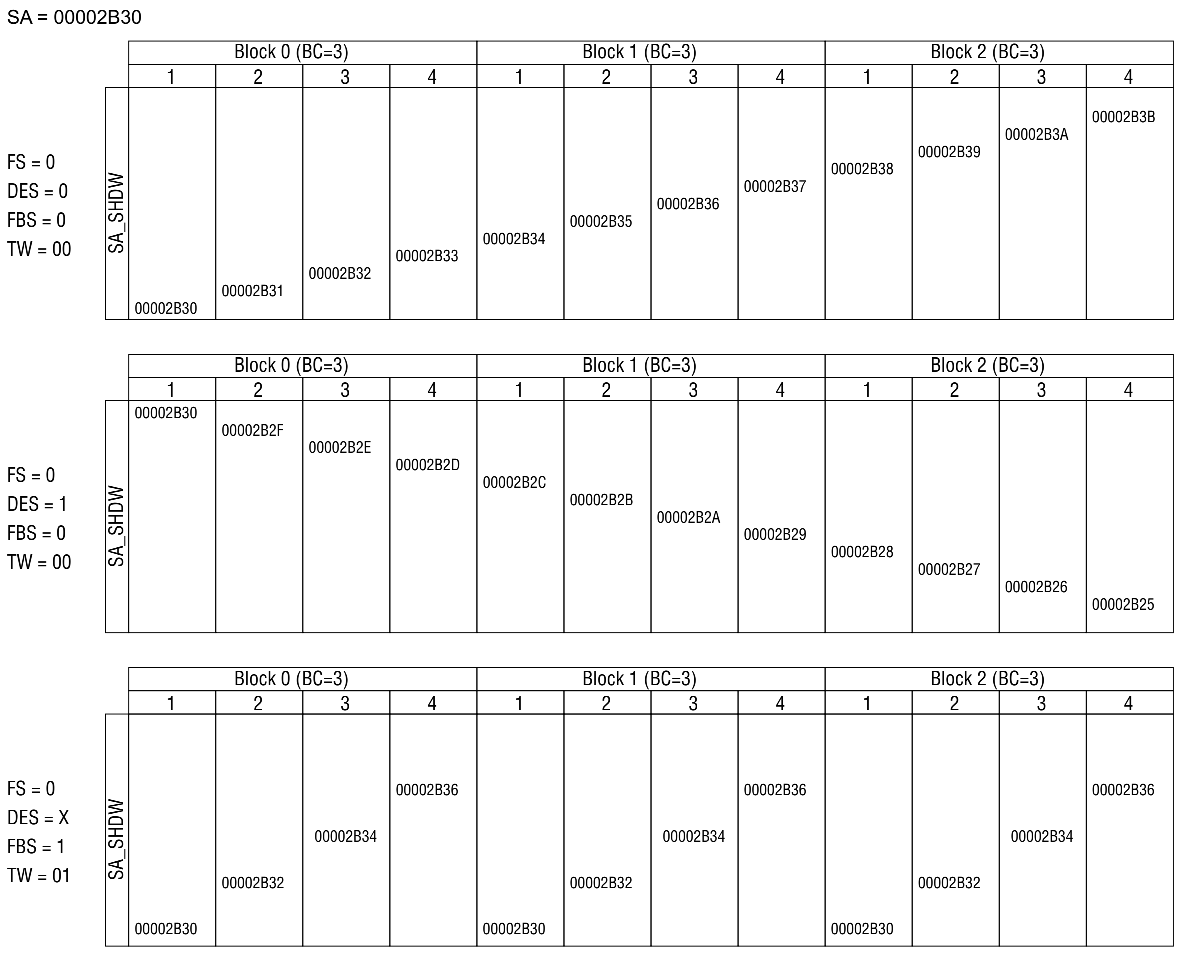


Figure 3-28: Illustration of SASHDW update

At the successful end of a DMA transfer Source Address or Destination Address can be updated with the value stored in Source Address Shadow or Destination Address Shadow respective. This can be configured with the bits Update Source Address (DMAi_Dn:US) or Update Destination Address (DMAi_Dn:UD).

3.10.4.2.3 DMA Transfer Size

The DMA transfer size is calculated by the following formula:

DMA transfer size [Byte]= Number of data transfers * (2^Transfer Width) = (BC + 1) * (TC + 1) * (2^TW)

Transfer Count (DMAi_An:TC[15:0]) determines the number of blocks to be transferred in a DMA transfer. Block Count (DMAi_An:BC[3:0]) determines the number of data transfers in each block.

3.10.4.2.4 DMA Transfer Completion and Error Handling

Each DMA Channel issues an interrupt at the end of a DMA transfer. This can be either a completion interrupt if the DMA transfer completed successfully, or an error interrupt in case of an error condition, or a stop request. A completion interrupt is signalled with Flag of DIRQ (DMAi_Bn:DQ) and an error interrupt with Flag of EDIRQ (DMAi_Bn:EQ). There is a DMA Transfer end code associated with each interrupt which is encoded in Stop Status (DMAi_Bn:SS[2:0]).

If a DMA transfer is completed successfully and the completion interrupt raised, Stop Status will show ‘Normal end’ (SS = 101). If it is ended due to an error and the error interrupt raised, Stop Status will show one of the following possibilities:

- Source access error (SS = 011)
- Destination access error (SS = 100)

A ‘Stop request’ during a running DMA transfer can be caused by assertion of the stop request signal (DSTP) of the DMA transfer requesting client, if the DMA Channel is disabled (DMAi_An:EB), if the complete DMA Controller is disabled (DMAi_R).

Both interrupts, completion as well as error interrupt can be masked with bits Completion Interrupt (DMAi_Bn:CI) and Error Interrupt (DMAi_Bn:EI) respective. If these bits are set to 1 the interrupts are not masked. All unmasked completion interrupts are OR’ed and signalled to the Interrupt Controller. The same is done for the error interrupts.

All completion interrupt flags are, in addition to the channel registers, available in two 32-bit registers (DMAi_DIRQ1 and DMAi_DIRQ2) for easier software handling. All error interrupts are handled in the same way and are available in registers DMAi_EDIRQ1 and DMAi_EDIRQ2.

Completion interrupt DQ must be cleared by setting Clear DIRQ (DMAi_Cn:CD). Error interrupt EQ must be cleared by setting Clear EDIRQ (DMAi_Cn:CE). Stop Status will be cleared to ‘Initial value’ (SS = 000) if DQ or EQ is set to 1.

3.10.4.2.5 Channel Disabling and Halting

After a reset, DMA Channels are disabled by default in order to ensure that they are correctly configured before DMA requests are serviced. A DMA Channel is enabled by setting Channel Enable (DMAi_An:EB) to 1. After setting EB to 1, the channel waits for a DMA request.

Each DMA Channel can be independently disabled. This is done by setting Channel Enable (DMAi_An:EB) to 0. Setting this bit to 0 can be done at any time but has different effects, depending on when it is done. If EB is set to 0 during a running DMA transfer, the transfer is stopped at the next transfer gap, an error interrupt is raised and the channel is disabled. Transfer gap means the DMAC has transferred a block of data and the AHB master interface releases the bus request for a few cycles. If EB is set to 0 while an interrupt is pending (DMAi_Bn:DQ = 1 or DMAi_Bn:EQ = 1) or no DMA transfer is running, there will be no other effect besides the channel being disabled.

Channel halting is done by setting Pause Bit (DMAi_An:PB) to 1. If this bit is set to 1 during a running DMA transfer, it will halt after completion of the current transfer block. If it is set to 1 before receiving a transfer request, the halt state is entered immediately. Clearing this bit will put the channel into run state and it will wait for the next transfer request to continue the DMA transfer or, if a transfer request is already pending, it will continue immediately.

3.10.4.3 Burst Transfer Mode

Burst transfer mode is almost identical to block transfer mode. The only difference is the number of requests required during the DMA transfer. In block transfer mode, a request is required for each block of data. In burst transfer mode, only one request is needed at the beginning of the DMA transfer (for the complete transfer). In this mode, the requests needed for the subsequent blocks of data are generated internally by the DMA Controller itself.

3.10.4.4 Demand Transfer Mode

In demand transfer mode, the DMA Client requests data to be transferred as long as the transfer request is asserted. However, the length of a transfer is limited to ‘Block Count’ number of data transfers during an arbitration phase. The DMA Client can control the transfer of data over time by asserting or de-asserting the request signal. The DMA transfer will be successfully completed if the specified number of data transfers are done without error or, it will be unsuccessfully completed if an error condition occurred.

After each transferred block of data, the DMA Arbiter does another arbitration and proceeds with the next requesting channel with the highest priority. The arbitration depends on the selected arbitration scheme and the set priorities of the requesting channels (see 3.10.6 for the details about the arbitration).

NOTE In the MB88F33x ‘Indigo2(-x)’, no DMA client requires “Demand Transfer Mode”.

3.10.4.4.1 DMA Transfer Requests

In Demand transfer mode, only hardware requests are possible.

For a channel hardware request Input Select (DMAi_An:IS) need to be set to ‘01’. The DMA Client which is routed through the DMA Client Matrix to the channel will give the trigger by asserting DREQ. Refer to 3.10.5 for the function and configuration of the DMA Client Matrix.

The DMA Client need to assert DREQ until the DMAC acknowledges this request by asserting DREQ_ACK. From this point in time, data is transferred as long as DREQ is asserted. The maximum number of data transfers that can be done is determined by Block Count (DMAi_An:BC[3:0]). The DMA Client can also de-assert its request if no data transfer is wanted. In order to not block the DMAC for too long by de-asserting DREQ this time is limited.

The DMA Client can de-assert DREQ for Time out (DMAi_An:TO[3:0]) cycles (DMAC clock cycles) continuously in order not to get a time-out. When DREQ is asserted again before time-out has been reached, time-out is reset to the value set in TO.

When time-out has been reached, the arbitration phase for this channel is ended and DREQ_ACK is de-asserted. The arbiter then continues with the next requesting channel with the highest priority. Software requests are not available in Demand transfer mode.

3.10.4.4.2 Block of Data

A block of data is determined by the setting of Block Count (DMAi_An:BC[3:0]) and Transfer Width (DMAi_Bn:TW[1:0]). The DMA Controller can make BC + 1 data transfers at maximum from source address range starting at Source Address (DMAi_SAn:SA) to destination address range starting at Destination Address (DMAi_DAn:DA). The DMA Client controls how many data transfers will be made during the arbitration phase. The AHB Master Interface will issue only alternating SINGLE reads and SINGLE writes. See Table 3-11 on page 76 with settings BC = 0-16, BL = SINGLE, and AL = 1 for the possible AHB transfers.

Addressing in Demand transfer mode is equal to the addressing in Block transfer mode described in section “3.10.4.2.2 Block of Data”.

3.10.4.4.3 DMA Transfer Size

The DMA transfer size is calculated by the following formula:

$$\text{DMA transfer size[Byte]} = \text{Number of data transfers} \times (2\text{^Tansfer Width}) =$$

$$= (\text{TC+1}) \times (2\text{^TW})$$

Transfer Count (DMAi_An:TC[15:0]) determines the number of data transfers to be done in a DMA transfer.

3.10.4.4.4 DMA Transfer Completion and Error Handling in Demand Transfer Mode

DMA transfer completion and error handling in Demand transfer mode is equal as in Block transfer mode described in “DMA Transfer Completion and Error Handling” on page 81.

3.10.4.4.5 Source and Destination Protection in Demand Transfer Mode

Source and destination protection in Demand transfer mode is equal as in Block transfer mode described in “Channel Disabling and Halting” on page 81.

3.10.4.4.6 Channel Disabling and Halting in Demand Transfer Mode

Channel disabling and halting in Demand transfer mode is equal as in Block transfer mode described in “Channel Disabling and Halting” on page 81.

3.10.5 DMA Client Matrix

3.10.5.1 Overview

The DMA Client Matrix provides the possibility to route ‘M’ DMA clients to ‘N’ DMA Channels. ‘M’ is greater than or equal to ‘N’. The selection which DMA Channel serves which DMA client will be set with Client Interface (DMAi_CMCHICn:CI). The configuration of the DMA clients will be done with the registers DMAi_CMICICj.

3.10.5.2 MB88F33x ‘Indigo2(-x)’ Client Table

Table 3-14: Client table DMAC

Number	Client	Description
29	GC_1	Programmable DMA request of Global Control 1
28	GC_0	Programmable DMA request of Global Control 0
27	PRG_CRC	DMA request for buffer empty of Programmable CRC
26	SPI_RX	DMA read request from external device SPI
25	SPI_TX	DMA write request form external device SPI
24	SPI_FLSH_RX	DMA read request from external flash SPI
23	SPI_FLSH_TX	DMA write request from external flash SPI
22	PPG_1	DMA request from pulse pattern generator 1
21	PPG_0	DMA request from pulse pattern generator 0
20	LIN_RX	DMA request for reception from LIN interface
19	LIN_TX	DMA request for transmission from LIN interface
18	SG	DMA request for register update from Sound Generator
17	I2C_1_RX	DMA request for reception complete from I2C interface
16	I2C_1_TX	DMA request for transmission complete from I2C interface
15	I2C_0_RX	DMA request for reception complete from I2C interface
14	I2C_0_TX	DMA request for transmission complete from I2C interface
13	ADC_REQ2	DMA request for scan end from ADC
12	ADC_REQ	DMA request for conversion end from ADC
11	EIRQ_1	DMA request from external interrupt 1
10	EIRQ_0	DMA request from external interrupt 0
9	RLT_1	DMA request from reload timer 1
8	RLT_0	DMA request from reload timer 0
Client0 - Cleint7 are not used in MB88F33x ‘Indigo2(-x)’		

3.10.5.3 Programming Information

- Channel settings are applied for each channel with channel enable .eb low $\rightarrow$ high only (register DMAi_A0 field EB)
- Global and Matrix settings with dmac enable .de low $\rightarrow$ high only (register DMAi_DMACR field DE)
- A software irq clear is required after every end of a hw DMA request.
  To reduce the interrupts setup "block mode" where a single hardware trigger initiates a transfer of a chunk of transfer-width (≤16 double words). An interrupt is only generated after transferring transfer-count (≤ 65536) chunks.

3.10.5.4 Modes of Operation

Each DMA Client Interface can work in one of the following modes:

Disabled Mode

An DMA Client Interface is disabled if it is not selected by any of the DMA Client Matrix Channel Configuration registers (DMAi_CMCHICn:CI). Reconfiguration of the internal DMA Client Interface shall only be done in disabled mode.

Normal Mode

In this mode, a DMA Channel is routed directly to the specified (DMAi_CMCHICn:CI) DMA client. The operation of the DMA Client Matrix in this mode is fully transparent and behaves as if the DMA client would be connected directly to the DMA Channel Interface.

3.10.5.5 Functional Description

Purpose of the DMA Client Matrix is to provide flexibility in the use of available DMA Channels. The configuration of the DMA Client Matrix is intended to be static and shall be done after the boot code execution when the software is setting up the system. However, the DMA Client Matrix configuration can be changed during normal operation of the system if the procedure description in sections "Selecting the same DMA Client Interface in two or more DMA Channel Interfaces leads to unpredictable behavior of the DMAC. Therefore DMAi_CMCHICn:CI must be properly configured before enabling the DMAC and one or more of its channels." is followed.

3.10.5.5.1 Structure of the DMA Client Matrix

The DMA Client matrix will be a full matrix where each DMA Client Interface ‘m’ can be routed to every DMA Channel Interface ‘n’.

3.10.5.5.2 DMA Client Matrix Configuration

The configuration of the DMA Client Matrix is done with the following registers:

- DMAC Client Matrix Internal Client Configuration registers (DMAi_CMICICj)
- DMAC Client Matrix Channel Interface Configuration registers (DMAi_CMCHICn)

The DMAC Client Matrix Internal Client Interface Configuration Register contains two signal behavior bits. For their function see the descriptions below.

The config bit Behavior Request Acknowledge, DMAi_CMICICj:BEHREQACK, sets the behavior of the output signal DREQ_ACK[j] if the internal DMA Client Interface ‘j’ is not selected in any of the channel configuration registers (DMAi_CMCHICn). The user can choose whether DREQ_ACK[j] drives inactive level or DREQ[j] is connected to DREQ_ACK[j], in that case. Due to software misbehavior falsely set, the later can be used to reset a DMA request signal without violating the two-way handshake protocol.

The DMAC Client Matrix Channel Interface Configuration Register contains up to nine selection bits. For their function see the description below.

The selection bits Client Interface, DMAi_CMCHICn:CI, specify which DMA Client Interface ‘m’ is connected to the DMA Channel Interface ‘n’. The configuration of these bits must take place before DMAi_R:DE and DMAi_An:EB is set to 1. The client interface number must be programmed as binary value to DMAi_CMCHICn:CI. Setting of CI makes the connection between DMA Client Interface defined by the value of CI and DMA Channel Interface ‘n’. Selecting twice or more times the same DMA Client Interface in any of the DMA Client Matrix Channel Configuration registers results in unpredictable behavior of the DMAC and must be avoided.

Availability of certain DMA Clients depends on specific device.

3.10.5.6 Initialization and Application Information

3.10.5.6.1 Reset

The reset state of each DMA Client Matrix configuration bit is shown in the register description of DMAi_CMICICj and DMAi_CMCHICn. To summarize it, after hardware reset, all internal DMA Client Interfaces are disabled, all signals set to high-active level and DMA Channel Interface 0 - ‘N-1’ is configured to route to DMA Client Interface 0 - ‘N-1’.

NOTE Selecting the same DMA Client Interface in two or more DMA Channel Interfaces leads to unpredictable behavior of the DMAC. Therefore DMAi_CMCHICn:CI must be properly configured before enabling the DMAC and one or more of its channels.

3.10.6 DMA Arbiter

3.10.6.1 Overview

The DMA Arbiter is responsible for choosing a DMA Channel based on the arbitration scheme selected in the Global Configuration Register (DMAi_R:PR). There are three arbitration schemes available:

- Fixed Priority
- Dynamic Priority
- Round-Robin

The arbitration schemes are explained in detail in the following sections. The arbitration scheme can be changed any time. However, it becomes effective only after the current running data transfer has been completed at the next transfer gap.

3.10.6.2 Fixed Priority

In Fixed Priority arbitration scheme, the DMA Channels have a “fixed” priority which can be set with Priority Number (DMAi_Bn:PN). Priority Number equal to 0 has the highest priority whereas Priority Number equal to 127 has the lowest priority.

DMA Channels with equal Priority Number, the channel with the lowest channel number ‘n’ has the highest priority. The initial value of DMAi_Bn:PN is 127. Priority Number can be changed any time. However, it becomes effective only after the current running data transfer has been completed at the next transfer gap. Table Table 3-15 shows an arbitration example for 8 DMA Channels to illustrate the behavior.

Table 3-15: Fixed Priority Arbitration Example

Arbitration Cycle	Requesting DMA Channel ‘n’	PN of requesting DMA Channel ‘n’	Grant given to DMA Channel ‘n’
x+1	2	0	2
	4	2	
	7	6	
x+2	4	2	4
	7	6	
	8	5	
x+3	4	2	4
	7	6	
	8	5	
x+4	1	5	1
	7	6	
	8	5	

Table 3-15: Fixed Priority Arbitration Example (Continued)

Arbitration Cycle	Requesting DMA Channel ‘n’	PN of requesting DMA Channel ‘n’	Grant given to DMA Channel ‘n’
x+5	3	9	8
	7	6	
	8	5	

3.10.6.3 Dynamic Priority

The Dynamic Priority arbitration scheme is an extension of the Fixed Priority arbitration scheme. The priority of the DMA Channels is dynamically adjusted based on the criterion whether a channel got a grant or not. If a channels request was granted, its dynamic priority number is loaded with the Priority Number stored in DMAi_Bn:PN and, if a channels request was not granted, its dynamic priority number is decremented by 1.

The arbiter is giving grant to the requesting DMA Channel with the lowest dynamic priority number. If two or more requesting channels have equal dynamic priority numbers, the DMA Channel with the lowest channel number ‘n’ has the highest priority and will win the arbitration process. Priority Number can be changed any time. However, it becomes effective only after the current running data transfer has been completed at the next transfer gap. Table 3-16 shows an arbitration example for 4 DMA Channels to illustrate the behavior.

Table 3-16: Dynamic Priority Arbitration Example

Arbitration Cycle	Requesting DMA Channel		Dynamic PN of DMA Channel	PN of DMA Channel	Grant given to DMA Channel
1	Ch. 0	yes	1	1	0
	Ch. 1	yes	2	2	
	Ch. 2	yes	3	3	
	Ch. 3	no	3	3	
2	Ch. 0	no	1	1	1
	Ch. 1	yes	1	2	
	Ch. 2	yes	2	3	
	Ch. 3	no	3	3	
3	Ch. 0	no	1	1	2
	Ch. 1	no	2	2	
	Ch. 2	yes	2	3	
	Ch. 3	yes	3	3	
4	Ch. 0	no	1	1	1
	Ch. 1	yes	2	2	
	Ch. 2	no	3	3	
	Ch. 3	yes	2	3	
5	Ch. 0	no	1	1	3
	Ch. 1	no	2	2	
	Ch. 2	yes	3	3	
	Ch. 3	yes	1	3	

3.10.6.4 Round-Robin

In Round-Robin arbitration scheme, the turn is rotated in directional and cyclic order from DMA Channel 0 to DMA Channel ‘n’. At most, one DMA Channel request can be granted at any time, this is defined as a turn being given. The turn is moved forward at each transfer gap.

The turn’s rotation isn’t strictly Round-Robin in order not to waste an arbitration phase by giving the turn to a non-requesting DMA Channel. Instead, the turn is given to the next requesting DMA Channel in the rotation direction. If no DMA Channel was served last (only possible in initial state), the requesting DMA Channel with the lowest channel number ‘n’ has the highest priority and will win the arbitration process. Table Table 3-17 shows an arbitration example for 8 DMA Channels to illustrate the behavior.

Table 3-17: Round-Robin Arbitration Example

Arbitration Cycle	Requesting DMA Channels	Grant given to DMA Channel	Last served DMA Channel
1	2	2	none
	4		
	7		
2	4	4	2
	7		
	8		
3	4	7	4
	7		
	8		
4	1	8	7
	4		
	8		
5	1	1	8
	4		
	7		

3.10.6.5 Application Information

3.10.6.5.1 Fixed Priority Arbitration

With this arbitration scheme, the DMA Channel request from the channel with the highest priority will be selected for service. If the DMAC is programmed, that channel 0 is assigned the highest priority and this channel has a higher service request rate compared to the other channels. It is possible that this channel absorbs the complete bandwidth of the DMA Controller, which means, that the other channels will not be serviced.

3.10.6.5.2 Dynamic Priority Arbitration

With this arbitration scheme, starving is tried to be avoided by assigning a requesting channel which got no grant the next higher priority level. However, starving cannot be avoided if the DMAC is not programmed properly. I.e., if channel 0 is assigned the highest priority, the other channels cannot reach a higher priority than channel 0. If this channel has in addition a higher service request rate that the other channels, it will use the complete bandwidth of the DMAC.

3.10.6.5.3 Round-Robin Arbitration

With this arbitration scheme, starving of requesting channels is not possible even if channel 0 has a service request rate that is equal to or exceeds the arbitration rate.

3.10.7 DMA AHB Slave Interface

This is the DMA controllers system interface through which the DMACs registers are accessed.

3.10.7.1 Supported Data Transfers

The Slave Interface supports 8-, 16-, and 32-bit wide AHB data transfers. 16-bit and 32-bit accesses shall be 16-bit address respective 32-bit address aligned.

Single data and fixed incremental burst accesses are supported (SINGLE, INCR4, INCR8, and INCR16).

3.10.7.2 Data Transfer Response

The DMA AHB Slave Interface will respond with the following possibilities to any kind of access.

- OKAY response
- ERROR response

The ERROR response will be given for accesses where a register access error occurs.

3.10.7.2.1 Register Access Error

A register access error is raised if a read or write to a reserved address location is attempted. See Memory layout of DMA Controller Registers for the location of the reserved addresses.

3.10.8 DMAC Register Overview

Table 3-18: Registers Overview

Base Address(es)	**Instance no 0: BASEADDR="00028000"**	
Absolute Address	**Register Name**	**Register Description**
BASEADDR + 0x0000	DMAi_A0	DMAC Channel Configuration A Register Channel 0
BASEADDR + 0x0004	DMAi_B0	DMAC Channel Configuration B Register Channel 0
BASEADDR + 0x0008	DMAi_SA0	DMAC Channel Configuration Source Address Register Channel 0
BASEADDR + 0x000C	DMAi_DA0	DMAC Channel Configuration Destination Address Register Channel 0
BASEADDR + 0x0010	DMAi_C0	DMAC Channel Configuration C Register Channel 0
BASEADDR + 0x0014	DMAi_D0	DMAC Channel Configuration D Register Channel 0
BASEADDR + 0x0018	DMAi_SASHDW0	DMAC Channel Configuration Source Address Shadow Register Channel 0
BASEADDR + 0x001C	DMAi_DASHDW0	DMAC Channel Configuration Destination Address Shadow Register Channel 0
BASEADDR + 0x0040	DMAi_A1	DMAC Channel Configuration A Register Channel 1
BASEADDR + 0x0044	DMAi_B1	DMAC Channel Configuration B Register Channel 1
BASEADDR + 0x0048	DMAi_SA1	DMAC Channel Configuration Source Address Register Channel 1
BASEADDR + 0x004C	DMAi_DA1	DMAC Channel Configuration Destination Address Register Channel 1
BASEADDR + 0x0050	DMAi_C1	DMAC Channel Configuration C Register Channel 1
BASEADDR + 0x0054	DMAi_D1	DMAC Channel Configuration D Register Channel 1
BASEADDR + 0x0058	DMAi_SASHDW1	DMAC Channel Configuration Source Address Shadow Register Channel 1
BASEADDR + 0x005C	DMAi_DASHDW1	DMAC Channel Configuration Destination Address Shadow Register Channel 1
BASEADDR + 0x1000	DMAi_R	DMAC Global Configuration Register
BASEADDR + 0x1004	DMAi_DIRQ1	DMAC Global Completion Interrupt 1 Register
BASEADDR + 0x1008	DMAi_DIRQ2	DMAC Global Completion Interrupt 2 Register
BASEADDR + 0x100C	DMAi_EDIRQ1	DMAC Global Error Interrupt 1 Register
BASEADDR + 0x1010	DMAi_EDIRQ2	DMAC Global Error Interrupt 2 Register
BASEADDR + 0x1014	DMAi_ID	DMAC ID Register
BASEADDR + 0x2000	Reserved	Do not modify
BASEADDR + 0x2020	DMAi_CMICIC0	DMAC Client Matrix Internal Client Interface Configuration Register 0 (for Client 8)
BASEADDR + 0x2024	DMAi_CMICIC1	DMAC Client Matrix Internal Client Interface Configuration Register 1 (for Client 9)
BASEADDR + 0x2028	DMAi_CMICIC2	DMAC Client Matrix Internal Client Interface Configuration Register 2 (for Client 10)
BASEADDR + 0x202C	DMAi_CMICIC3	DMAC Client Matrix Internal Client Interface Configuration Register 3 (for Client 11)
BASEADDR + 0x2030	DMAi_CMICIC4	DMAC Client Matrix Internal Client Interface Configuration Register 4 (for Client 12)
BASEADDR + 0x2034	DMAi_CMICIC5	DMAC Client Matrix Internal Client Interface Configuration Register 5 (for Client 13)
BASEADDR + 0x2038	DMAi_CMICIC6	DMAC Client Matrix Internal Client Interface Configuration Register 6 (for Client 14)
BASEADDR + 0x203C	DMAi_CMICIC7	DMAC Client Matrix Internal Client Interface Configuration Register 7 (for Client 15)
BASEADDR + 0x2040	DMAi_CMICIC8	DMAC Client Matrix Internal Client Interface Configuration Register 8 (for Client 16)

Table 3-18: Registers Overview

Base Address(es)	Instance no 0: BASEADDR="00028000"	
Absolute Address	**Register Name**	**Register Description**
BASEADDR + 0x2044	DMAi_CMICIC9	DMAC Client Matrix Internal Client Interface Configuration Register 9 (for Client 17)
BASEADDR + 0x2048	DMAi_CMICIC10	DMAC Client Matrix Internal Client Interface Configuration Register 10 (for Client 18)
BASEADDR + 0x204C	DMAi_CMICIC11	DMAC Client Matrix Internal Client Interface Configuration Register 11 (for Client 19)
BASEADDR + 0x2050	DMAi_CMICIC12	DMAC Client Matrix Internal Client Interface Configuration Register 12 (for Client 20)
BASEADDR + 0x2054	DMAi_CMICIC13	DMAC Client Matrix Internal Client Interface Configuration Register 13 (for Client 21)
BASEADDR + 0x2058	DMAi_CMICIC14	DMAC Client Matrix Internal Client Interface Configuration Register 14 (for Client 22)
BASEADDR + 0x205C	DMAi_CMICIC15	DMAC Client Matrix Internal Client Interface Configuration Register 15 (for Client 23)
BASEADDR + 0x2060	DMAi_CMICIC16	DMAC Client Matrix Internal Client Interface Configuration Register 16 (for Client 24)
BASEADDR + 0x2064	DMAi_CMICIC17	DMAC Client Matrix Internal Client Interface Configuration Register 17 (for Client 25)
BASEADDR + 0x2068	DMAi_CMICIC18	DMAC Client Matrix Internal Client Interface Configuration Register 18 (for Client 26)
BASEADDR + 0x206C	DMAi_CMICIC19	DMAC Client Matrix Internal Client Interface Configuration Register 19 (for Client 27)
BASEADDR + 0x2070	DMAi_CMICIC20	DMAC Client Matrix Internal Client Interface Configuration Register 20 (for Client 28)
BASEADDR + 0x2074	DMAi_CMICIC21	DMAC Client Matrix Internal Client Interface Configuration Register 21 (for Client 29)
BASEADDR + 0x2800	DMAi_CMCHIC0	DMAC Client Matrix Channel Interface Configuration Register 0
BASEADDR + 0x2804	DMAi_CMCHIC1	DMAC Client Matrix Channel Interface Configuration Register 1

3.11 Programmable CRC

The Programmable CRC is a hardware implementation of a serial CRC calculation unit which can be configured by software. The serial CRC logic uses modulo-2 arithmetic to calculate a checksum, so that the CRC module can detect errors in data blocks.

3.11.1 Features of the Programmable CRC

- Programmable 8, 16, 24 or 32 bit input data width.
- Programmable polynomial value (polynomial degree from 2 to 32).
- Programmable initial seed value.
- Programmable final checksum XOR value.
- Interrupt and DMA trigger capability.
- Configurable input/output bit reflection and byte swapping.
- Supports block/multiple data transfers (more than 32-bit).

3.11.2 Areas of Application

- Data security/integrity
- Communication protocols

The Programmable CRC module can be configured to widely-used common CRC standards, of which some are listed below:

- CRC-32-IEEE 802.3
- CRC-16-CCITT
- CRC-8-CCITT
- CRC-5-USB
- CRC-XMODEM
- 12bit-CRC
- 10bit-CRC
- 8bit-CRC

3.11.3 Block Diagram of the Programmable CRC

Figure 3-29 shows the top level block diagram of the Programmable CRC.

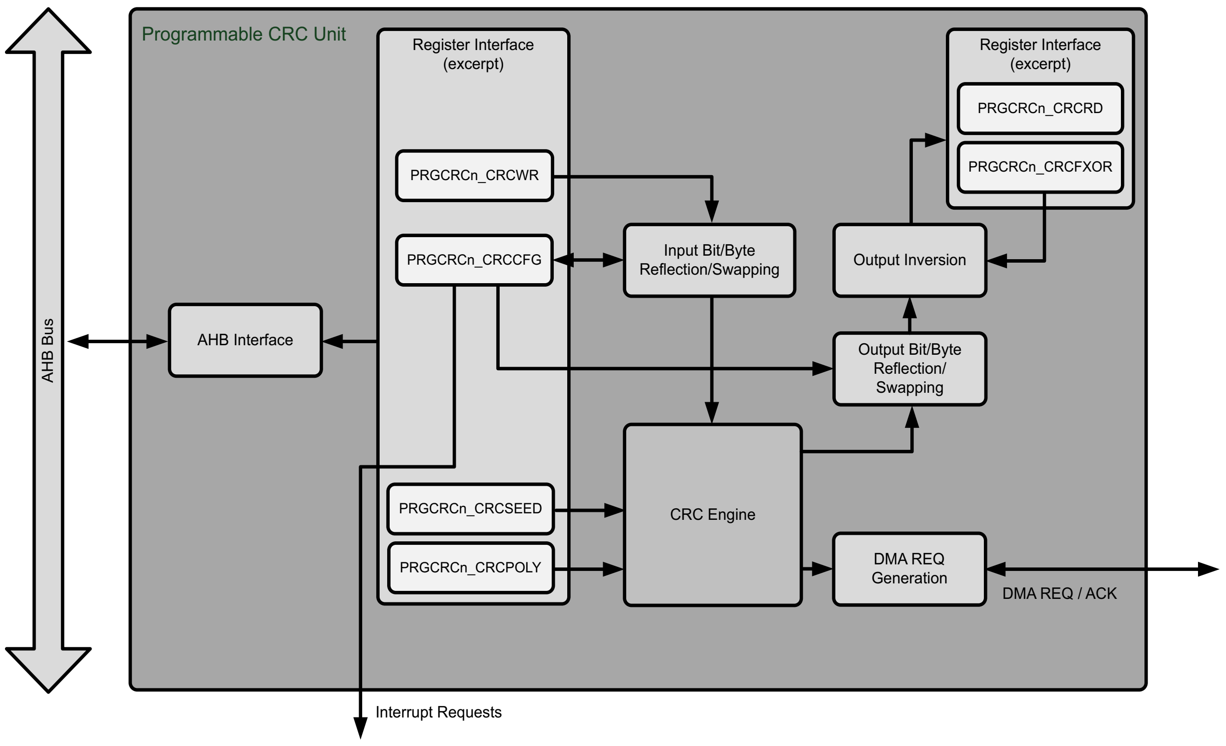


Figure 3-29: Programmable CRC Block Diagram

3.11.4 Operation of Programmable CRC

This section describes flow charts for CRC operation (see Section 3.11.4.1 "CRC Operation Flow Charts"), CRC calculation flow (see Section 3.11.5 "CRC Input Data and Checksum Calculation Flow") and an example for CRC calculation (see Section 3.11.6 "CRC Calculation Example").

3.11.4.1 CRC Operation Flow Charts

Figure 3-30, Figure 3-31, and Figure 3-32 show the steps to configure CRC registers and to perform a CRC calculation.

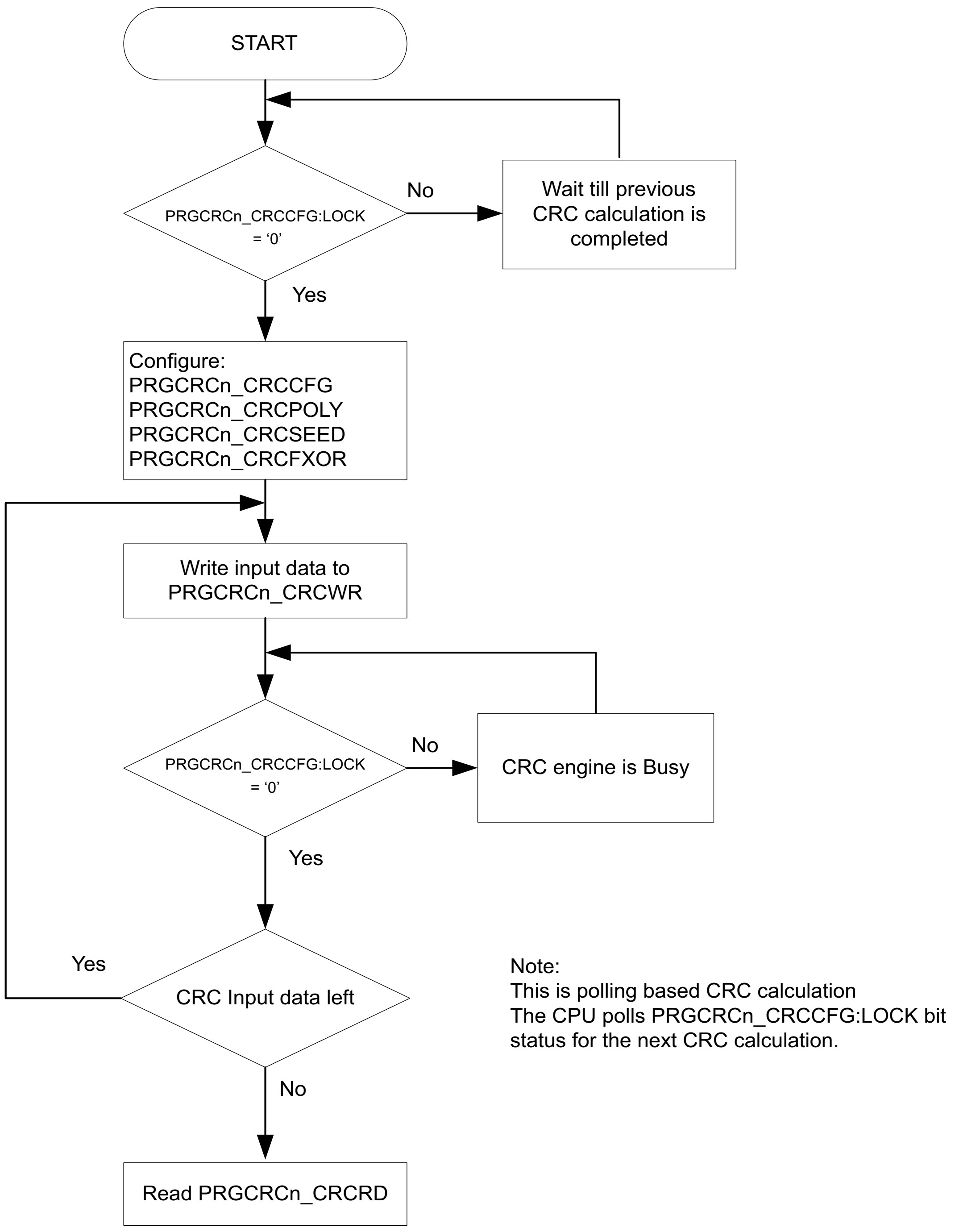

START
PRGCRCn_CRCCFG:LOCK = ‘0’
No
Wait till previous CRC calculation is completed
Yes
Configure:
PRGCRCn_CRCCFG
PRGCRCn_CRCPOLY
PRGCRCn_CRCSEED
PRGCRCn_CRCFXOR
Write input data to PRGCRCn_CRCWR
PRGCRCn_CRCCFG:LOCK = ‘0’
No
CRC engine is Busy
Yes
Yes
CRC Input data left
No
Read PRGCRCn_CRCRD
Note:
This is polling based CRC calculation
The CPU polls PRGCRCn_CRCCFG:LOCK bit status for the next CRC calculation.

Figure 3-30: Polling based CRC operation

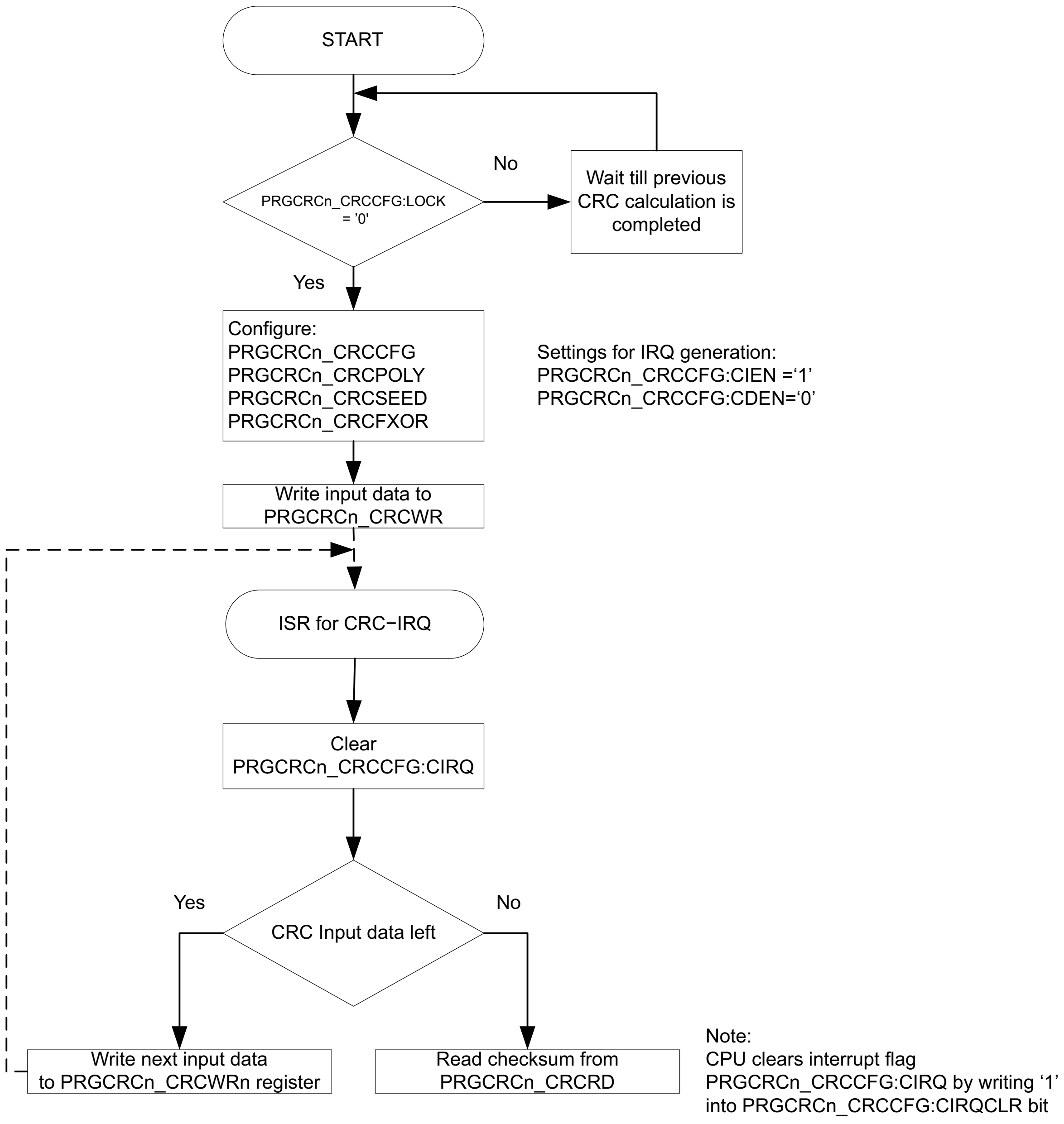

START
PRGCRCn_CRCCFG:LOCK = '0'
No
Wait till previous CRC calculation is completed
Yes
Configure:
PRGCRCn_CRCCFG
PRGCRCn_CRCPOLY
PRGCRCn_CRCSEED
PRGCRCn_CRCFXOR
Settings for IRQ generation:
PRGCRCn_CRCCFG:CIEN =‘1’
PRGCRCn_CRCCFG:CDEN=‘0’
Write input data to PRGCRCn_CRCWR
ISR for CRC−IRQ
Clear PRGCRCn_CRCCFG:CIRQ
Yes
No
CRC Input data left
Write next input data to PRGCRCn_CRCWRn register
Read checksum from PRGCRCn_CRCRD
Note:
CPU clears interrupt flag
PRGCRCn_CRCCFG:CIRQ by writing ‘1’
into PRGCRCn_CRCCFG:CIRQCLR bit

Figure 3-31: CRC operation with IRQ

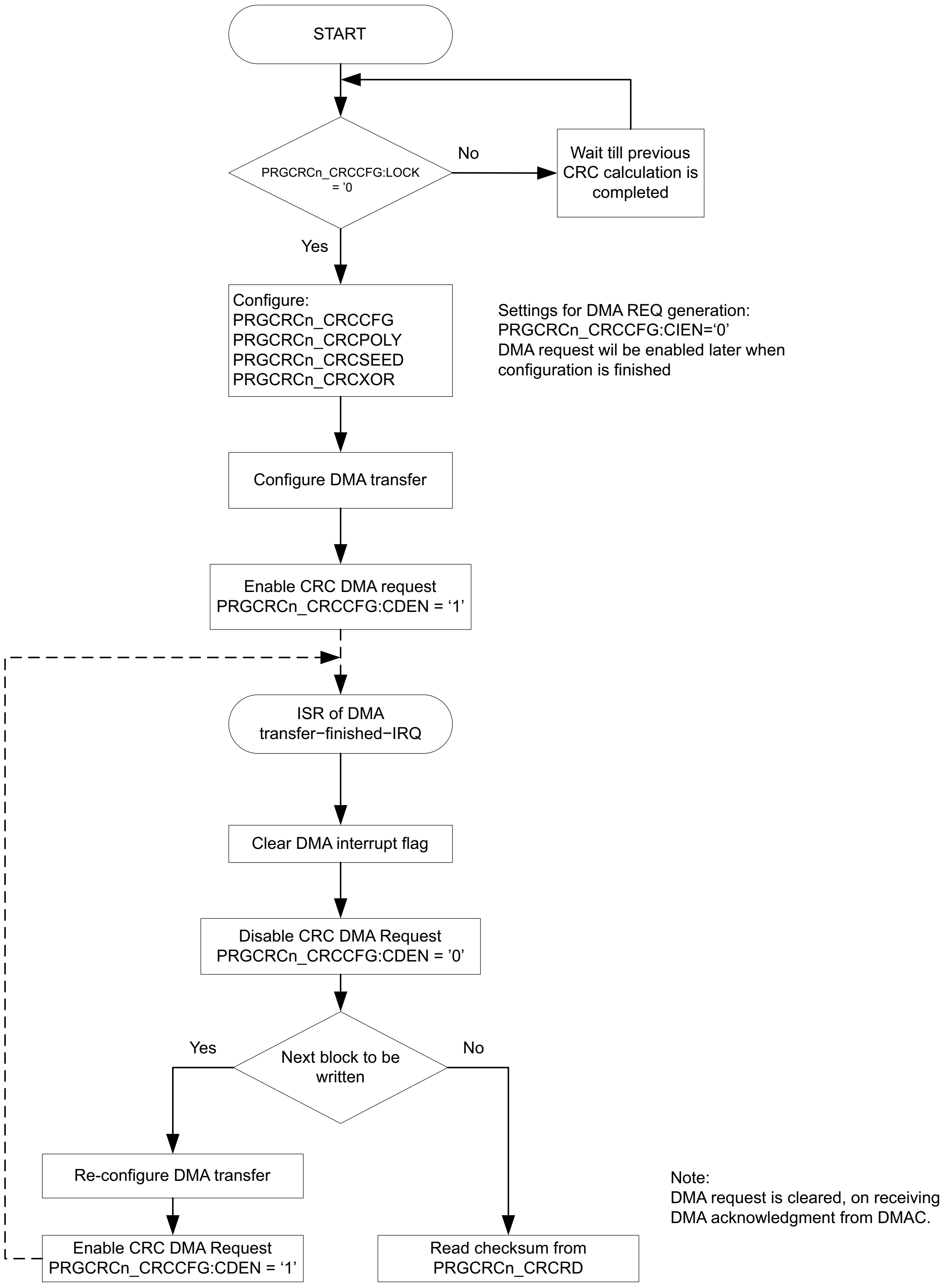

START
PRGCRCn_CRCCFG:LOCK = ’0
No
Wait till previous CRC calculation is completed
Yes
Configure:
PRGCRCn_CRCCFG
PRGCRCn_CRCPOLY
PRGCRCn_CRCSEED
PRGCRCn_CRCXOR
Settings for DMA REQ generation:
PRGCRCn_CRCCFG:CIEN=‘0’
DMA request wil be enabled later when configuration is finished
Configure DMA transfer
Enable CRC DMA request
PRGCRCn_CRCCFG:CDEN = ‘1’
ISR of DMA transfer−finished−IRQ
Clear DMA interrupt flag
Disable CRC DMA Request
PRGCRCn_CRCCFG:CDEN = ’0’
Next block to be written
Yes
No
Re-configure DMA transfer
Enable CRC DMA Request
PRGCRCn_CRCCFG:CDEN = ‘1’
Read checksum from PRGCRCn_CRCRD
Note:
DMA request is cleared, on receiving DMA acknowledgment from DMAC.

Figure 3-32: CRC operation with DMA request

3.11.5 CRC Input Data and Checksum Calculation Flow

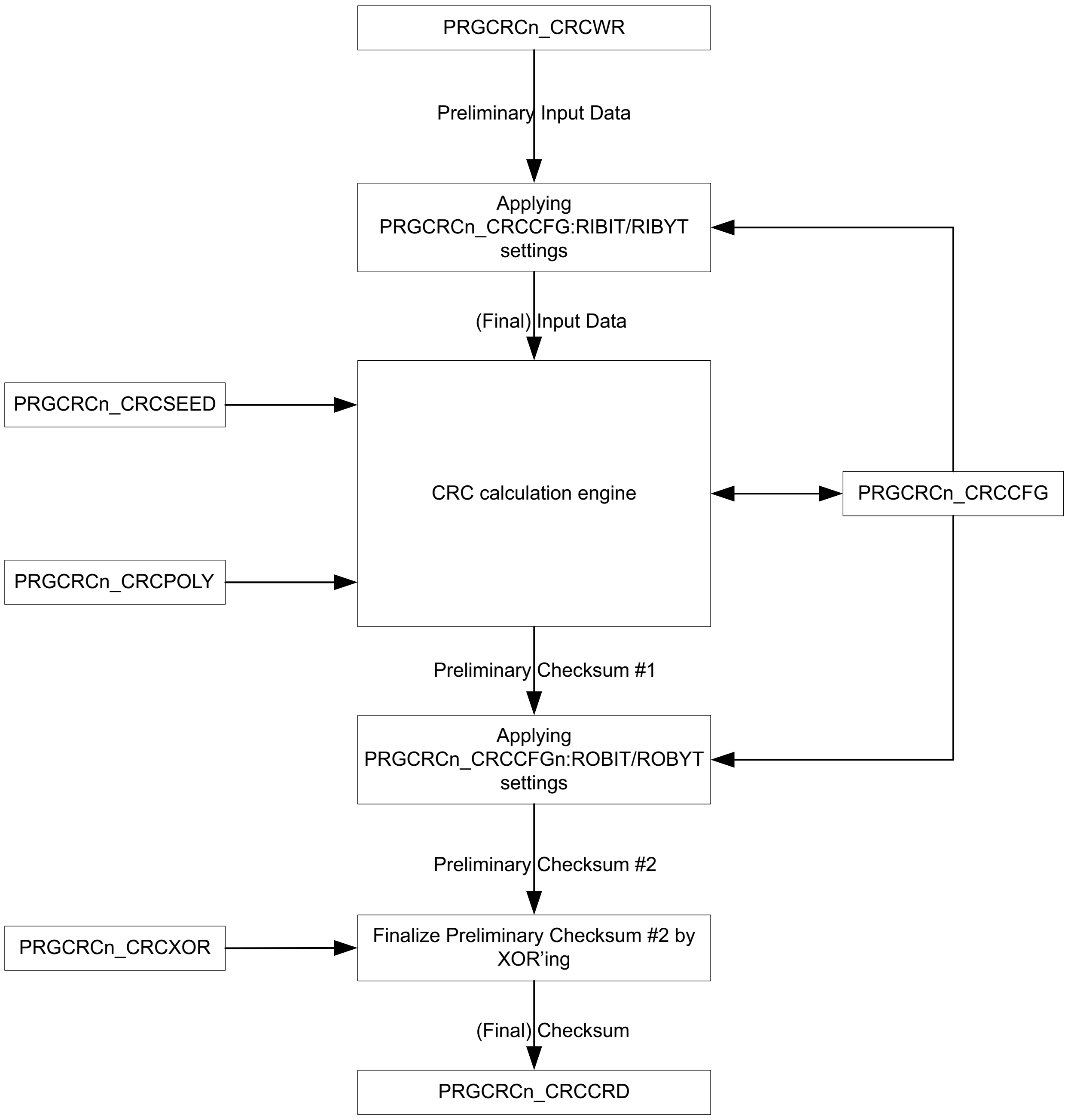


Figure 3-33: Block diagram of CRC input data and checksum calculation flow

- The input data for which CRC is to be calculated is written to PRGCRCn_CRCWR register. This is the ‘Preliminary Input Data’.
- The ’Preliminary Input Data’ bytes can be swapped/reflected bit-wise using PRGCRCn_CRCCFG:RIBIT and/or byte-wise using PRGCRCn_CRCCFG:RIBYT before they enter the CRC engine.
  The settings are shown below:

"Preliminary Input Data" in PRGCRCn_CRCWR register:

A7----A0 B7----B0 C7----C0 D7----D0

If the input data size is less than 32-bit (SZ < 11 (Binary)), then the remaining bits (8,16 or 24-bit) of the data are considered as don’t care (X) as shown in Table 3-19 .

Table 3-19: Preliminary Input Data bit-wise and/or byte-wise reflection/swapping

			"Final Input Data" for CRC engine			
RIBYT	**RIBIT**	**SZ**	**+3**	**+2**	**+1**	**+0**
0	0	00	XXXX XXXX	XXXX XXXX	XXXX XXXX	D7------- D0
		01	XXXX XXXX	XXXX XXXX	C7--------C0	D7--------D0
		10	XXXX XXXX	B7--------B0	C7--------C0	D7--------D0
		11	A7--------A0	B7--------B0	C7--------C0	D7--------D0
	1	00	XXXX XXXX	XXXX XXXX	XXXX XXXX	D0--------D7
		01	XXXX XXXX	XXXX XXXX	C0--------C7	D0--------D7
		10	XXXX XXXX	B0--------B7	C0--------C7	D0--------D7
		11	A0---------A7	B0--------B7	C0--------C7	D0--------D7
1	0	00	XXXX XXXX	XXXX XXXX	XXXX XXXX	D7--------D0
		01	XXXX XXXX	XXXX XXXX	D7--------D0	C7--------C0
		10	XXXX XXXX	D7--------D0	C7--------C0	B7--------B0
		11	D7---------D0	C7--------C0	B7--------B0	A7--------A0
	1	00	XXXX XXXX	XXXX XXXX	XXXX XXXX	D0--------D7
		01	XXXX XXXX	XXXX XXXX	D0--------D7	C0--------C7
		10	XXXX XXXX	D0--------D7	C0--------C7	B0--------B7
		11	D0---------D7	C0--------C7	B0--------B7	A0--------A7

- The ’Preliminary Input Data’ after applying the settings of PRGCRCn_CRCCFG:RIBIT/RIBYT results in the ’Final Input Data’, which is sent to the CRC engine for checksum calculation.
- The PRGCRCn_CRCSEED register provides the initial value to the CRC engine. The required polynomial is provided by PRGCRCn_CRCPOLY register. The CRC engine starts its operation once PRGCRCn_CRCWR register is written with the input data.
- CRC engine performance:

The performance of CRC engine for CRC checksum calculation is based on the input data size and number of clock cycles required to complete a calculation. Table 3-20 shows number of clock cycles required to get final checksum at PRGCRCn_CRCRD register with respect to input data size.

Table 3-20: Clock cycles requirement for checksum calculation

Input data size	**Number of clocks required for final checksum at PRGCRCn_CRCRD**
8-bit	Input data size (8-bit) + 2 = 10 clock cycles.
16-bit	Input data size (16-bit) + 2 = 18 clock cycles.
24-bit	Input data size (24-bit) + 2 = 26 clock cycles.
32-bit	Input data size (32-bit) + 2 = 34 clock cycles.

- The ’Preliminary Checksum #1’ bytes can be swapped/reflected bit-wise using PRGCRCn_CRCCFG:ROBIT and/or byte-wise using PRGCRCn_CRCCFG:ROBYT. The settings are shown in Table 3-21 .

NOTE Only some examples for PRGCRCn_CRCCFG:LEN are shown. The clock considered for the calculation is the bus clock.

Table 3-21: Preliminary Checksum #1 bit-wise and/or byte-wise reflection/swapping
’Preliminary Checksum #1’: S[(LEN-1):0]

			"Preliminary Checksum #2"				
ROBYT	**ROBIT**	**LEN**	**+3**	**+2**	**+1**	**+0**	**Action**
0	0	32	S31---S24	S23---S16	S15---S8	S7---S0	No swapping/reflection. The checksum is aligned with the polynomial degree/length.
		21	0000 0000	000 S20---S16	S15---S8	S7---S0	No swapping/reflection. The checksum is aligned with the polynomial degree/length. The bits S21 to S31 are ’0’.
		16	0000 0000	0000 0000	S15---S8	S7---S0	No swapping/reflection. The checksum is aligned with the polynomial degree/length. The bits S16 to S31 are ’0’.
		3	0000 0000	0000 0000	0000 0000	00000 S2_S0	No swapping/reflection. The checksum is aligned with the polynomial degree/length. The bits S3 to S31 are ’0’.
	1	32	S24---S31	S16---S23	S8---S15	S0---S7	Byte aligned checksum reflection.
		21	0000 0000	S16---S20 000	S8___S15	S0____S7	Bit reflection. The checksum is byte aligned. Bit S21-S23 and S24-S31 are 0.
		16	0000 0000	0000 0000	S8___S15	S0____S7	Bit reflection. The checksum is byte aligned. Bit S24-S31 are 0.
		3	0000 0000	0000 0000	0000 0000	S0---S2 00000	Bit reflection. The checksum is byte aligned. Bit S3-S7 and S8-S31 are 0.
1	0	32	S7---S0	S15---S8	S23---S16	S31---S24	Byte aligned checksum swapping.
		21	0000 0000	S7---S0	S15---S8	000 S20---S16	Byte swapping. The checksum is byte aligned. Bit S21-S23 and S24-S31 are 0.
		16	0000 0000	0000 0000	S7---S0	S15---S8	Byte aligned checksum swapping. Bit S16-S31 are 0.
		3	0000 0000	0000 0000	0000 0000	00000 S2_S0	No byte swapping. Bit S3-S7 and S8-S31 are 0.
	1	32	S0---S7	S8---S15	S16---S23	S24---S31	Bit reflection and byte swapping aligned with polynomial length.
		21	0000 0000	000 S0---S4	S5---S12	S13---S20	Bit reflection and Byte swapping. The checksum is aligned with polynomial length/degree. Bit S21-S23 and S24-S31 are 0.
		16	0000 0000	0000 0000	S0---S7	S8---S15	Bit reflection and Byte swapping. The checksum is aligned with polynomial length/degree. Bit S16-S31 are 0.
		3	0000 0000	0000 0000	0000 0000	00000 S0---S2	Bit reflection and Byte swapping. The checksum is aligned with polynomial. Bit S3-S7 and S8-S31 are 0.

- The checksum after applying settings of PRGCRCn_CRCCFG:ROBIT/ROBYT is ’Preliminary Checksum #2’.
- The ’Preliminary Checksum #2’ is XOR’ed with the contents of PRGCRCn_CRCFXOR register to get ’Final Checksum’.
- The ’Final Checksum’ gets available at PRGCRCn_CRCRD register.

3.11.6 CRC Calculation Example

Consider the following values for calculating 8-bit CRC checksum value.

- Input Data = 0x0F (Hex)
- Polynomial = X^8 + X^2 + X +1
- Seed = 0xFF (Hex)
- Final XOR = 0x00 (Hex)

The coefficients for polynomial are arranged in Table 3-22 .

Table 3-22: Coefficients of the Polynomial

X^8	X^7	X^6	X^5	X^4	X^3	X^2	X^1	X^0
1	0	0	0	0	0	1	1	1

The highest order of coefficient X^8 provides the degree of polynomial/CRC checksum length (PRGCRCn_CRCCFG:LEN = 8), it must not be set to ’1’ while configuring PRGCRCn_CRCPOLY register. Therefore the value of polynomial in accordance with above coefficients is 0x07 (Hex).

The input/output bit reflection is disabled in this example.

The Programmable CRC registers should be configured as follows for the given values:

The CRC configuration register is configured by considering 8-bit input data size and 8-bit polynomial/ checksum length as follows.

PRGCRCn_CRCCFG = 0x00080000 (Hex)

PRGCRCn_CRCPOLY = 0x00000007 (Hex)

PRGCRCn_CRCSEED = 0x000000FF (Hex)

PRGCRCn_CRCFXOR = 0x00000000 (Hex)

PRGCRCn_CRCWR = 0x0000000F (Hex)

The final result of CRC checksum calculation is 0xDE (Hex), which gets available after 11 clock cycles (once PRGCRCn_CRCWR is written) in the PRGCRCn_CRCRD register. If another input data is given to the CRC module, then ’Preliminary Checksum #1’ (0xDE) is used as the initial seed value.

If the new CRC calculation should start from the seed value instead of from the last CRC result, then the PRGCRCn_CRCSEED register needs to be re-written (even if it’s the same value as before).

3.11.7 Programmable CRC Register Overview

Table 3-23: Registers Overview

Base Address(es)	**Instance no 0: BASEADDR="00027000"**	
Absolute Address	**Register Name**	**Register Description**
BASEADDR + 0x0000	PRGCRCn_CRCPOLY	CRC Polynomial Register
BASEADDR + 0x0004	PRGCRCn_CRCSEED	CRC Seed Register
BASEADDR + 0x0008	PRGCRCn_CRCFXOR	CRC Final XOR Register
BASEADDR + 0x000C	PRGCRCn_CRCCFG	CRC Configuration Register
BASEADDR + 0x0010	PRGCRCn_CRCWR	CRC Write Register
BASEADDR + 0x0014	PRGCRCn_CRCRD	CRC Read Register

This page intentionally left blank

Chapter 4: APIX2 Interface

4.1 General

The APIX2 link transmits uncompressed pixel data with a resolution-independent link data rate of up to 3GBit/s over one single pair of STP copper cable. The APIX2 link layer offers two independent video channels. In addition to the pixel data, audio data as well as bidirectional communication or control data can be transmitted. The communication data are protected using the built-in AShell protocol, offering highly robust data transmission over the high speed link by error detection and retransmission mechanisms. The APIX2 devices offer full backwards compatibility to the APIX1 devices, by supporting the APIX1 physical layer as well as offering the downstream and upstream sideband functionality. In addition, existing APIX1 implementations can also be connected to the device.

Definitions:

- The term **TX** is used for a block which transmits pixel or audio data.
  This block is capable of sending and receiving other data.
- The term **RX** is used for a block which receives pixel or audio data.
  This block is capable of sending and receiving other data.
- **Downstream** is the link from TX to RX, means in the same direction as pixel or audio data.
- **Upstream** is the link from RX to TX. Only AShell or GPIO data is transferred in this direction.

4.2 Features

- APIX receiver
- APIX1 and APIX2 mode
- Daisy-chain mode for APIX2
- Downstream link bandwidth: 500MBit/s, 1000MBit/s, 3000MBit/s
- Upstream link bandwidth: 31.25MBit/s, 62.5 MBit/s, 187.5 MBit/s
- Up to two video streams
- I2S audio interface
- GPIO for direct signaling
- AShell generic data interface
- Ethernet via APIX bandwidth: 10MBit/s, 100MBit/s
- Remote commands for software reset, AShell realignment and ticket counter clear
- Local interrupt controller

Limitations:

The following APIX2 interfaces / channels are not supported.

- APIX2 bulk mode
- APIX2 nibble data interface
- APIX2 support data interface
- APIX2 I2C interface
- APIX2 remote controller interface
- APIX2 remote service interface

The APIX2 interface consists of five main blocks:

- APIX2 PHY with PHY configuration register
- APIX2 RX Link Layer with configuration register and AHB master for remote configuration accesses
- AShell remote handler, which handles all sideband AShell commands
- Ethernet over APIX with Ethernet remote handler, which handles ethernet sideband communication
- AShell remote handler that handles all Ethernet over APIX commands.

4.3 Block Diagram

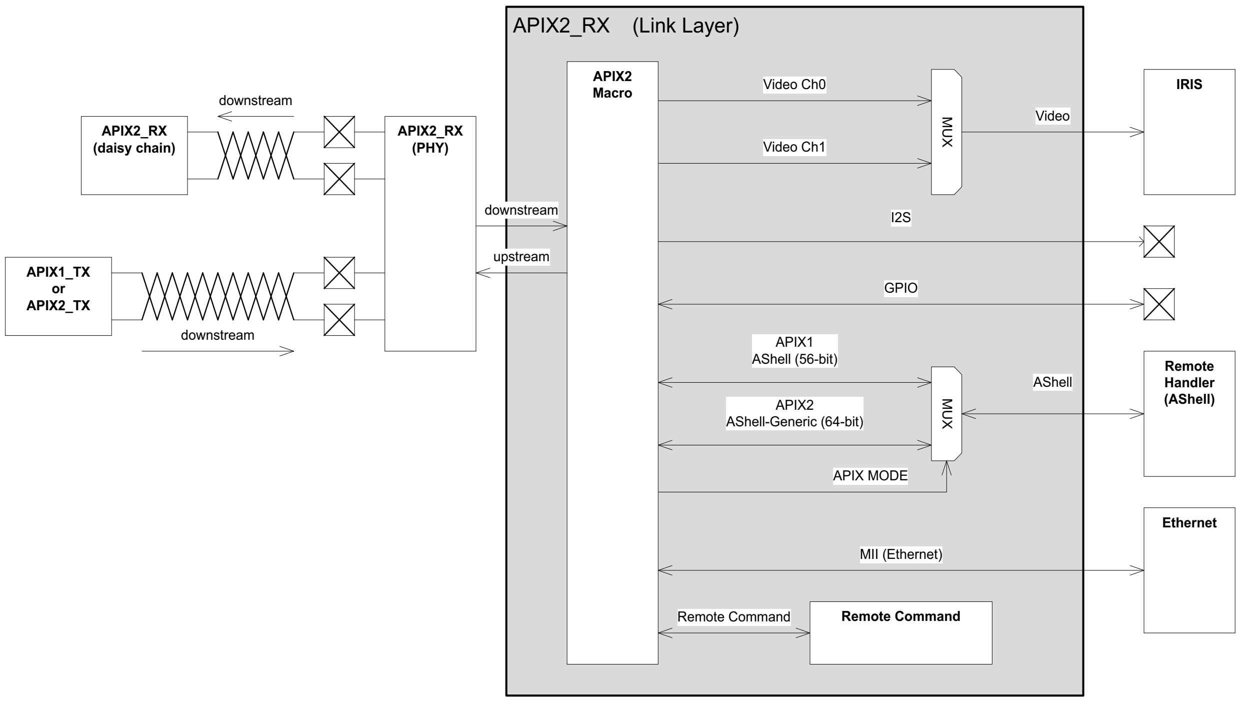


Figure 4-1: Block Diagram

4.4 APIX2 PHY

4.4.1 Overview

The APIX2 PHY unit consists of the analog APIX2 receiver unit together with the PHY related configuration registers, the calibration state machine, the BIST (built-in self test) unit and the test monitor.

4.4.2 Block Diagram

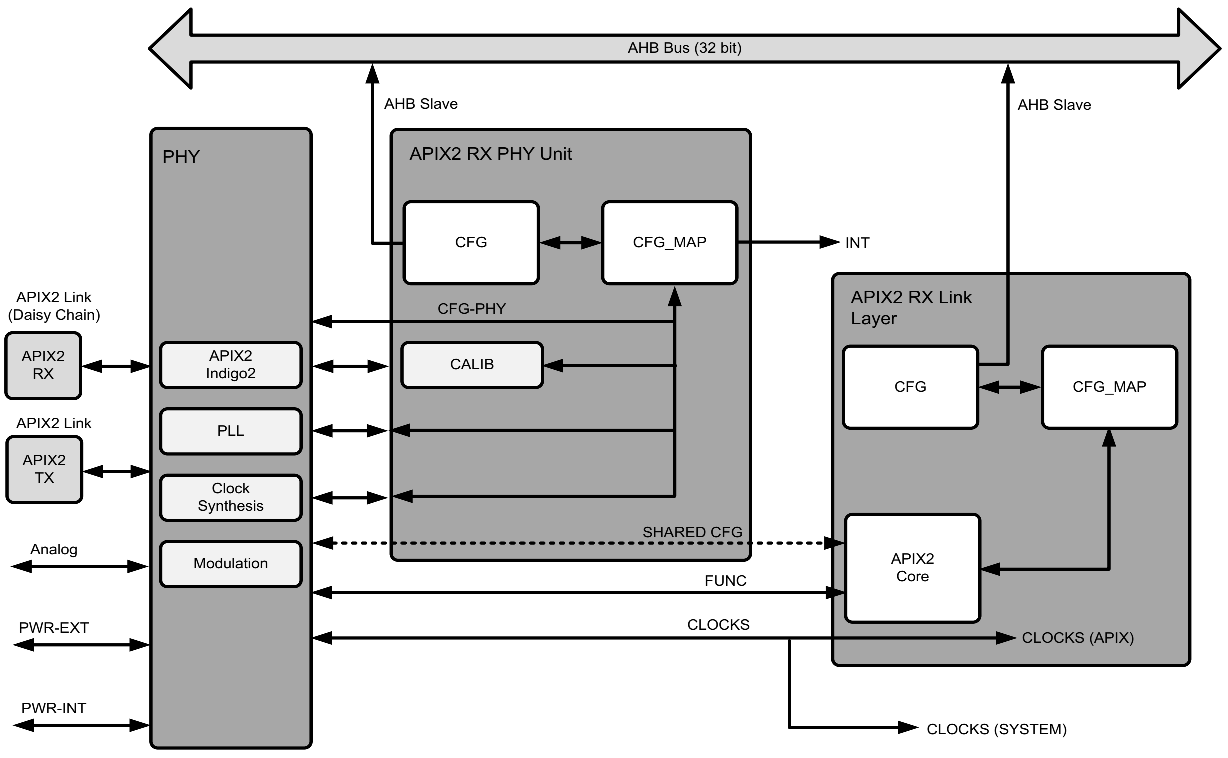


Figure 4-2: APIX2_RX_PHY Block diagram

4.4.3 Daisy-chain Operation

NOTE In our Register Descriptions, ‘Daisy-chain’ is also called ‘looptrough’.

It is possible to connect a second APIX2 RX device (RX1) to the APIX2 TX device through the first APIX2 receiver (RX0). This scenario is also called ‘daisy-chaining’ and is only available in APIX2 mode.

Video and audio channels are shared between the two RX devices. That means that both RX devices are receiving the same video and audio streams. Between TX and RX0 as well as between TX and RX1 independent bi-directional data channels are implemented, which can be used for communication through the asynchronous GPIO channels and for communication, either through the Media Independent Interface or through AShell2 protected data paths.

The downstream link is looped through the APIX2 RX0 device. Therefore, the analog part of the MB88F33x ‘Indigo2(-x)’ devices provides a loop through driver.

The upstream path of the bi-directional data channel between RX1 and TX is be routed through the first receiver RX0.

The following basic configurations must be fulfilled to make a daisy-chain inter-connectivity working:

- enable daisy-chaining in the TX device
- configure the Indigo2 directly connected with the TX as device RX0
  - set APIX_CFG_MODE.apix_rx0_rx1 to ‘0’
- enable daisy-chaining in the RX0 Indigo2 device
  - set APIX_CFG_4.rx_daisy_chain to ‘1’
- setup loop through transmitter in the RX0 Indigo2 device
  - set PHY_LT_CFG_CTRL.lt_tx_rate = APIX_CFG_0.down_bw
  - set PHY_LT_CTRL_1.lt_apix2_mode = 1
  - set BIST_PATTGEN_LINK.bist_down_pattgen = 2
  - set PHY_LT_CTRL_1.lt_select_clksource = 1
  - set PHY_LT_CTRL_1.lt_select_datasource = 0
- configure the second Indigo2 connected to RX0 as device RX1
  - set APIX_CFG_MODE.apix_rx0_rx1 to ‘1’
- disable daisy-chaining in the RX1 Indigo2 device
  - set APIX_CFG_4.rx_daisy_chain to ‘0’

NOTE This functionality is not implemented in MB88F335 ‘Indigo2-S’.

4.4.4 Automatic Gain Control (AGC)

Please refer to the following table for proper settings:

Mode	Description
1Gbps and 500Mbps	DFE and AGC are not connected
3Gbps	DFE and AGC are connected $\rightarrow$ DFE and AGC automatically activated

4.4.5 Application Note

Please refer to the Application Notes: “**APIX FIR SETUP**” and “**APIX PCB-DESIGN GUIDELINE**“ for detailed information.

4.4.6 APIX2 PHY Register Overview

Table 4-1: Registers Overview

Base Address(es)	Instance no 0: BASEADDR="00020000"	
Absolute Address	**Register Name**	**Register Description**
BASEADDR + 0x0000	PHY_RST_CTRL	Control PHY reset and PLL reset
BASEADDR + 0x0004	PHY_RST_STAT	PHY Startup Status
BASEADDR + 0x0008	Reserved	Do not modify
BASEADDR + 0x000C	PHY_PWR_CTRL	PHY Power Control
BASEADDR + 0x0010	PHY_LT_CFG_CTRL	Loopthru configuration
BASEADDR + 0x0014	PHY_LT_CFG_CTRL_1	Loopthru Tx FIR Coefficients
BASEADDR + 0x0018	PHY_LT_CFG_CTRL_2	Loopthru Tx FIR structure
BASEADDR + 0x001C	PHY_LT_CTRL_1	Loopthru configuration
BASEADDR + 0x0020	PHY_LT_CTRL_2	Loopthru Tx Calibration
BASEADDR + 0x0024	PHY_LT_I_STAT	Loopthru Calibration Status
BASEADDR + 0x0028	PHY_LT_P_STAT	Loopthru Calibation Status
BASEADDR + 0x002C	Reserved	Do not modify
BASEADDR + 0x0030	Reserved	Do not modify
BASEADDR + 0x0034	Reserved	Do not modify
BASEADDR + 0x0038	PHY_CDR_CFG	CDR configuration
BASEADDR + 0x003C	Reserved	Do not modify
BASEADDR + 0x0040	Reserved	Do not modify
BASEADDR + 0x0044	Reserved	Do not modify
BASEADDR + 0x0048	PHY_RX_TST	Upstream Swing
BASEADDR + 0x004C	Reserved	Do not modify
BASEADDR + 0x0050	Reserved	Do not modify
BASEADDR + 0x0054	PHY_RX_UP	RX Upstream Calibration
BASEADDR + 0x0058	Reserved	Do not modify
BASEADDR + 0x005C	Reserved	Do not modify
BASEADDR + 0x0060	Reserved	Do not modify
BASEADDR + 0x0064	Reserved	Do not modify
BASEADDR + 0x0068	Reserved	Do not modify
BASEADDR + 0x006C	Reserved	Do not modify
BASEADDR + 0x0070	Reserved	Do not modify
BASEADDR + 0x0074	Reserved	Do not modify
BASEADDR + 0x0078	Reserved	Do not modify
BASEADDR + 0x00E0	Reserved	Do not modify
BASEADDR + 0x00E4	Reserved	Do not modify
BASEADDR + 0x00E8	Reserved	Do not modify
BASEADDR + 0x00EC	Reserved	Do not modify
BASEADDR + 0x00F0	OBS_RX_2	Add observability to DFE regsiter
BASEADDR + 0x00F4	Reserved	Do not modify
BASEADDR + 0x0100	BIST_PATTGEN_LINK	Bist Link Control
BASEADDR + 0x0104	BistTestDuration	test duration
BASEADDR + 0x0108	BistPrbsCfg	setup for 16 bit PRBS generator
BASEADDR + 0x010C	BistChkPrbsCfg	specify a polynomial for 16 bit PRBS checker
BASEADDR + 0x0110	BistCtrl	Trigger to start checking test pattern
BASEADDR + 0x0114	BistDownStatus	Status of Bist from downstream receiver
BASEADDR + 0x0118	Reserved	Do not modify
BASEADDR + 0x011C	Reserved	Do not modify
BASEADDR + 0x0120	Reserved	Do not modify
BASEADDR + 0x0124	Reserved	Do not modify

Table 4-1: Registers Overview

Base Address(es)	**Instance no 0: BASEADDR="00020000"**	
Absolute Address	**Register Name**	**Register Description**
BASEADDR + 0x0180	Reserved	Do not modify
BASEADDR + 0x0184	Reserved	Do not modify
BASEADDR + 0x0200	CLKMEAS_CFG	Clock Measurement Configuration Register
BASEADDR + 0x0204	CLKMEAS_CNT	Clock Measurement Count Register
BASEADDR + 0x0240	Reserved	Do not modify
BASEADDR + 0x0280	MII_CLK_CTRL	MII Clock Control
BASEADDR + 0x0284	Reserved	Do not modify
BASEADDR + 0x0288	SYS_CLK_CTRL	PHY SYS clock control
BASEADDR + 0x028C	PERI_CLK_CTRL	PHY SYS clock control
BASEADDR + 0x0290	VID_CLK_CTRL	PHY VID clock control
BASEADDR + 0x0294	VIDS_CLK_CTRL	PHY VIDS (shifter) clock control

4.5 APIX2 RX Link Layer

In order to keep compatibility to already available APIX1 devices, the APIX2 offers two modes of operation:

- APIX1 Mode
- APIX2 Mode

The two modes differ in available features and interfaces. The table below provides an overview of the differences.

Features / Interfaces	APIX1 Mode	APIX2 Mode
Bandwidth modes	500 MBit, 1 GBit	500 MBit, 1 GBit, 3 GBit
Upstream speeds	31.25 MBit, 62.5 MBit	62.5 MBit, 187.5 MBit
Number of RGB Video channels	1	2
Pixel Control Protection	Not Available	Yes
I2S Audio interface	Not Available	Yes
Protected data transmission over AShell	AShell1 protocol, 56 bit payload, supported if AShell1 available in corresponding device	AShell2 protocol, 64 bit payload
Media Independent Interface (MII)	Not Available	Yes
Asynchronous GPIO interface	Only Available If Internal Ashell1 Is Not Used	Yes
Support for two APIX RX devices (daisy chain)	Not Available	Yes

NOTE APIX2 Mode with 3Gbit is not implemented in MB88F335 ‘Indigo2-S’.

The APIX2 technology offers high speed differential data transmission over a single twisted pair cable. The optional upstream, transferred as separate differential signal, operates at speeds up to 187.5MBit/s and enables full-duplex data communication and GPIO capabilities. At link level, APIX2 uses transmission frames, which are dynamically filled with either video, audio, GPIO or AShell data, depending on the configuration of the link and the application requirements at each interface. This flexible configuration allows APIX2 to be optimized for various applications. On connect of an APIX2 transmitter to a receiver, the receiver automatically starts to ‘scan’ the incoming data stream of the transmitter, recovers and synchronizes to the clock and receives ‘frame alignment’ as soon as it successfully locked to the frame structure. Since this structure is based on various configuration options, it is necessary that transmitter and receiver are configured correctly to the same key frame parameters for link speed, data bandwidth, AShell, GPIO channels and audio channel to reach frame alignment.

4.5.1 APIX1

4.5.1.1 APIX1 Video

The APIX1 video channel is able to transport one video stream, with pixel clock and synchronization information. Therefore, APIX2 embedded offers one parallel RGB interface for the APIX1 video channel.

APIX1 supports RGB modes with 10, 12, 18 and 24 Bit color depth. Each RGB configuration includes the external pixel clock and three pixel control signals like HSYNC, VSYNC and DE.

4.5.1.1.1 APIX1 Data Communication

APIX2 in APIX1 mode offers full duplex, video independent data communication over the high speed serial link. All data transfers are protected by the integrated AShell protocol, using CRC error detection and optional retransmission mechanisms.

4.5.1.1.2 Overview

APIX Automotive Shell (hereinafter referred to as AShell) is Inova Semiconductor’s hardware implementation of an abstraction layer between the APIX Physical Layer (APIX PHY) and its application. An application, AShell and the APIX Physical Layer form a stacked bidirectional wire-based communication system. As part of this communication stack, AShell provides two interfaces for the exchange of messages from local to a remote island or vice versa. AShell hides implementation details and specific features of APIX PHY and provides an appropriate interface to the embedding application of the APIX serial link. This encapsulation provides for higher levels of abstraction and supports a functional layered architecture of the communication system. A pixel (video) data stream of an application to be transmitted from TX to RX device is bypassed and thus not affected by the AShell layer.

As with most layered communication architectures, a collection of related functions is referred to as layer that provides services to the layer above and receives services from the layer below. Each layer has interfaces to the layers above and below. In general, each layer communicates by the means of a protocol with its counterpart on the remote island.

4.5.1.1.3 AShell Services

Apart from the error control functions added to the APIX1 communication stack, the APIX1 AShell is more or less a wrapper layer. The APIX1 AShell provides the following services:

- Transmission of application data ensuring data integrity
- Reception of application data ensuring data integrity
- Supply of information about transmission link status as well as simple error statistics

4.5.1.1.4 AShell Functions

To provide the aforementioned services, APIX1 AShell implements the following functions:

1. Transaction framing and de-framing
   Payload data sinked to or sourced by an application is compiled to or extracted from PDUs (transactions) exchanged as protocol entities between local and remote AShell.
2. Data exchange through APIX1 PHY
   AShell handles various interface signals from APIX1 PHY such that data is sent and received via APIX1 PHY.

3. Data integrity control
   AShell implements a CRC-24 polynomial to detect most transmission errors. Only valid transactions are offered to the application. The transmit path of the AShell generates and the receive path checks the CRC sum that is part of the PDU exchanged between the AShells of both communicating islands.

4. Error control (optional)
   AShell implements a window based ACK protocol to manage bit errors occurring during serial transmission over the wire-based link. As long as the bit errors do not affect synchronization at different levels, communication is kept alive without any intervention by the application. This function can be disabled to support different requirements of the application or software-based implementations of AShell services.

5. Flow control
   AShell implements a back-pressure function to stop the transmitting device sending further payload while the buffers of the receiving device are full.

6. Status report
   Information about the status of APIX1 PHY and the serial transmission link as well as simple error statistics are collected and presented to the application at the interface.

7. Convenient wrapping of the APIX1 PHY interface
   AShell provides an appropriate broker between APIX1 Physical Link Layer and the respective application. AShell thus hides implementation details and special features of APIX PHY.

The AShell must be enabled to get operational and to provide the services.

4.5.1.1.5 AShell Data Interface

The APIX1 AShell offers to transmit generic 56 bit parallel data.

4.5.1.1.6 AShell Back-pressure

In case of application cannot receive further data from AShell, a back-pressure signal is available. As long as back-pressure is active, payload transmission in inbound direction stops. The AShell back-pressure doesn't cause data loss.

Caution: A disabled automatic back-pressure can cause data loss.

4.5.1.1.7 AShell Error Control

Mode	Description
ARQ ON	Automatic retransmission on transmission errors without intervention of the application layer Indication of transmission errors at the inbound interface,
ARQ OFF	No automatic retransmission Indication of transmission errors at the inbound interface Retransmission possible with intervention of the application layer

4.5.1.1.8 ARQ ON, Automatic Retransmission

SDUs accepted by AShell are transmitted until successful reception is acknowledged by the remote counterpart. The acknowledge mechanism is part of the protocol defined by the AShell layer. In case of transmission errors detected through unmatched CRC sums or a back-pressure was requested by the application, all transactions of a certain sliding window including the affected transaction(s) are resent without any intervention by the application. In any case, the order of transactions accepted for transmission is maintained.

To use the protocol, ARQ has to be enabled on TX and RX side.

4.5.1.1.9 AShell Status Signals

AShell provides information about the status of APIX PHY and transmission link as well as simple error statistics. These are presented at the APIX status interface.

4.5.1.1.10 AShell Operational

The operational status signalizes, that transaction alignment between local and remote AShell is established and local AShell is ready to accept transactions from application and to deliver received transactions.

4.5.1.1.11 APIX1 GPIO

As alternative to the protected data communication, in APIX1 mode the APIX2 IP allows to sample one 2bit GPIO input and provides one 2bit GPIO output. The GPIO input is asynchronously sampled and directly transferred through a 2bit width sideband channel to receiver device with lowest delay. The GPIO output reflects the signals sampled at the source device.

The GPIO interfaces have to be enabled to be usable. If they are enabled the APIX1 AShell interface is not operable.

4.5.1.1.12 GPIO Downstream

The sampling speed of the GPIO downstream inputs depends on the APIX1 downstream link rate (1GBit/s or 500MBit/s).

Downstream link rate	Sampling frequency
500 MBit/s	6.94 MHz
1000 MBit/s	13.89 MHz

4.5.1.1.13 GPIO Upstream

The sampling speed of the GPIO upstream inputs depends on the APIX1 upstream link rate (62.5MBit/s or 31.25MBit/s).

Upstream link rate	Sampling frequency
31.25 MBit/s	5.21 MHz
62.5 MBit/s	10.41 MHz

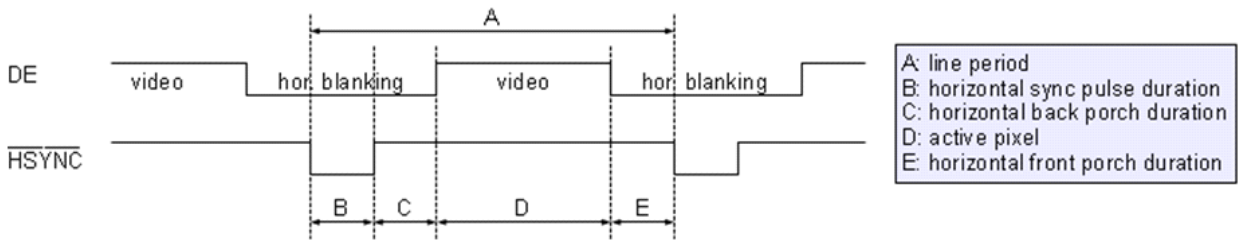
4.5.2 APIX2

4.5.2.1 APIX2 Video

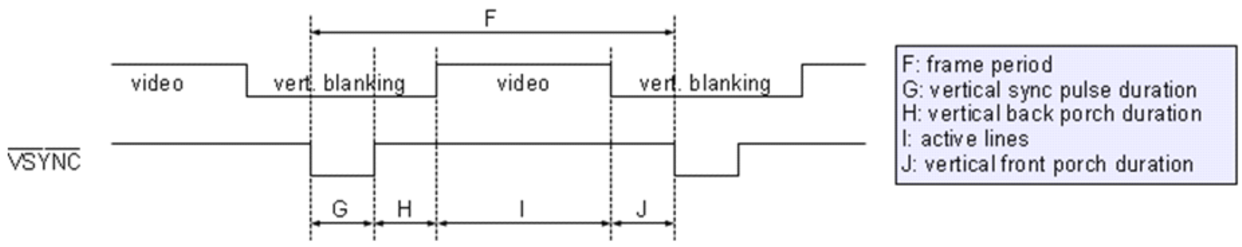
The video channel is able to transport two independent video streams, with individual pixel clock and synchronization information. Therefore APIX2 offers one parallel RGB interface for each video channel.

APIX2 supports RGB modes with 10, 12, 18 and 24 Bit color depth. Each RGB configuration includes the external pixel clock and three pixel control signals like HSYNC, VSYNC and DE.

4.5.2.2 Horizontal Timing Parameter


A
DE
video
hor. blanking
video
hor. blanking
HSYNC
B
C
D
E
A: line period
B: horizontal sync pulse duration
C: horizontal back porch duration
D: active pixel
E: horizontal front porch duration

4.5.2.3 Vertical Timing Parameter


F
video
vert. blanking
video
vert. blanking
VSYNC
G
H
I
J
F: frame period
G: vertical sync pulse duration
H: vertical back porch duration
I: active lines
J: vertical front porch duration

NOTE Important: Timing parameters A to J must be constant for the duration of the video transmission.

4.5.2.4 Absolute Maximum Timings

Timing parameter	Min	Max
D: active pixels per line	127 pixel clocks	2047 pixel clocks
E: horizontal front porch duration	1 pixel clocks	127 pixel clocks
B: horizontal sync pulse duration	1 pixel clocks	255 pixel clocks
C: horizontal back porch duration	1 pixel clocks	511 pixel clocks
I: active lines per frame	127 lines	2047 lines
J: vertical front porch duration	0 lines	15 lines
G: vertical sync pulse duration	0 lines	15 lines
H: vertical back porch duration	0 lines	63 lines

The pixel control information is transferred with protection mechanisms in order to be able to compensate bit errors.

4.5.2.5 APIX2 Audio

APIX2 offers the transmission of real-time digital audio data over the downstream link, with support of up to eight 32 Bit audio channels (4 times stereo).

i2s standard denotation	APIX2 denotation
mclk - master clock	mclk - master clock
sclk - continuous serial clock	bclk - bit clock
ws - word select	frck - frame clock
sd - serial data	sdata - serial data

MB88F33x ‘Indigo2(-x)’ is always clock master and outputs all clock signals.

NOTE This functionality is not implemented in MB88F335 ‘Indigo2-S’.

4.5.2.5.1 Audio Clock Synthesizer

The audio clock synthesizer generates the MCLK which is used to provide the SDATA, BCLK and FRCK at the I2S interface. The synthesizer offers a precise digital regulation, to generate a very low clock jitter at the output. The regulation is triggered with each buffer update, analyzing the buffer level and applies a certain adjustment to the clock when necessary. The amount of adjustment is based on a defined initial step size, which again will be optimized by the algorithm to the defined minimum step size.

4.5.2.5.2 Audio Formats

The implemented audio interface supports the widely used audio standard interface I2S. Other formats (left-justified, right-justified) are also supported.

Important:
The number of bit clocks between two audio frame starts must be constant.
frck polarity = 0 : frame starts at falling edge of FRCK
frck polarity = 1 : frame starts at rising edge of FRCK.

NOTE

The duty cycle of FRCK at I2S-transmitter is always 0.5

4.5.2.5.3 Supported I2S Timings

	mclk/frck	128	192	256	384	512
bclk/frck						
32		Yes	-	Yes	-	Yes
48		-	Yes	-	Yes	-
64		Yes	-	Yes	-	Yes
128 (TDM)		Yes	-	Yes	-	Yes
256 (TDM)		-	-	Yes	-	Yes

4.5.2.6 APIX2 Data Communication

APIX2 offers full duplex, video independent data communication over the high speed serial link. All data transfers are protected by the integrated AShell protocol, using CRC error detection and optional retransmission mechanisms.

4.5.2.6.1 Overview

APIX2 Automotive Shell (hereinafter referred to as AShell2) is Inova Semiconductor's hardware implementation of an abstraction layer between the APIX2 Physical Layer (APIX2 PHY) and its application. An application, AShell2 and the APIX2 PHY form a stacked bidirectional wire-based communication system. As part of this communication stack, AShell2 provides two interfaces for the exchange of messages from local to a remote island or vice versa. AShell2 hides implementation details and specific features of APIX PHY and provides an appropriate interface to the embedding application of the APIX2 serial link. This encapsulation provides for higher levels of abstraction and supports a functional layered architecture of the communication system. A pixel (video) data stream and an I2S (audio) data stream of an application to be transmitted from TX to RX device is bypassed and thus not affected by the AShell2 layer.

As with most layered communication architectures, a collection of related functions is referred to as layer that provides services to the layer above and receives services from the layer below. Each layer has interfaces to the layers above and below. In general, each layer communicates by the means of a protocol with its counterpart on the remote island.

4.5.2.6.2 AShell2 Services

Apart from the error control functions added to the APIX2 communication stack, the AShell2 is more or less a wrapper layer. The AShell2 provides the following services:

- Transmission of application data ensuring data integrity
- Reception of application data ensuring data integrity
- Supply of information about transmission link status as well as simple error statistics

4.5.2.6.3 AShell2 Functions

To provide the aforementioned services, AShell2 implements the following functions:

1. Transaction framing and de-framing
   Payload data sinked to or sourced by an application is compiled to or extracted from PDUs (transactions) exchanged as protocol entities between local and remote AShell.
2. Data exchange through APIX2 PHY
   AShell2 handles various interface signals from APIX2 PHY such that data is sent and received via APIX2 PHY.
3. Data integrity control
   AShell2 implements a CRC-12 polynomial to detect most transmission errors. Only valid transactions are offered to the application. The transmit path of the AShell2 generates and the receive path checks the CRC sum that is part of the PDU exchanged between the AShells of both communicating islands.
4. Error control (optional)
   AShell2 implements an automatic retransmission protocol to manage bit errors occurring during serial transmission over the wire-based link. As long as the bit errors do not affect synchronization at different levels, communication is kept alive without any intervention by the application. This function can be disabled to support different requirements of the application or software based implementations of AShell2 services.

5. Flow control
   AShell2 implements a freeze function to stop the transmitting device sending further payload while the buffers of the receiving device are full.
6. Status report
   Information about the status of APIX2 PHY and the serial transmission link as well as simple error statistics are collected and presented to the application at the interface.
7. Convenient wrapping of the APIX PHY interface
   AShell2 provides an appropriate broker between APIX2 Physical Link Layer and the respective application. AShell2 thus hides implementation details and special features of APIX2 PHY.

The AShell must be enabled to get operational and to provide the services.

4.5.2.6.4 AShell2 Data Interface

All APIX2 data communications but GPIO are processed via the AShell2. These includes the following:

- Media independent interface (MII)
- AShell2 generic data (64 Bit parallel)
- AShell2 support data
- Remote resource access (read/write)

In case, more than one type of outbound data is pending at the AShell2, a priority control is implemented.

Table 4-2: Priorities

Priority	Priority Group	Type of outbound data
0 (highest)	0	MII
1	0	AShell2 generic data
2	1	AShell2 support data
3	1	remote read response
4	1	remote resource access
5 (lowest)	1	remote command

High prioritized data can displace lower prioritized data. It is possible, that priority group 0 data (MII, AShell2 generic data) uses all available AShell2 bandwidth, so no priority group 1 data (e.g. AShell2 support data or remote resources access) could be transferred to the remote side.

Therefore, the AShell outbound path can be configured to portion a guaranteed minimum AShell2 bandwidth for priority group 1.

4.5.2.6.5 AShell2 Freeze

In case of application cannot receive further data from AShell, a freeze signal is available. As long as freeze is active, payload transmission in both directions (inbound and outbound) stops. The AShell2 freeze doesn’t cause data loss.

As long as pending data is not captured with its strobe signal and further data is pending at internal buffer stages, an automatic freeze is signalized to the local and remote AShell.

Caution: A disabled automatic freeze can cause data loss

Important:
AShell2 freeze (manual and automatic) is only functional if ARQ is enabled!

4.5.2.6.6 AShell2 Error Control

Mode	Description
ARQ ON	Automatic retransmission on transmission errors without intervention of the application layer Indication of transmission errors at the inbound interface,
ARQ OFF with acknowldgement	No automatic retransmission Indication of transmission errors at the inbound and outbound interface Retransmission possible with intervention of the application layer Easy implementation of a ‘stop and wait’ protocol
ARQ OFF without acknowldgement	No automatic retransmission Indication of transmission errors at the inbound interface Retransmission possible with intervention of the application layer

4.5.2.6.7 ARQ ON, Automatic Retransmission

AShell2 implements a NACK-based protocol, meaning that the protocol only intervenes if there is a problem at the receiving side. In case of transmission errors detected through unmatched CRC sums or an AShell2 freeze requested by the application, all transactions of a certain sliding window with variable size including the affected transaction(s) are resent without any intervention by the application. In any case, the order of transactions accepted for transmission is maintained.

To use the protocol, ARQ has to be enabled on TX and RX side.

Important:

Transactions which have to be resent are stored in a buffer of a fixed depth at the transmit path. AShell2 can store a history of the last 10 payload transactions. When a new payload transaction is stored in that buffer, the entry with the longest history will be dropped and is no longer available. The AShell2 protocol initiates retransmission of stored payload transactions, when a NACK was received from the remote island. To ensure that payload transactions, which have to be resent, are not dropped before a NACK can be received, a maximum round trip delay (RTD) must not be exceeded. The maximum RTD is maintained, if the bandwidth of the transmitting direction is not more than four times higher than the bandwidth of the receiving direction.

Because there are bandwidth configurations that exceed the quotient, an AShell2 bandwidth divider is implemented to slow the transmit path (outbound) down. The higher bandwidth is always the dividend, the lower bandwidth is always the divisor.

Example:

Direction	Downstream	Upstream
link rate	3000 MBit/s	62.5 MBit/s
dchannel_bw_cfg	00001	/
GPIO	no	no
I2S	no	/
Resulting AShell2 gross bandwidth	424 MBit/s	55 MBit/s

424 MBit/s / 55 MBit/s = 7.7 $\rightarrow$ quotient exceeds 4

divide downstream (TX) AShell2 bandwidth by 2 : 212 MBit/s / 55 MBit/s = 3.85 $\rightarrow$ quotient falls below 4

The bandwidth divider can be 1 (no division), 2, 4 or 8.

NOTEThe AShell2 bandwidth divider does not affect the resulting video, audio or GPIO bandwidth of the APIX2 link.

4.5.2.6.8 ARQ OFF, With Acknowledgement

In case AShell2 ARQ needs to be disabled, AShell2 implements flags for six different channels to acknowledge the receipt of AShell2 transactions. With the help of these flags, an easy implementation of a Stop and Wait ARQ protocol is possible.

Channel	Dedicated TX device interfaces
1	AShell2 generic data
2	AShell2 support data
3	I2C job spec or I2C job cancel *
4	I2C read data or I2C status *
5	remote resource access or remote command
6	remote read response
* Note: I2C is not implemented in the MB88F33x 'Indigo2(-x)'	

Each acknowledgement consists of the two flags *data delivered* and *data lost*.

If *data delivered* is active, the last transaction was successfully transmitted by the AShell and the pending data was strobed at the remote side.

If *data lost* is active, the last transaction was NOT successfully transmitted because of a CRC error.

If *data delivered* and *data lost* are both active, a fatal error occurred because the remote APIX2 device was soft reset or hard reset. In that case the delivery status is unknown.

AShell2 Status Signals

AShell2 provides information about the status of APIX PHY and transmission link as well as simple error statistics. These are presented at the APIX2 status interface.

AShell2 Operational

The operational status signalizes, that transaction alignment between local and remote AShell2 is established and local AShell2 is ready to accept transactions from application and to deliver received transactions. AShell2 operational becomes low after fatal error, AShell2 restart (local or remote) or APIX2 reset.

There are two different AShell2 modes, which influence behaviour of AShell2 operational after a detected APIX2 link error.

- Link error tolerance is set to 0

An APIX2 link error (falling edge of APIX2 frame aligned) results in a fatal error and hence in an AShell2 operational loss.

- Link error tolerance is set to 1

An APIX2 link error doesn't result in a fatal error and hence not in an AShell2 operational loss. But if the link error was caused by a reset of the remote APIX2 device, the AShell will lose operational after the APIX2 link is established and AShell2 is transaction aligned again.

4.5.2.6.9 AShell2 Ticket Counter

The ticket counter presents the current difference between transmitted and received transactions with application payload at the transmit path. If an error occurs, the application can ascertain a loss of payload transactions.

The outbound ticket counter counts all transmitted transactions with AShell2 generic data and Nibble data. The inbound ticket counter counts all received transactions with AShell2 generic data and Nibble data. The remote AShell2 sends the inbound ticket counter back to the local AShell2 via its reverse channel with every control transaction.

Control transactions will be sent, if no payload is transmitted (realignment or idle state). The difference between the remote inbound ticket counter and the local outbound ticket counter is shown at the status interface.

NOTE The ticket counter system can only be used, if AShell2 ARQ is enabled.

If the AShell2 stops payload transmission, because of a fatal error or an unknown reason, the ticket counter system helps to detect lost payload transactions. In such an error scenario, the AShell2 has to be realigned. After a fatal error, which is shown at the status interface, the AShell2 starts an automatic realignment. A finished alignment is indicated by the operational status.

After that, the ticket counter difference can be read. If the ticket counter difference is *n* and the ticket counter difference is valid, the last *n* transactions have not been received at the remote AShell2. A retransmission of these lost payloads can be executed by the application after AShell2 was realigned. In case the fatal error was caused by a reset of the remote APIX2 device, the ticket counter difference is indicated as invalid.

Before retransmission of the lost payload transactions or transmission of new payload, the ticket counter difference must be cleared to ensure a valid value after a new fatal error.

The ticket counter difference can be reset to 0 by activating the respective clear signal.

4.5.2.6.10 AShell2 Unidirectional Mode

If no APIX2 upstream link (RX -> TX) is used, the AShell2 can operate unidirectional. The unidirectional mode is limited to downstream direction (TX -> RX) only, it cannot be used in upstream direction.

The AShell2 Functions are limited, so no error control and no flow control is available.

- No automatic retransmission
- No delivery status
- No ticket counter
- No AShell2 freeze

All other AShell2 Functions are still provided.

Important: AShell2 unidirectional mode cannot be utilized with older revisions of APIX2.

4.5.2.7 APIX2 MII

The Media Independent Interface (MII) enables the application to use the APIX2 as Ethernet PHY, directly connecting the device to a Media Access Controller (MAC). The MII has been implemented according to the IEEE Standard 802.3-2008, Section 2 offering up to 100MBit/s full-duplex Ethernet functionality.

NOTE This functionality is not implemented in MB88F335 ‘Indigo2-S’

The APIX2 MII does not offer all interfaces described by the IEEE standard. The following interface functionality is different to the standard.

Table 4-4: Interface functionality

Functionality	Description as defined in EEE Standard 802.3-2008, Section 2	APIX2 support
TX_ER	Transmit coding error: The PHY shall emit one or more symbols that are not part of valid data or delimiter set somewhere in the frame being transmitted.	Not supported

Table 4-4: Interface functionality (Continued)

Functionality	Description as defined in EEE Standard 802.3-2008, Section 2	APIX2 support
CRS	Carrier sense: CRS shall be asserted by the PHY when either the transmit or receive medium is non idle. CRS shall be de-asserted by the PHY when both the transmit and receive media are idle. The CRS signal is unspecified for duplex mode.	The APIX2 device operates in full duplex mode over its downstream and upstream links, hence CRS and COL are not supported. If the connected MAC utilizes half duplex operation its CRS and COL can be connected to RX_DV, to provide similar signaling behavior. In case a 2nd APIX2 RX is connected (daisy-chain), the interface MII_COL indicates if the other RX is currently sending. In this case, CRS and COL should be connected to MII_COL. Due to this limitation of the APIX2 interface, the MB88F33x 'Indigo2(-x)' does not support half duplex operation, neither for internal Embedded Ethernet, nor for the MII interface to an external MAC.
COL	Collision detected: COL shall be asserted by the PHY upon detection of a collision on the medium, and shall remain asserted while the collision condition persists. The COL signal is unspecified for duplex mode.	

The MII communication is not protected by the automatic retransmission mechanism of the AShell2, even if AShell ARQ is enabled. The protection of the MII communication must be guaranteed by the upper MAC layer which is not part of the APIX2 IP core.

The other AShell2 interfaces (generic data, remote access) are still operable and protected by AShell2 error control functions.

Important:
It must be ensured that the AShell is configured to get operational and to provide enough AShell bandwidth for MII.
While the AShell2 operational status signalizes, that transaction alignment between local and remote AShell2 is not established, the MII outbound interface is not usable.

MII can be used in APIX2 daisy chain mode.

NOTE MII with ARQ enabled is only usable with APIX2 IP revision 3 and future APIX2 devices.
AShell broadcast mode in daisy-chain configuration is not allowed with enabled ARQ (rule applies to all APIX2 versions).
If an earlier version of APIX2 IP or an INAP375 device including INAP375 revision 3 is part of an APIX2 communication system, ARQ must be disabled, and AShell broadcast mode must be enabled for daisy-chain configuration.

Because two APIX2 RX devices communicate with one APIX2 TX device, that implements one MII interface, the MII packet streams are scheduled at the TX upstream data path. The scheduler has two operating modes:

- **'First come First Serve'** service discipline of incoming MAC packets issued by two APIX2 RX devices
- **'Fair scheduling'** discipline (MAC packets from both APIX2 RX devices can displace each other in a fairly manner)

4.5.2.7.1 AShell2 Bandwidth

The required AShell gross bandwidth depends on the MII clock.

Downstream

$$\text{AShell grossbandwidth} = \frac{40}{7} \cdot \text{mii clk}$$

Upstream

$$\text{AShell grossbandwidth} = \frac{16}{3} \cdot \text{mii clk}$$

4.5.2.7.2 MII Multiplexer

The MB88F33x ‘Indigo2(-x)’ has implemented a static 3-way MII-Multiplexer This multiplexer has to be set at device configuration and cannot be re-configurated during operation.

The multiplexer allows to connect the MII-Interface, coming from the APIX Interface, either to the internal Embedded Ethernet unit or to the MII pins for connecting an external Ethernet MAC. The third option allows to connect an external Ethernet PHY to the internal Embedded Ethernet unit.

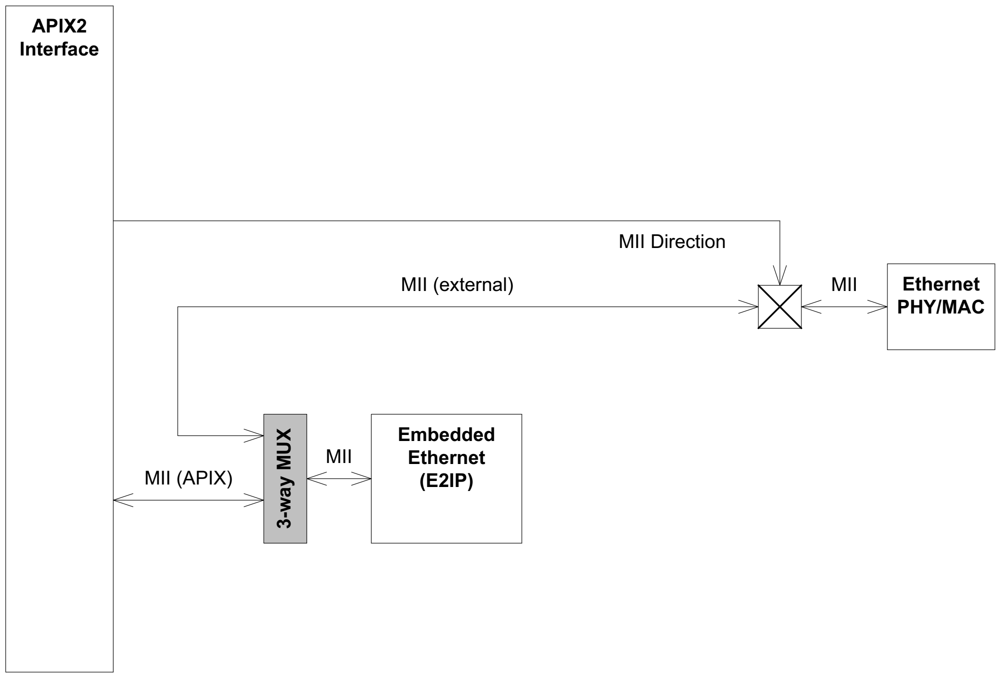


Figure 4-3: MII Multiplexer

4.5.2.8 AShell2 Generic Data

The APIX2 AShell offers to transmit generic 64 bit parallel data with an appending target ID.

APIX2 comes with two AShell data paths. Therefore the AShells are enumerated, and hereinafter referred to as AShell2_0 and AShell2_1.

APIX2 TX has the two instances AShell2_0 and AShell2_1. APIX2 RX has one AShell, named AShell2_0. The AShell of the second RX in daisy chain mode belongs to AShell2_1.

4.5.2.8.1 AShell2 Target ID

The AShell2 target id controls the data flow of application data.

The target id will be interpreted by the receiving AShell2. In general AShell2_0 extracts only payload with target id 0 and AShell2_1 extracts only payload with target id 1, but per configuration each AShell can be set to broadcast mode. In this mode the AShell2 extracts payload with every target id.

In conclusion in general the outbound target ids must be hard wired (see below), because they are used by the AShells to identify their belonging payload, and the output ports *ashell2_*_inbound_data_target_id_o* have no relevance.

TX *ashell2_0_outbound_data_target_id_i* = 0

TX *ashell2_1_outbound_data_target_id_i* = 1

RX0 *ashell2_0_outbound_data_target_id_i* = 0

RX1 *ashell2_0_outbound_data_target_id_i* = 0

But if only one APIX2 RX device is connected to a APIX2 TX device (no daisy-chain mode) the AShell2 target ids can be utilized to realize effortlessly two protected downstream data channels and two protected upstream data channels through one AShell2 link by enabling the broadcast mode. Then the AShell2 target ids are used to display the data channel the payload is belonging to.

Operating in daisy-chain mode the broadcast mode for AShell2 payload is not supported (only for MII communication).

4.5.2.8.2 Flow Control

APIX2 implements flow control protocol for AShell2 generic data. This feature supports a back pressure to stall the outbound data path of AShell2 generic data while the remote inbound data path of AShell2 generic data is not able to receive further data. This flow control does not intervene to the AShell2 protocol layer.

The flow control protocol signalizes a ‘stall assert’ and a ‘stall release’ to the remote site via AShell2. ‘Stall assert’ is always signalized as soon as possible, and hence it defers other AShell2 outbound data. ‘Stall release’ signalization can be configured to operate in one of two modes.

1. It can be signalized as soon as possible, just as ‘stall assert’.
2. It can be signalized only if unused AShell2 bandwidth is available, without deferring other AShell2 outbound data.

4.5.2.9 APIX2 GPIO

In addition to the protected data communication, in APIX2 mode the APIX2 IP allows to sample 2 GPIO inputs and provides up to 4 GPIO outputs (2 per source device). The GPIO inputs are asynchronously sampled and directly transferred through up to two 1bit width GPIO channels to receiver devices with lowest delay. The GPIO outputs reflect the signals sampled at the source devices.

NOTE The two GPIO inputs are not sampled and transmitted at the same time but one after another.
Thus the signals have an offset to each other when they appear at the GPIO outputs of the receiver.

4.5.2.9.1 GPIO Downstream

The sampling speed of the GPIO downstream inputs depends on following aspects:

- APIX2 downstream link rate (3GBit/s, 1GBit/s or 500MBit/s)
- Number of enabled GPIO downstream channels
- GPIO bandwidth mode (low speed or high speed)
- GPIO downstream bandwidth divider (allows to divide the bandwidth by half)

Downstream link rate (Mbit/s)	Number of GPIO channels	Halved GPIO downstream bandwidth	Sampling frequency in low speed mode (MHz)	Sampling frequency in high speed mode (MHz)
3000	1	full	6.696	26.786
3000	1	halved	3.348	13.393
3000	2	full	3.348	13.393
3000	2	halved	unsupported	6.696
1000 or 500	1	full	4.464	17.857
1000 or 500	1	halved	2.232	8.929

Downstream link rate (Mbit/s)	Number of GPIO channels	Halved GPIO downstream bandwidth	Sampling frequency in low speed mode (MHz)	Sampling frequency in high speed mode (MHz)
1000 or 500	2	full	2.232	8.929
1000 or 500	2	halved	1.116	4.464

4.5.2.9.2 GPIO Upstream

The sampling speed of the GPIO upstream inputs depends on following aspects:

- APIX2 upstream link rate (187.5MBit/s or 62.5MBit/s)
- Number of APIX2 receivers connected to the APIX2 TX (daisy-chain)
- Number of enabled GPIO upstream channels
- GPIO bandwidth mode (low speed or high speed)

Upstream link rate (Mbit/s)	Number of connected RX devices	Number of GPIO channels	Sampling frequency in low speed mode (MHz)	Sampling frequency in high speed mode (MHz)
187.5	1	1	3.35	13.392
187.5	1	2	3.35	13.392
187.5	2 (daisy-chain)	1	3.35	6.696
187.5	2 (daisy-chain)	2	3.35	6.696
62.5	1	1	1.117	4.464
62.5	1	2	1.117	4.464
62.5	2 (daisy-chain)	1	1.117	2.232
62.5	2 (daisy-chain)	2	1.117	2.232

4.5.3 APIX2RX Link Register Overview

The APIX2 RX Link Layer is configured by applying the configuration data together with an config_valid at the configuration interface.

The configuration data register APIX_CFG_* may only be changed if APIX_CFG_VALID.config_valid is inactive.

The APIX2 RX Link Layer is disabled and a soft reset is issued to the block immediately after deasserting APIX_CFG_VALID.config_valid.

Table 4-5: Registers Overview

Base Address(es)	Instance no 0: BASEADDR="00021000"	
Absolute Address	**Register Name**	**Register Description**
BASEADDR + 0x0000	APIX_CFG_0	Inova Config Byte
BASEADDR + 0x0004	APIX_CFG_1	Inova Config Byte
BASEADDR + 0x0008	APIX_CFG_2	Inova Config Byte
BASEADDR + 0x000C	APIX_CFG_3	Inova Config Byte
BASEADDR + 0x0010	APIX_CFG_4	Inova Config Byte
BASEADDR + 0x0014	APIX_CFG_5	Inova Config Byte
BASEADDR + 0x0018	APIX_CFG_6	Inova Config Byte
BASEADDR + 0x001C	Reserved	Do not modify
BASEADDR + 0x0020	APIX_CFG_8	Inova Config Byte
BASEADDR + 0x0024	APIX_PARAM_3	Config Parameter
BASEADDR + 0x0028	Reserved	Do not modify
BASEADDR + 0x0100	APIX_CFG_VALID	Inova Config Signals, resets APIX IP core
BASEADDR + 0x0104	APIX_CFG_MODE	Inova Config Signals, general configuration register
BASEADDR + 0x0108	APIX_STAT_CTRL_0	Inova Status Control Signals
BASEADDR + 0x010C	APIX_STAT_CTRL_1	Inova Counter Clear Signals
BASEADDR + 0x0110	APIX_STAT_TIMEOUT	Inova frame alignment status
BASEADDR + 0x0114	APIX_STAT_0	Inova Status Bytes
BASEADDR + 0x0118	APIX_STAT_1	Inova Status Bytes
BASEADDR + 0x011C	APIX_STAT_2	Inova Status Bytes
BASEADDR + 0x0120	APIX_STAT_4	Inova Status Bytes
BASEADDR + 0x0124	TST_CFG_0	Inova Test Config Bytes
BASEADDR + 0x0128	TST_STAT	Inova Test Status Bytes
BASEADDR + 0x012C	Reserved	Do not modify
BASEADDR + 0x0130	Reserved	Do not modify
BASEADDR + 0x0134	Reserved	Do not modify
BASEADDR + 0x0138	Reserved	Do not modify
BASEADDR + 0x013C	DBG_STAT_4	Inova Debug Data from IP
BASEADDR + 0x0140	DBG_STAT_5	Inova Debug Data from IP,
BASEADDR + 0x0144	APIX_REM_CMD_EN	APIX Remote Command Inbound Enable
BASEADDR + 0x0148	APIX_REM_CMD_REQ	APIX Remote Command Request
BASEADDR + 0x014C	APIX_REM_CMD_STAT	APIX Remote Command Status
BASEADDR + 0x0150	Reserved	Do not modify
BASEADDR + 0x0154	INT_STAT_LINK	Interrupt Status Link
BASEADDR + 0x0158	INT_STAT_ASHELL	Interrupt Status Ashell
BASEADDR + 0x015C	INT_STAT_PIX	Interrupt Status Ashell
BASEADDR + 0x0160	INT_EN_LINK	Interrupt Enable Link
BASEADDR + 0x0164	INT_EN_ASHELL	Interrupt Enable Ashell
BASEADDR + 0x0168	INT_EN_PIX	Interrupt Enable Ashell
BASEADDR + 0x016C	INT_SET_LINK	Interrupt Set Link
BASEADDR + 0x0170	INT_SET_ASHELL	Interrupt Set Ashell

Table 4-5: Registers Overview

Base Address(es)	Instance no 0: BASEADDR="00021000"	
Absolute Address	**Register Name**	**Register Description**
BASEADDR + 0x0174	INT_SET_PIX	Interrupt Set Pixel
BASEADDR + 0x0178	Reserved	Do not modify

4.5.4 APIX2 HDCP Register Overview

NOTE Please be aware, that this section is only valid for MB88F334 ’Indigo2’ and not functional for MB88F33x ’Indigo2-N’and MB88F33x ’Indigo2-S’**.**

Table 4-6: Registers Overview

Base Address(es)	**Instance no 0: BASEADDR="00025000"**	
Absolute Address	**Register Name**	**Register Description**
BASEADDR + 0x0000	CFG_HDCP	HDCP configuration signals
BASEADDR + 0x0004	Reserved	Do not modify
BASEADDR + 0x0008	CTRL_HDCP_0	HDCP control signals
BASEADDR + 0x000C	STATUS_HDCP_0	HDCP status signals
BASEADDR + 0x0010	STATUS_HDCP_1	HDCP status signals
BASEADDR + 0x0014	STATUS_HDCP_2	HDCP status signals
BASEADDR + 0x0018	STATUS_HDCP_3	HDCP status signals
BASEADDR + 0x001C	STATUS_HDCP_4	HDCP status signals
BASEADDR + 0x0020	STATUS_HDCP_5	HDCP status signals
BASEADDR + 0x0024	STATUS_HDCP_6	HDCP status signals
BASEADDR + 0x0028	STATUS_HDCP_7	HDCP status signals
BASEADDR + 0x002C	STATUS_HDCP_8	HDCP status signals
BASEADDR + 0x0030	STATUS_HDCP_9	HDCP status signals
BASEADDR + 0x0034	STATUS_HDCP_10	HDCP status signals
BASEADDR + 0x0038	STATUS_HDCP_11	HDCP status signals
BASEADDR + 0x003C	STATUS_HDCP_12	HDCP status signals
BASEADDR + 0x0040	STATUS_HDCP_13	HDCP status signals
BASEADDR + 0x0044	STATUS_HDCP_14	HDCP status signals
BASEADDR + 0x0048	STATUS_HDCP_15	HDCP status signals
BASEADDR + 0x004C	Reserved	Do not modify
BASEADDR + 0x0050	INT_STAT_HDCP	Interrupt Status HDCP
BASEADDR + 0x0054	INT_EN_HDCP	Interrupt Enable HDCP
BASEADDR + 0x0058	INT_SET_HDCP	Interrupt Set HDCP

4.6 AShell Remote Handler

The remote handler unwraps transactions received from external host CPU via the APIX automotive shell (AShell) to the hardware protocol of the AHB system bus and vice versa. Additionally, this module has the task of an intelligent interrupt controller. It builds interrupt messages and sends them as event messages to the external host CPU. If an application requires to read some addresses from the local register space, every time a specific interrupt is received the remote handler autonomously collects the data to be read (later) and sends this interrupt message right along with the data. In this way, the required communication overhead is reduced.

4.6.1 Features of the AShell Remote Handler

- Interfacing to APIX automotive shell
  - Apix1: 56-bit AShell generic data interface
  - Apix2: 64-bit AShell generic data interface
  - Apix2: Message filtering based on AShell target identifier
  - Apix2: Message forwarding based on AShell target identifier
  - Apix2: Support acknowledge-of-receipt-mode (programmable)
  - Apix1/2: AShell flow control based on fill level of internal receive and transmit buffers
  - Apix1/2: Time-out detection for AShell outbound (programmable)
- Messages
  - Common 56-bit message format for APIX1 and APIX2
  - Downstream:
    read request message from host
    write request message from host
    lock/unlock message from host
  - Upstream:
    read response message to host
    event message to host
    push message to host (single message or block-transfer mode)
- Message buffering for downstream and upstream messages
- Single read/write AHB-master architecture
  - Keep the transaction ordering given by the host
  - Arbitrate between remote (read, write message) and local (event, push message) requests
  - Lock mechanism to suppress AHB-write request messages
  - Consistency check before starting any AHB transaction
- Up to 209 event sources
  - bit position of the interrupts defined by interrupt controller
  - fixed priority based on event index (event index equals the bit position)
    lowest event index highest priority (refer interrupt controller)
  - configurable AHB-read
- Automatic clear of event message request flag (programmable)
- FIFO level observation:
  - Configurable FIFO threshold for receive and transmit buffer
- Error detection in case FIFO overflow

4.6.2 Block Diagram

The following figure shows the remote handler within its environment. This figures shall provide a brief overview of the data paths within the remote handler.

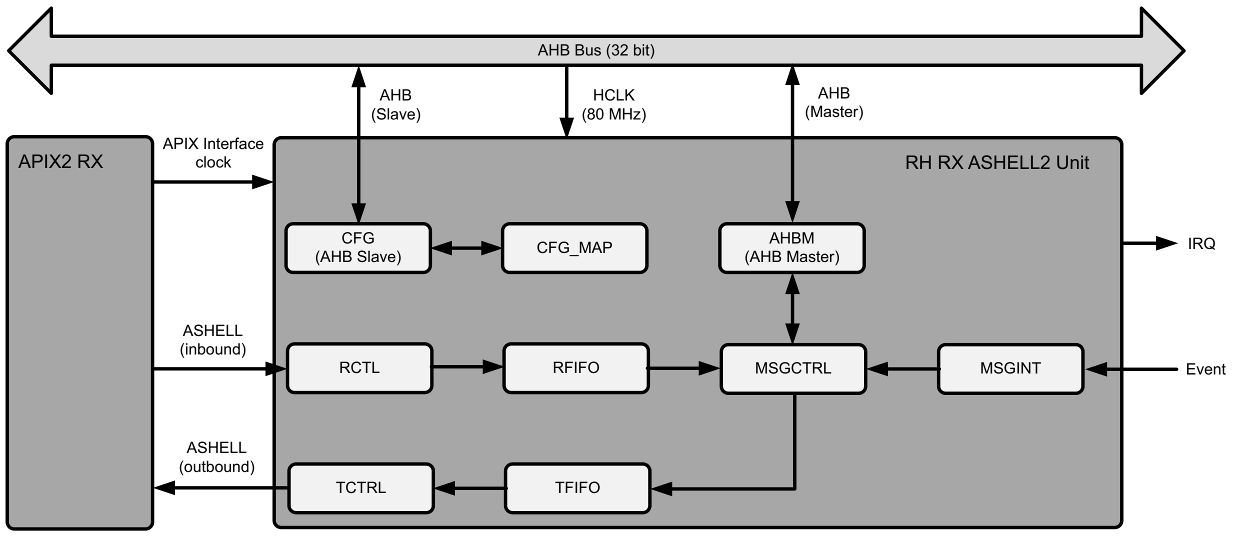


Figure 4-4: Block diagram

4.6.3 AShell Remote Handler Operation

Please refer to Register Description document for additional information.

4.6.3.1 AShell Messages

The remote handler defines following message format.

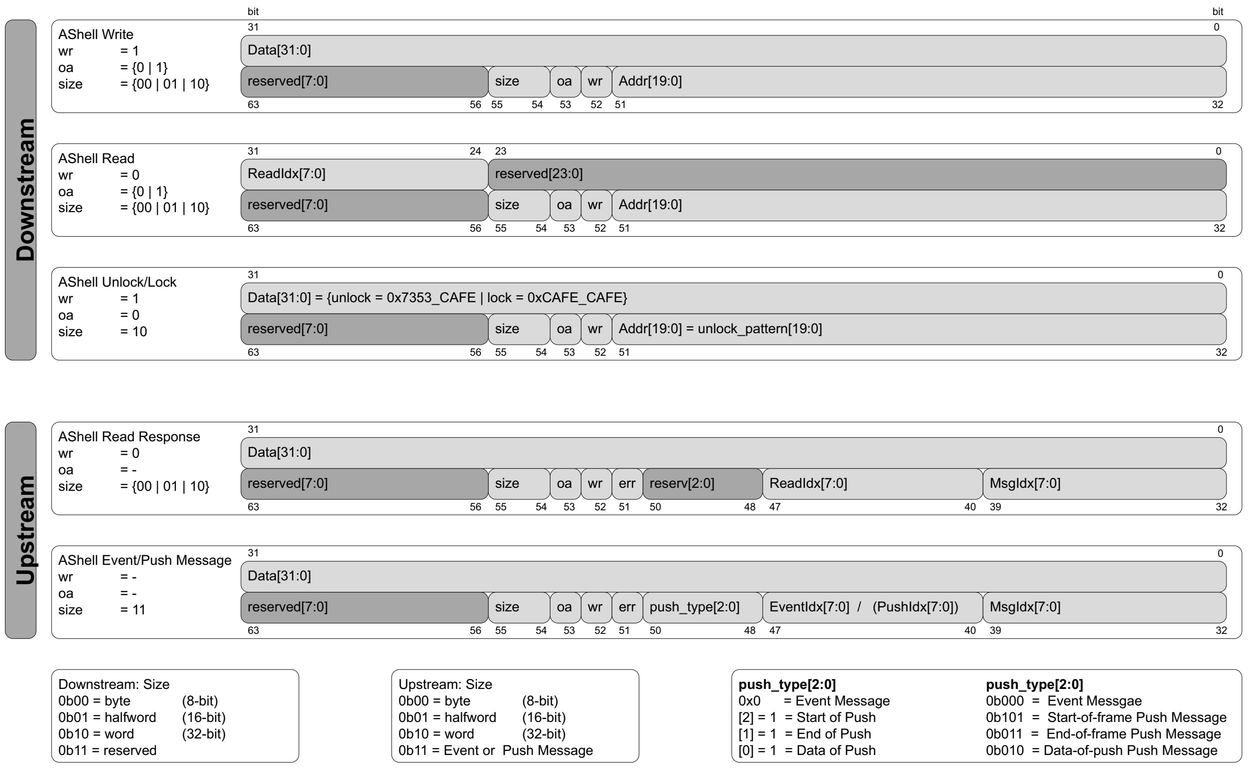


Figure 4-5: AShell messages overview

Type	Message Name	Description
Message	Ashell_DownstreamWrite	AShell Downstream Write Transaction
Message	Ashell_DownstreamRead	AShell Downstream Read Transaction
Message	Ashell_DownstreamUnlock	AShell Downstream Unlock Write
Message	Ashell_DownstreamLock	AShell Downstream Lock Write
Message	Ashell_UpstreamReadResponse	AShell Upstream Read Response
Message	Ashell_UpstreamEvent	AShell Upstream Event Message
Message	Ashell_UpstreamPush	AShell Upstream Push Message

The message descriptions in the following sections use the format shown below to describe each bit field of a message.

Message	**<Message Name>**			
Bit Number/ Range	**Field**	**Length**	**Value**	**Description**
<bit> or <range>	<field name>	<length>	<value>	<description>

NOTE Please refer to register description for the corresponding offset address registers: BASE_ADDR_WRITE and BASE_ADDR_READ.

AShell Downstream Write Transaction

Message	**Ashell_DownstreamWrite**			
Bit Number/ Range	**Field**	**Length**	**Value**	**Description**
63:56	-	8	-	Reserved
55:54	size[1..0]	2	00b=byte 01b=halfword (2-byte) 10b=word(4-byte) 11b=reserved	Transfer Size
53	oa	1	0b=absolute address 1b=offset address	Offset Address Enable, defines to use field Addr as an offset address
52	wr	1	1b	Write-not-Read
51:32	Addr[19..0]	20	Addr	Write Address
31:0	Data[31..0]	32	Data	Write Data If size = b00 then Data[31:24] is used If size = b01 then Data[31:16] is used If size = b10 then Data[31: 0] is used

AShell Downstream Read Transaction

Message	**Ashell_DownstreamRead**			
Bit Number/ Range	**Field**	**Length**	**Value**	**Description**
63:56	-	8	-	Reserved
55:54	size[1..0]	2	00b=byte 01b=halfword(2-byte) 10b=word(4-byte) 11b=reserved	Transfer Size
53	oa	1	0b=absolute address 1b=offset address	Offset Address Enable, defines to use field Addr as an offset address
52	wr	1	0b	Write-not-Read
51:32	Addr[19..0]	20	Addr	Read Address
31:24	RdIdx[7..0]	8	RdIdx	Read Index
23:0	-	24	-	Reserved

AShell Downstream Unlock Write

Message	**Ashell_DownstreamUnlock**			
Bit Number/ Range	**Field**	**Length**	**Value**	**Description**
63:56	-	8	-	Reserved
55:54	size[1..0]	2	10b	Transfer Size, Tsize=10b=word(4-byte)
53	oa	1	0	Offset Address Enable
52	wr	1	1b	Write-not-Read
51:32	AddrKey[19..0]	20	f_ffffh	Write Address Pattern (defined by AHBM_LOCK.unlock_pattern).[1)]
31:0	DataKey[31..0]	32	7353_CAFEh	Write Data Pattern (fix pattern)

[1)] Note: The AddrKey defines an address to which the DataKey has to be applied to.
Example for APIX access (for embedded Ethernet please use the corresponding addresses):
- The field AddrKey is set to 0x11004h. This will set the AHBM_LOCK Register to the same address as the MB88F332 ‘Indigo’.
- Unlocking the register will now be done by writing the field Data 7353_CAFEh to the address 0x11004h via APIX
- Reading the UNLOCK status is done via register AHBM_LOCK (2200Ch) (this is different compared to MB88F332 ‘Indigo’)

AShell Downstream Lock Write

Message	**Ashell_DownstreamLock**			
Bit Number/ Range	**Field**	**Length**	**Value**	**Description**
63:56	-	8	-	Reserved
55:54	size[1..0]	2	10b	Transfer Size, Tsize=10b=word(4-byte)
53	oa	1	0	Offset Address Enable
52	wr	1	1b	Write-not-Read
51:32	AddrKey[19..0]	20	f_ffffh	Write Address Pattern (defined by AHBM_LOCK.unlock_pattern) [1)]
31:0	DataKey[31..0]	32	cafe_cafeh	Write Data Pattern (fix pattern)

[1)] Note: Please refer to the same procedure as for AShell Downstream Unlock Write, when you need to apply the lock pattern.

AShell Upstream Read Response

Message	**Ashell_UpstreamReadResponse**			
Bit Number/ Range	**Field**	**Length**	**Value**	**Description**
63:56	-	8	-	Reserved
55:54	size[1..0]	2	00b=byte 01b=halfword(2-byte) 10b=word(4-byte) 11b=reserved	Transfer Size
53	oa	1	-	Offset Address Enable from Read Request
52	wr	1	0b	Read Response Transaction
51	err	1	err	Chip internal bus error
50:48	-	3	-	Reserved
47:40	RdIdx[7..0]	8	RdIdx	Read Index

39:32	MsgIdx[7..0]	8	0x0	Message index: Remote handler will increment this value for each read message of a dedicated target ID (AShell channel)
31:0	Data[31..0]	32	Data	Read Response Data If size = b00 then Data[31:24] is used If size = b01 then Data[31:16] is used If size = b10 then Data[31: 0] is used

AShell Upstream Event Message

Message	**Ashell_UpstreamEvent**			
Bit Number/ Range	**Field**	**Length**	**Value**	**Description**
63:56	-	**8**	-	Reserved
55:54	size[1..0]	**2**	11b=Event Message	Transfer Size
53	oa	**1**	-	Reserved
52	wr	**1**	-	Reserved
51	err	**1**	err	Chip internal bus error
50:48	push_type[2..0]	**3**	000b=Event Message	Push Type (Event Message)
47:40	EventIdx[7..0]	**8**	EventIdx	Event Index
39:32	MsgIdx[7..0]	**8**	0x0	Message index: Remote handler will increment this value for each event message of a dedicated target ID (AShell channel)
31:0	Data[31..0]	**32**	Data	Event Data If EVENT_MSG_TABLE[n].event_msg_size = b00 then Data[31:24] is used If EVENT_MSG_TABLE[n].event_msg_size = b01 then Data[31:16] is used If EVENT_MSG_TABLE[n].event_msg_size = b10 then Data[31: 0] is used n is the Event Index

AShell Upstream Push Message

Message	**Ashell_UpstreamPush**			
Bit Number/ Range	**Field**	**Length**	**Value**	**Description**
63:56	-	8	-	Reserved
55:54	size[1..0]	2	11b=Event Message	Transfer Size
53	oa	1	-	Reserved
52	wr	1	-	Reserved
51	err	1	err	Chip internal bus error
50:48	push_type[2..0]	3	[2]=1b...Start of push [1]=1b...End of push [0]=1b...Data of push	Push Type (Push Message)
47:40	PushIdx[7..0]	8	PushIdx	Push Index
39:32	MsgIdx[7..0]	8	0x0	Message index: remote handler will increment this value for each push message of a dedicated target ID (AShell channel)

Message	**Ashell_UpstreamPush**			
Bit Number/ Range	**Field**	**Length**	**Value**	**Description**
31:0	Data[31..0]	32	Data	Push Data If EVENT_MSG_TABLE[PUSH_SETUP.push_event_idx].event_msg_size = b00 then Data[31:24] is used If EVENT_MSG_TABLE[PUSH_SETUP.push_event_idx].event_msg_size = b01 then Data[31:16] is used If EVENT_MSG_TABLE[PUSH_SETUP.push_event_idx].event_msg_size = b10 then Data[31: 0] is used

4.6.3.1.1 Write Transaction

Before any write access tis possible the remote handler has to be in unlock state. For control flow, please refer to “4.6.4.1 Request Messages”.

4.6.3.1.2 Unlock/Lock Write

The AHB write master of the module is, at reset state, in a locked state. Before any write access to the AHB bus are possible, an “Unlock Write” (see chapter “4.6.3.1 AShell Messages”) must be received.

During operation, it is possible to lock the AHB write master by sending the ”Lock Write” message (see chapter “4.6.3.1 AShell Messages”) or by writing to register field AHBM_Lock.lock of the Remote Handler AHB slave configuration interface. This makes it possible to avoid competing access to resources.

Read access is possible at any time, even if the write master is locked.

For control flow see chapter “4.6.4.1 Request Messages”.

4.6.3.1.3 Read Transaction/Response

A read consists of read transaction which is initiated from the host and a read response coming from Indigo2 (see chapter “4.6.3.1 AShell Messages”). For control flow see chapter “4.6.4.1 Request Messages”.

4.6.3.1.4 Event Message

The AShell Remote Handler can send an interrupt message to the host via the APIX interface. An interrupt message is send for all internally generated interrupt of the AShell remoter handler and for multiple Indigo2 internal interrupts. Interrupt prioritization is handled by the interrupt number. The lowest number has the highest priority. Together with the interrupt message also a payload is send to the host. The content of the payload is read by the AHB read master. The read addresses of this read transaction can be defined by software in the event message table RAM. If the executed AHB read transaction can not be successfully executed, a dedicated message bit indicates the AHB error (for the bit number, please see the frame format definition table chapter “4.6.3.1 AShell Messages”). An automatic clear of the interrupt flag in the Ashell can be executed if enabled. For control flow see chapter “4.6.4.1 Request Messages”.

For the list of event numbers and corresponding interrupt sources, please refer to section “4.6.6 Event Messages”

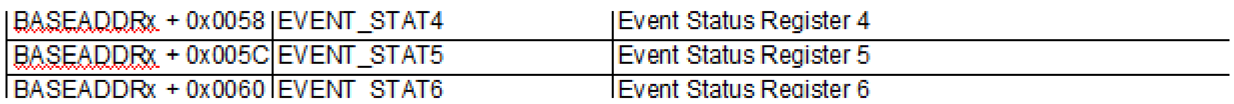
To enable event messages the global event_en bit has to be set in the RH_CTRL register. In addition each individual event has to be enabled and the event message table has to be set up correct (see examples). The same setup has also to be done for events based on interrupts from the remote handler itself. For these internal interrupts also the correct interrupt enable has to be set in the RH_IRQ_EN register.

Example: Setup Event Message for Iris-MVL frame generator synchronization loss

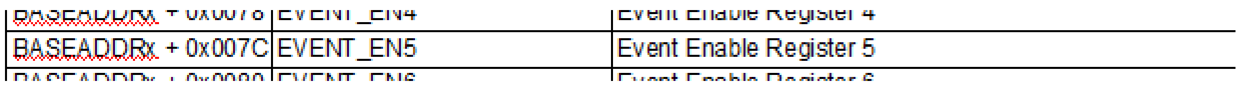
1. In Table 4-7, “Event messages” find the corresponding interrupt and check for the matching event (= 171):

Event	Name	Description
171	IRS_FG_SYNCERR_S	Iris-MVL frame generator synchronization loss (secondary input)

2. In the Register Description document (rd-mb88f334-register-rev0-0x.pdf), Table 4-9, “Register Overview AShell RH” search for the register(s) that handle(s) that event.
   - Register **EVENT_STAT5**, on location “BASEADDRx + 0x005C” at bit 12 (bit position for 171 in a 32-bit register), gives the status of the selected event.
   - Register **EVENT_EN5**, on location “BASEADDRx + 0x007C” at bit 12**,** enables that selected event.

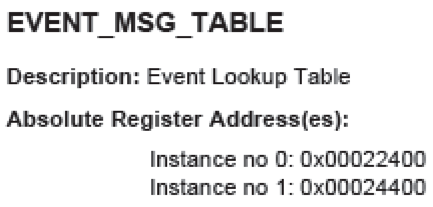


BASEADDRx + 0x0058	EVENT_STAT4	Event Status Register 4
BASEADDRx + 0x005C	EVENT_STAT5	Event Status Register 5
BASEADDRx + 0x0060	EVENT_STAT6	Event Status Register 6



BASEADDRx + 0x0078	EVENT_EN4	Event Enable Register 4
BASEADDRx + 0x007C	EVENT_EN5	Event Enable Register 5
BASEADDRx + 0x0080	EVENT_EN6	Event Enable Register 6

3. The message to be programmed in the selected event has to be stored in the Register **EVENT_MSG_TABLE**, starting at “BASEADDRx + 0x0400”.



EVENT_MSG_TABLE

Description: Event Lookup Table

Absolute Register Address(es):

Instance no 0: 0x00022400
Instance no 1: 0x00024400

4. The location for event 171 in the Register **EVENT_MSG_TABLE** is:
   BASEADDRx + 0x0400 + hex(171 * 4) →
   BASEADDRx + 0x0400 + 2ACh →
   BASEADDRx + 0x06ACh

NOTE The individual register descriptions can be found in the register description manual, chapter “4.6.8 AShell Remote Handler Register Overview”.

Example: Setup for Mailbox event message

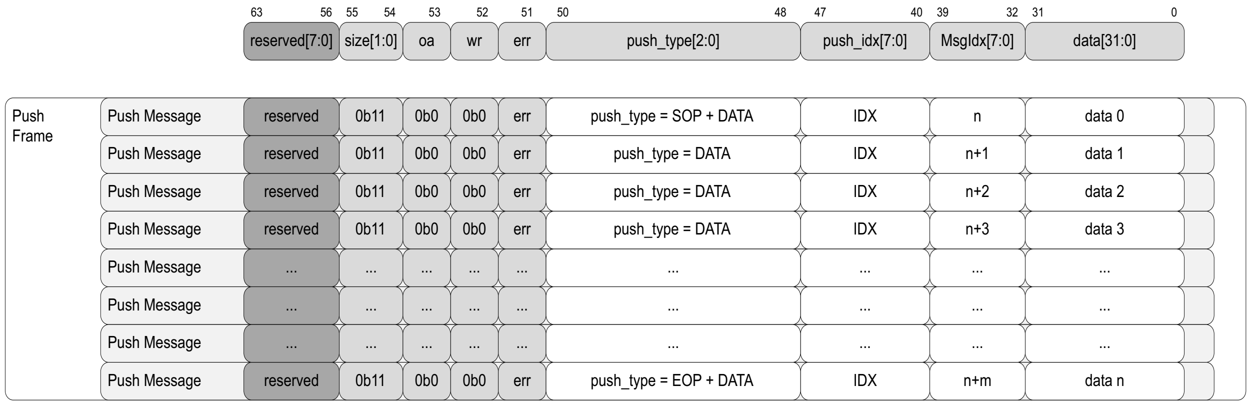
A special setup has to be done for the mailbox event. Writing to the MAILBOX register will trigger the mail_req interrupt. For this the mail_req has to be enabled in the RH_IRQ_EN register. This interrupt is routed to the MAIL_REQ event input. For this MAIL_REQ event the event has to be enabled (EVENT_EN0 bit 15 has to be set to 1) and the event message table has to be setup that it reads the MAILBOX register for the event payload (write into “base_addres + 400h + hex(15 * 4)” the setup for a 32bit access to the address of the MAILBOX register). The setup can be done similar to in the previous example.

4.6.3.1.5 Push Message

Push messages can be combined into push frame. A push frame consists out of at least one push message, but may consists out of several push messages, for which the number of push messages is in general not limited.

Each push frame needs a start-of-frame and a end-of-frame identifier. The start-of-frame and end-of-frame identifier may be part of the same push message. Push messages, which are not the first or last message shall set the data-of-push identifier.

The following pictures shows a push frame which consists out of multiple push messages.



		63–56	55–54	53	52	51	50–48	47–40	39–32	31–0
		reserved[7:0]	size[1:0]	oa	wr	err	push_type[2:0]	push_idx[7:0]	MsgIdx[7:0]	data[31:0]
Push Frame	Push Message	reserved	0b11	0b0	0b0	err	push_type = SOP + DATA	IDX	n	data 0
	Push Message	reserved	0b11	0b0	0b0	err	push_type = DATA	IDX	n+1	data 1
	Push Message	reserved	0b11	0b0	0b0	err	push_type = DATA	IDX	n+2	data 2
	Push Message	reserved	0b11	0b0	0b0	err	push_type = DATA	IDX	n+3	data 3
	Push Message	...	...	...	...	...	...	...	...	...
	Push Message	...	...	...	...	...	...	...	...	...
	Push Message	...	...	...	...	...	...	...	...	...
	Push Message	reserved	0b11	0b0	0b0	err	push_type = EOP + DATA	IDX	n+m	data n

Figure 4-6: Push Frame

A push message can be set up in “Single Mode” or “Block Mode”. For “Block Mode the length is limited to 32 single push messages, whereas in “Single Mode” the length is not limited. But sending a push frame in “Single Mode” requires some more control overhead when generating the frame.

To enable push messages the global push_en bit has to be set in the RH_CTRL register. The push_req interrupt has to be enabled in the RH_IRQ_EN register and the PUSH_REQ event has to be enabled (EVENT_EN0 bit 17 has to be set to 1). The event message table of the PUSH_REQ event has to be set up in a way that it reads from the PUSH_MSG register (write into “base_addres + 400h + hex(17 * 4)” the setup for a 32bit access to the address of the PUSH_MSG register).

For control flow see chapter “4.6.4.3 Push Message”.

Example: Setup for Push frame with 3 push messages (= 3 x 32bit data) in “Single Mode”

1. Set PUSH_TID.push_tid and PUSH_INDEX.push_idx register.
2. Set PUSH_MODE.push_mode = single.
3. Set PUSH_TYPE.push_type = SOP and DATA. (Start of push and data)
4. Write first 32bit data into register PUSH_MSG.
5. Wait for PUSH_ACK interrupt.
6. Set PUSH_TYPE.push_type = DATA. (only data)
7. Write second 32bit data into register PUSH_MSG.
8. Wait for PUSH_ACK interrupt.
9. Set PUSH_TYPE.push_type = EOP and DATA. (End of push and data)

10. Write third 32bit data into register PUSH_MSG.
11. Wait for PUSH_ACK interrupt.

Example: Setup for Push frame with 3 push messages (= 3 x 32bit data) in “Block Mode”

1. Set PUSH_TID.push_tid and PUSH_INDEX.push_idx register.
2. Set PUSH_MODE.push_mode = block.
3. Set PUSH_MODE.push_block_length = 3.
4. Write first 32bit data into register PUSH_MSG.
5. Write second 32bit data into register PUSH_MSG.
6. Write third 32bit data into register PUSH_MSG.
7. Set PUSH_MODE.push_block_start = 1.
8. Wait for PUSH_ACK interrupt.

4.6.3.2 Error Handling

The following scenarios are handled by the remote handler.

- Observation of lock status
  - Incoming write requests are dropped in lock state
  - Interrupt generation
- Observation of receive buffer fill level
  - Configurable threshold and interrupt generation
  - Buffer overflow and interrupt generation
    (drop of incoming message)
- Observation of transmit buffer fill level
  - Configurable threshold and interrupt generation
  - Buffer overflow and interrupt generation
- Observation of an AHB access
  - Check misalignment within read or write request
    prevent the AHB transaction, if the request is misaligned
    Note: For this error the WRERR_ADDR is not updated with the misaligned address.
  - Check AHB response
  - Interrupt generation and store absolute address for debugging
  - In case of read requests the error flag within the read response will be set

For all error cases an interrupt can be enabled (regster RH_IRQ_EN). This interrupt is routed to the event inputs of the remote handler. If the event is setup in the correct way an event message is send via the APIX link to the Host system.

4.6.4 AShell Remote Control Handler Control Flow

4.6.4.1 Request Messages

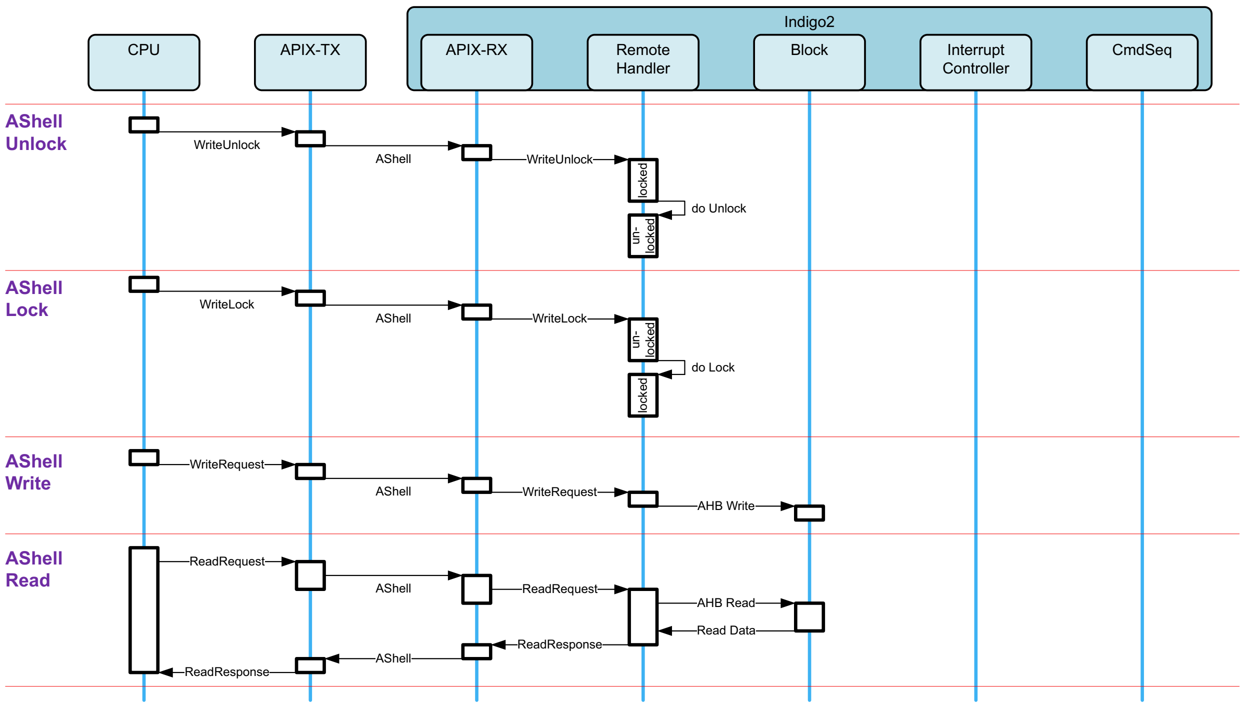


Figure 4-7: Sequence Diagram for Unlock-, Lock, Read-, Write and Read-Response-Messages

4.6.4.2 Event Message

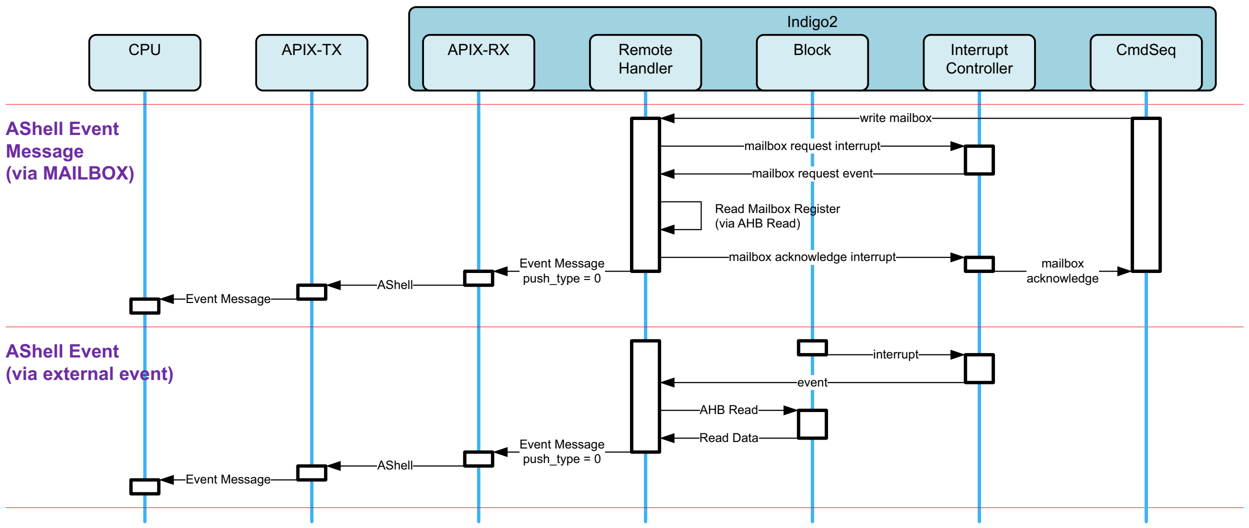

Indigo2
CPU
APIX-TX
APIX-RX
Remote Handler
Block
Interrupt Controller
CmdSeq
AShell Event Message (via MAILBOX)
write mailbox
mailbox request interrupt
mailbox request event
Read Mailbox Register (via AHB Read)
mailbox acknowledge interrupt
mailbox acknowledge
Event Message push_type = 0
AShell
Event Message
AShell Event (via external event)
interrupt
event
AHB Read
Read Data
Event Message push_type = 0
AShell
Event Message

Figure 4-8: Sequence Diagram for Event-Messages

4.6.4.3 Push Message

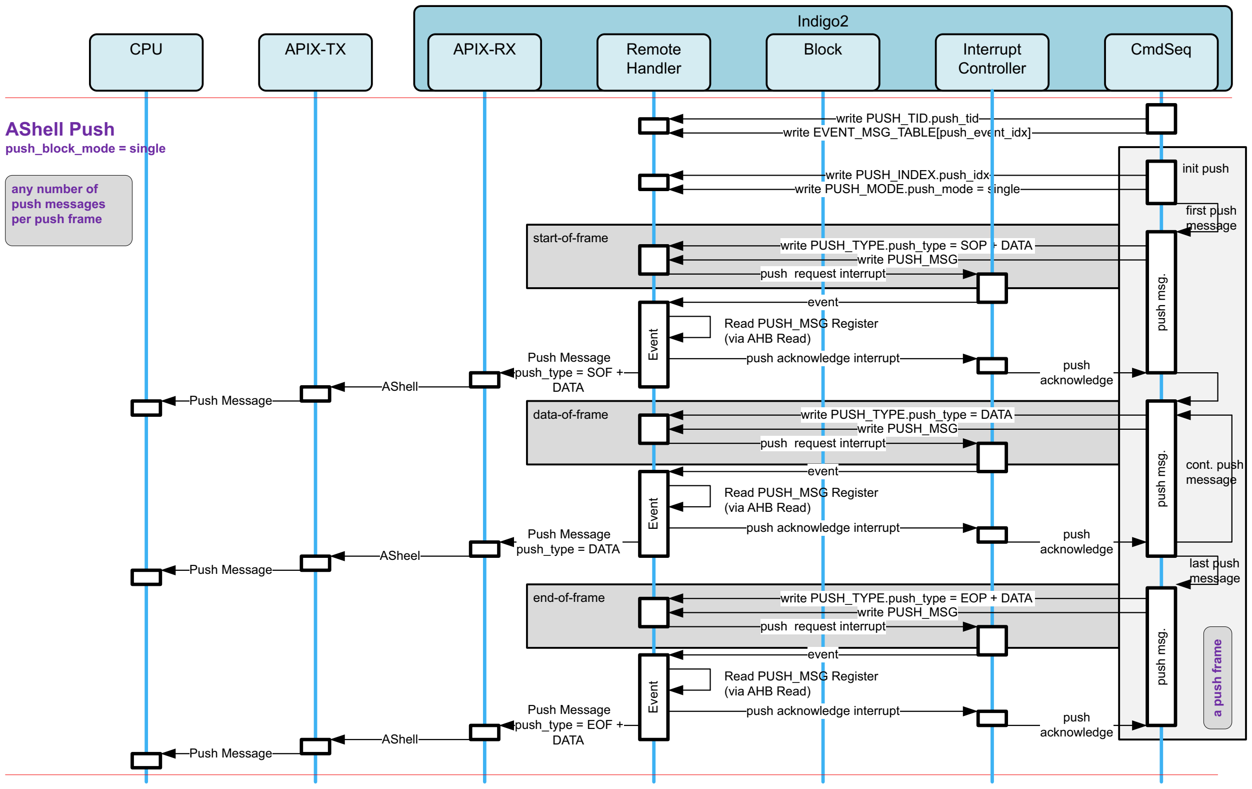

Indigo2
CPU
APIX-TX
APIX-RX
Remote Handler
Block
Interrupt Controller
CmdSeq
AShell Push
push_block_mode = single
any number of push messages per push frame
write PUSH_TID.push_tid
write EVENT_MSG_TABLE[push_event_idx]
init push
write PUSH_INDEX.push_idx
write PUSH_MODE.push_mode = single
first push message
start-of-frame
write PUSH_TYPE.push_type = SOP + DATA
write PUSH_MSG
push request interrupt
push msg.
event
Event
Read PUSH_MSG Register (via AHB Read)
push acknowledge interrupt
push acknowledge
Push Message push_type = SOF + DATA
AShell
Push Message
data-of-frame
write PUSH_TYPE.push_type = DATA
cont. push message
Push Message push_type = DATA
ASheel
last push message
end-of-frame
write PUSH_TYPE.push_type = EOP + DATA
Push Message push_type = EOF + DATA
a push frame

Figure 4-9: Sequence Diagram for Push-Messages (Single Mode)

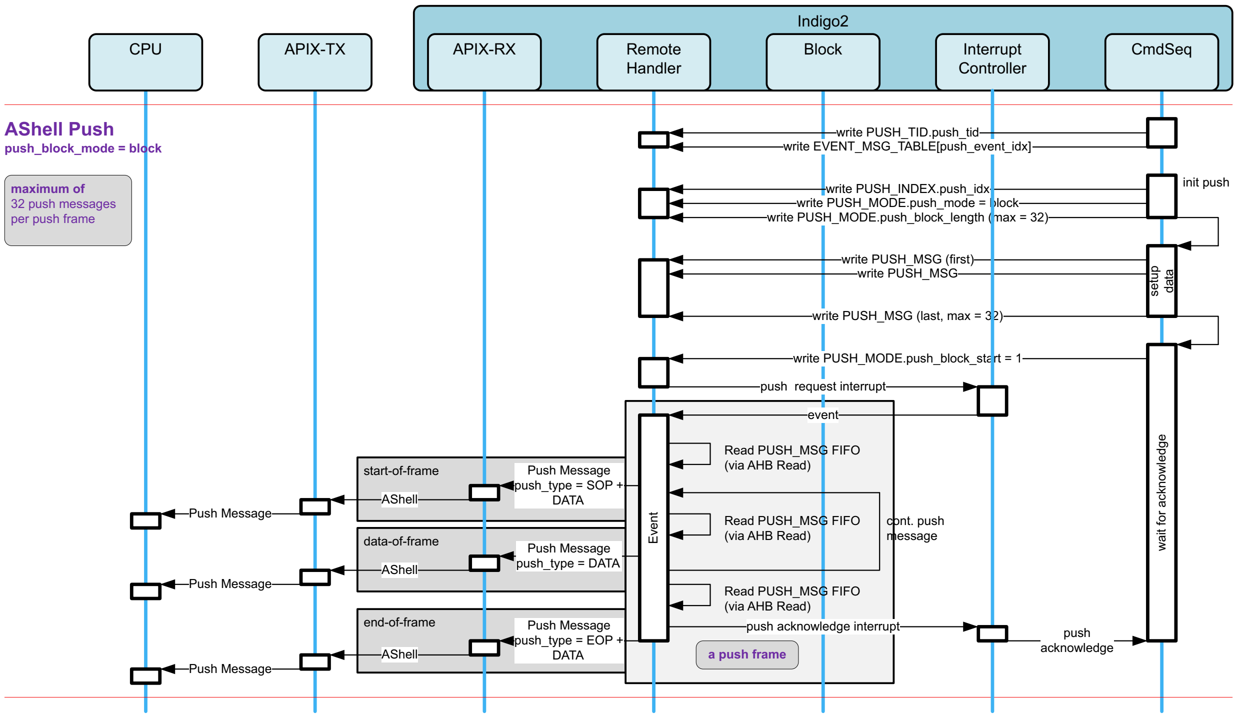
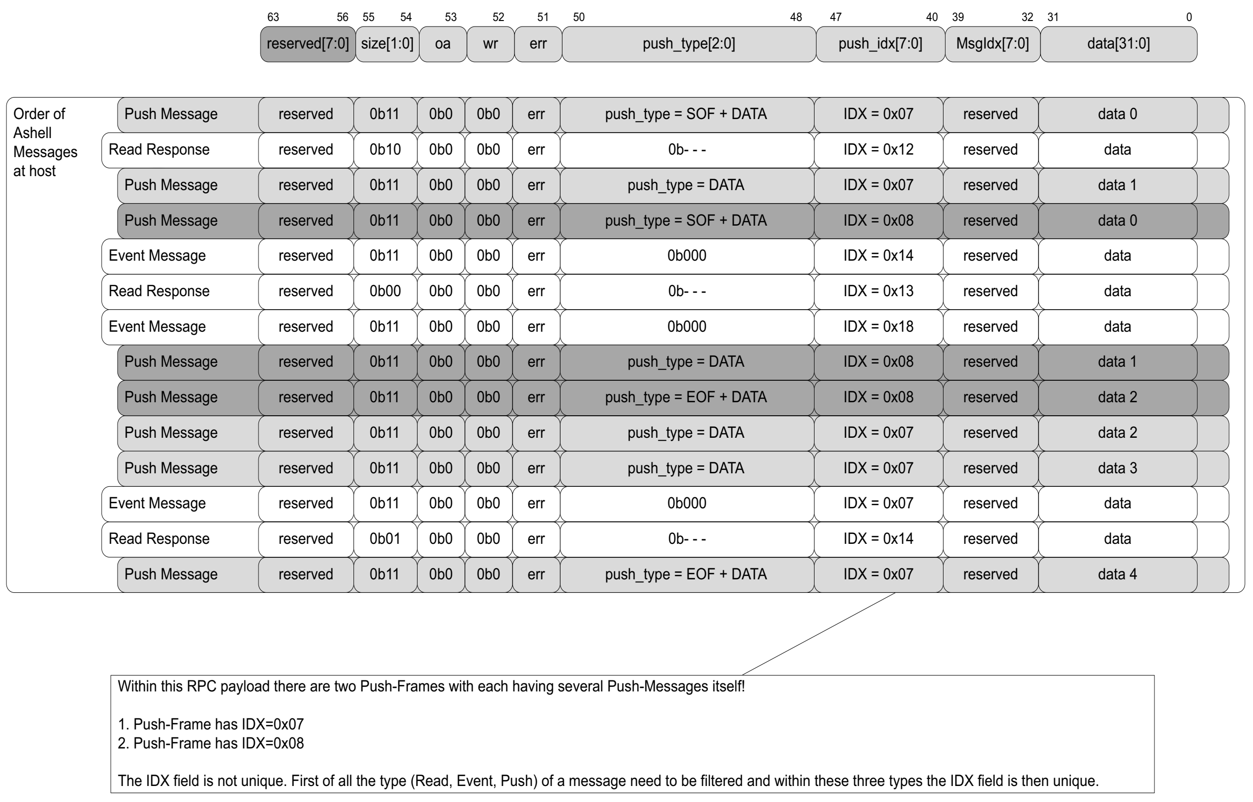

Indigo2
CPU
APIX-TX
APIX-RX
Remote Handler
Block
Interrupt Controller
CmdSeq
AShell Push
push_block_mode = block
maximum of 32 push messages per push frame
write PUSH_TID.push_tid
write EVENT_MSG_TABLE[push_event_idx]
init push
write PUSH_INDEX.push_idx
write PUSH_MODE.push_mode = block
write PUSH_MODE.push_block_length (max = 32)
write PUSH_MSG (first)
write PUSH_MSG
setup data
write PUSH_MSG (last, max = 32)
write PUSH_MODE.push_block_start = 1
push request interrupt
event
Read PUSH_MSG FIFO (via AHB Read)
cont. push message
Event
wait for acknowledge
start-of-frame
Push Message push_type = SOP + DATA
AShell
Push Message
data-of-frame
Push Message push_type = DATA
end-of-frame
Push Message push_type = EOP + DATA
push acknowledge interrupt
push acknowledge
a push frame

Figure 4-10: Sequence Diagram for Push-Messages (Block Mode)

4.6.5 Application

The following figures shows incoming messages at the external host CPU provided by the remote handler. The host system need to be able to sort the corresponding messages.

In general, there is no limitation, that multiple push messages from different push frames occur within a time window. This might happen if (push_mode = single message) is used. Well sorted push frame will occur if push_mode is set to block-transfer mode. But still these sorted push frames may interrupted by read responses.



63–56	55–54	53	52	51	50–48	47–40	39–32	31–0
reserved[7:0]	size[1:0]	oa	wr	err	push_type[2:0]	push_idx[7:0]	MsgIdx[7:0]	data[31:0]

Order of Ashell Messages at host

Push Message	reserved	0b11	0b0	0b0	err	push_type = SOF + DATA	IDX = 0x07	reserved	data 0
Read Response	reserved	0b10	0b0	0b0	err	0b- - -	IDX = 0x12	reserved	data
Push Message	reserved	0b11	0b0	0b0	err	push_type = DATA	IDX = 0x07	reserved	data 1
Push Message	reserved	0b11	0b0	0b0	err	push_type = SOF + DATA	IDX = 0x08	reserved	data 0
Event Message	reserved	0b11	0b0	0b0	err	0b000	IDX = 0x14	reserved	data
Read Response	reserved	0b00	0b0	0b0	err	0b- - -	IDX = 0x13	reserved	data
Event Message	reserved	0b11	0b0	0b0	err	0b000	IDX = 0x18	reserved	data
Push Message	reserved	0b11	0b0	0b0	err	push_type = DATA	IDX = 0x08	reserved	data 1
Push Message	reserved	0b11	0b0	0b0	err	push_type = EOF + DATA	IDX = 0x08	reserved	data 2
Push Message	reserved	0b11	0b0	0b0	err	push_type = DATA	IDX = 0x07	reserved	data 2
Push Message	reserved	0b11	0b0	0b0	err	push_type = DATA	IDX = 0x07	reserved	data 3
Event Message	reserved	0b11	0b0	0b0	err	0b000	IDX = 0x07	reserved	data
Read Response	reserved	0b01	0b0	0b0	err	0b- - -	IDX = 0x14	reserved	data
Push Message	reserved	0b11	0b0	0b0	err	push_type = EOF + DATA	IDX = 0x07	reserved	data 4

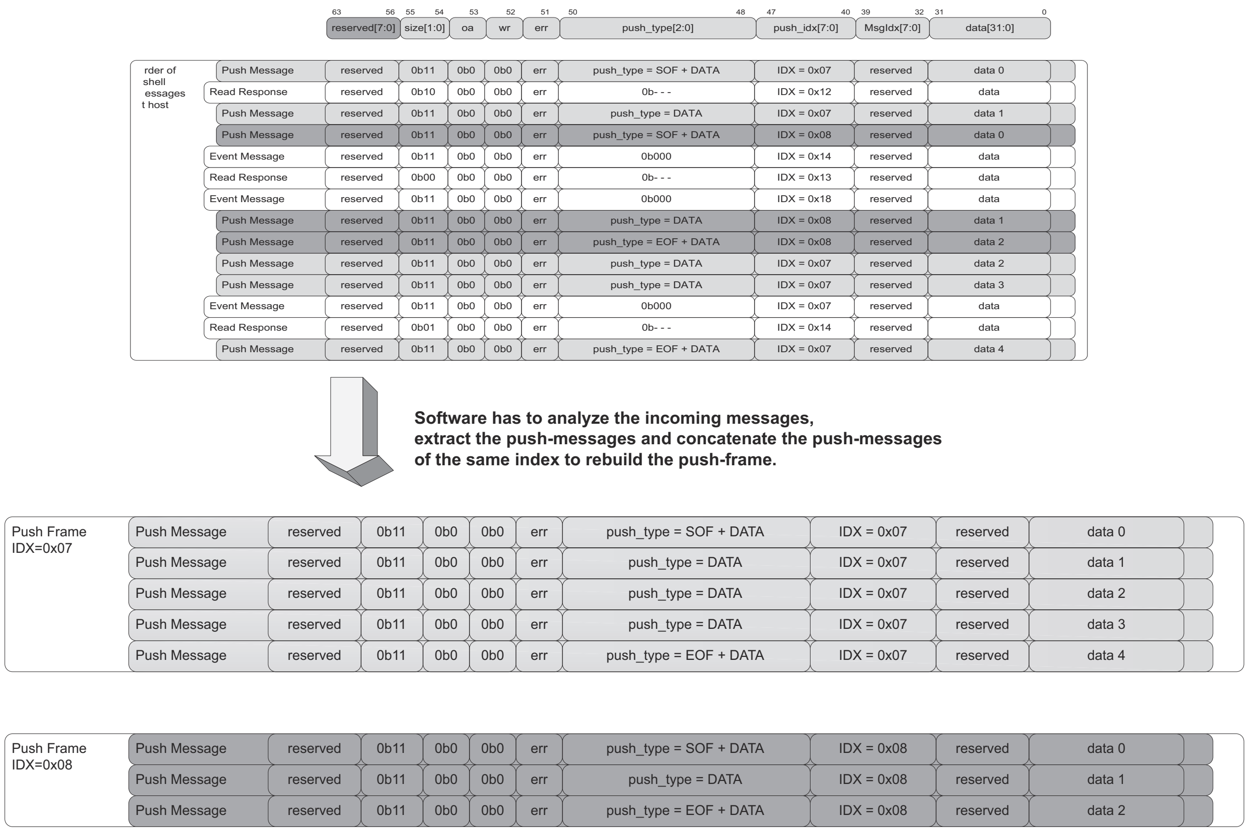
Within this RPC payload there are two Push-Frames with each having several Push-Messages itself!

1. Push-Frame has IDX=0x07
2. Push-Frame has IDX=0x08

The IDX field is not unique. First of all the type (Read, Event, Push) of a message need to be filtered and within these three types the IDX field is then unique.

Figure 4-11: Incoming messages

The next figure shows the push frames after sorting of the host.



63–56	55–54	53	52	51	50–48	47–40	39–32	31–0
reserved[7:0]	size[1:0]	oa	wr	err	push_type[2:0]	push_idx[7:0]	MsgIdx[7:0]	data[31:0]

rder of shell essages t host

Push Message	reserved	0b11	0b0	0b0	err	push_type = SOF + DATA	IDX = 0x07	reserved	data 0
Read Response	reserved	0b10	0b0	0b0	err	0b- - -	IDX = 0x12	reserved	data
Push Message	reserved	0b11	0b0	0b0	err	push_type = DATA	IDX = 0x07	reserved	data 1
Push Message	reserved	0b11	0b0	0b0	err	push_type = SOF + DATA	IDX = 0x08	reserved	data 0
Event Message	reserved	0b11	0b0	0b0	err	0b000	IDX = 0x14	reserved	data
Read Response	reserved	0b00	0b0	0b0	err	0b- - -	IDX = 0x13	reserved	data
Event Message	reserved	0b11	0b0	0b0	err	0b000	IDX = 0x18	reserved	data
Push Message	reserved	0b11	0b0	0b0	err	push_type = DATA	IDX = 0x08	reserved	data 1
Push Message	reserved	0b11	0b0	0b0	err	push_type = EOF + DATA	IDX = 0x08	reserved	data 2
Push Message	reserved	0b11	0b0	0b0	err	push_type = DATA	IDX = 0x07	reserved	data 2
Push Message	reserved	0b11	0b0	0b0	err	push_type = DATA	IDX = 0x07	reserved	data 3
Event Message	reserved	0b11	0b0	0b0	err	0b000	IDX = 0x07	reserved	data
Read Response	reserved	0b01	0b0	0b0	err	0b- - -	IDX = 0x14	reserved	data
Push Message	reserved	0b11	0b0	0b0	err	push_type = EOF + DATA	IDX = 0x07	reserved	data 4

Software has to analyze the incoming messages, extract the push-messages and concatenate the push-messages of the same index to rebuild the push-frame.

Push Frame IDX=0x07

Push Message	reserved	0b11	0b0	0b0	err	push_type = SOF + DATA	IDX = 0x07	reserved	data 0
Push Message	reserved	0b11	0b0	0b0	err	push_type = DATA	IDX = 0x07	reserved	data 1
Push Message	reserved	0b11	0b0	0b0	err	push_type = DATA	IDX = 0x07	reserved	data 2
Push Message	reserved	0b11	0b0	0b0	err	push_type = DATA	IDX = 0x07	reserved	data 3
Push Message	reserved	0b11	0b0	0b0	err	push_type = EOF + DATA	IDX = 0x07	reserved	data 4

Push Frame IDX=0x08

Push Message	reserved	0b11	0b0	0b0	err	push_type = SOF + DATA	IDX = 0x08	reserved	data 0
Push Message	reserved	0b11	0b0	0b0	err	push_type = DATA	IDX = 0x08	reserved	data 1
Push Message	reserved	0b11	0b0	0b0	err	push_type = EOF + DATA	IDX = 0x08	reserved	data 2

Figure 4-12: Push frames after sorting

4.6.6 Event Messages

Table 4-7: Event messages

Event	Name	Description
0	APIX_LINK_FUNC	APIX link functional
1	APIX_LINK_ERR	APIX link error
2	APIX_LINK_FATAL	APIX link fatal error
3	APIX_ASHELL_REQ	APIX Ashell request
4	APIX_ASHELL_FUNC	APIX Ashell functional
5	APIX_ASHELL_ERR	APIX Ashell error
6	APIX_ASHELL_FATAL	APIX Ashell fatal error
7	APIX_PIX_ERR	APIX Ashell Pixel error
8	APIX_PIX_FATAL	APIX Ashell Pixel fatal error
9	APIX_PHY_ARS	APIX PHY recal request
10	APIX_PHY_RES	APIX PHY reset request
11	APIX_PHY_NC1	APIX PHYinterface(notconnected)
12	APIX_PHY_NC2	APIX PHYinterface(notconnected)
13	APIX_HDCP_FUNC	APIX HDCP functional
14	APIX_HDCP_ERR	APIX HDCP error
15	ARH_MAIL_REQ	Ashell Remote Handler Mailbox request interrupt
16	ARH_MAIL_ACK	Ashell Remote Handler Mailbox request done interrupt
17	ARH_PUSH_REQ	Ashell Remote Handler Push message request interrupt
18	ARH_PUSH_ACK	Ashell Remote Handler Push message request done interrupt
19	ARH_RERR	Ashell Remote Handler AHB bus read bus error interrupt
20	ARH_WERR	Ashell Remote Handler AHB bus write bus error interrupt
21	ARH_WRLOCK	Ashell Remote Handler RX interrupt, receive write message while locked
22	ARH_R_THRESH	AshellRemoteHandlerRX-fifo threshold reached
23	ARH_R_OVL	Ashell Remote Handler RX-fifo overflow (loss of message)
24	ARH_T_THRESH	AshellRemoteHandlerTX-fifo threshold reached
25	ARH_T_OVL	Ashell Remote Handler TX-fifo overflow (loss of message)
26	ARH_T_TOUT	AshellRemoteHandler TCTRL timeout (loss of message)
27	ERH_MAIL_REQ	E2IP Remote Handler Mailbox request interrupt
28	ERH_MAIL_ACK	E2IP Remote Handler Mailbox request done interrupt
29	ERH_PUSH_REQ	E2IP Remote Handler Push message request interrupt
30	ERH_PUSH_ACK	E2IP Remote Handler Push message request done interrupt
31	ERH_RERR	E2IP Remote Handler AHB bus read bus error interrupt
32	ERH_WERR	E2IP Remote Handler AHB bus write bus error interrupt
33	ERH_WRLOCK	E2IP Remote Handler RX interrupt, receive write message while locked
34	ERH_R_THRESH	E2IPRemoteHandlerRX-fifo threshold reached
35	ERH_R_OVL	E2IP Remote Handler RX-fifo overflow (loss of message)
36	ERH_T_THRESH	E2IPRemoteHandlerTX-fifo threshold reached
37	ERH_T_OVL	E2IP Remote Handler TX-fifo overflow (loss of message)
38	ERH_T_TOUT	E2IP Remote Handler TCTRL timeout (loss of message)
39	E2IP_RX_DROP	E2IP RX frame dropped
40	E2IP_TX_DROP	E2IP TX frame dropped
41	E2IP_RX_OVWR	E2IP RX frame dropped, while not already processed
42	E2IP_MAC0_UDT	E2IP MAC address of Host 0 updated
43	E2IP_MAC1_UDT	E2IP MAC address of Host 1 updated
44	CFF_ALL	Combination of all Config FIFO interrupts
45	CFF_RERR	Config FIFO AHB Master received ERROR response interrupt
46	CFF_DW7	Config FIFO Data written channel 7 interrupt
47	CFF_DW6	Config FIFO Data written channel 6 interrupt
48	CFF_DW5	Config FIFO Data written channel 5 interrupt

Table 4-7: Event messages (Continued)

Event	Name	Description
49	CFF_DW4	Config FIFO Data written channel 4 interrupt
50	CFF_DW3	Config FIFO Data written channel 3 interrupt
51	CFF_DW2	Config FIFO Data written channel 2 interrupt
52	CFF_DW1	Config FIFO Data written channel 1 interrupt
53	CFF_DW0	Config FIFO Data written channel 0 interrupt
54	CFF_UFLW7	Config FIFO Underflow channel 7 interrupt
55	CFF_OFLW7	Config FIFO Overflow channel 7 interrupt
56	CFF_UTHD7	Config FIFO Upper Threshold channel 7 interrupt
57	CFF_LTHD7	Config FIFO Lower Threshold channel 7 interrupt
58	CFF_UFLW6	Config FIFO Underflow channel 6 interrupt
59	CFF_OFLW6	Config FIFO Overflow channel 6 interrupt
60	CFF_UTHD6	Config FIFO Upper Threshold channel 6 interrupt
61	CFF_LTHD6	Config FIFO Lower Threshold channel 6 interrupt
62	CFF_UFLW5	Config FIFO Underflow channel 5 interrupt
63	CFF_OFLW5	Config FIFO Overflow channel 5 interrupt
64	CFF_UTHD5	Config FIFO Upper Threshold channel 5 interrupt
65	CFF_LTHD5	Config FIFO Lower Threshold channel 5 interrupt
66	CFF_UFLW4	Config FIFO Underflow channel 4 interrupt
67	CFF_OFLW4	Config FIFO Overflow channel 4 interrupt
68	CFF_UTHD4	Config FIFO Upper Threshold channel 4 interrupt
69	CFF_LTHD4	Config FIFO Lower Threshold channel 4 interrupt
70	CFF_UFLW3	Config FIFO Underflow channel 3 interrupt
71	CFF_OFLW3	Config FIFO Overflow channel 3 interrupt
72	CFF_UTHD3	Config FIFO Upper Threshold channel 3 interrupt
73	CFF_LTHD3	Config FIFO Lower Threshold channel 3 interrupt
74	CFF_UFLW2	Config FIFO Underflow channel 2 interrupt
75	CFF_OFLW2	Config FIFO Overflow channel 2 interrupt
76	CFF_UTHD2	Config FIFO Upper Threshold channel 2 interrupt
77	CFF_LTHD2	Config FIFO Lower Threshold channel 2 interrupt
78	CFF_UFLW1	Config FIFO Underflow channel 1 interrupt
79	CFF_OFLW1	Config FIFO Overflow channel 1 interrupt
80	CFF_UTHD1	Config FIFO Upper Threshold channel 1 interrupt
81	CFF_LTHD1	Config FIFO Lower Threshold channel 1 interrupt
82	CFF_UFLW0	Config FIFO Underflow channel 0 interrupt
83	CFF_OFLW0	Config FIFO Overflow channel 0 interrupt
84	CFF_UTHD0	Config FIFO Upper Threshold channel 0 interrupt
85	CFF_LTHD0	Config FIFO Lower Threshold channel 0 interrupt
86	RLT0	Reload timer 0 interrupt
87	RLT1	Reload timer 1 interrupt
88	RLT2	Reload timer 2 interrupt
89	RLT3	Reload timer 3 interrupt
90	RLT4	Reload timer 4 interrupt
91	RLT5	Reload timer 5 interrupt
92	RLT6	Reload timer 6 interrupt
93	RLT7	Reload timer 7 interrupt
94	RLT8	Reload timer 8 interrupt
95	RLT9	Reload timer 9 interrupt
96	RLT10	Reload timer 10 interrupt
97	RLT11	Reload timer 11 interrupt
98	RLT12	Reload timer 12 interrupt
99	RLT13	Reload timer 13 interrupt
100	RLT14	Reload timer 14 interrupt

Table 4-7: Event messages (Continued)

Event	Name	Description
101	RLT15	Reload timer 15 interrupt
102	LIN_R	LIN Reception interrupt
103	LIN_T	LIN Transmission interrupt
104	LIN_E	LIN Error interrupt
105	PPG00	PPG / PWM module 0 interrupt 0
106	PPG01	PPG / PWM module 0 interrupt 1
107	PPG02	PPG / PWM module 0 interrupt 2
108	PPG03	PPG / PWM module 0 interrupt 3
109	PPG10	PPG / PWM module 1 interrupt 0
110	PPG11	PPG / PWM module 1 interrupt 1
111	PPG12	PPG / PWM module 1 interrupt 2
112	PPG13	PPG / PWM module 1 interrupt 3
113	PPG20	PPG / PWM module 2 interrupt 0
114	PPG21	PPG / PWM module 2 interrupt 1
115	PPG22	PPG / PWM module 2 interrupt 2
116	PPG23	PPG / PWM module 2 interrupt 3
117	PPG30	PPG / PWM module 3 interrupt 0
118	PPG31	PPG / PWM module 3 interrupt 1
119	PPG32	PPG / PWM module 3 interrupt 2
120	PPG33	PPG / PWM module 3 interrupt 3
121	I2C0_IRQ	I2C0 Operational interrupt
122	I2C0_ERIRQ	I2C0 Error interrupt
123	I2C1_IRQ	I2C1 Operational interrupt
124	I2C1_ERIRQ	I2C1 Error interrupt
125	SGE_IRQ	Sound generator interrupt
126	SGE_RLD	Sound generator register reload interrupt
127	ADC_IRQ	ADC Conversion end interrupt
128	ADC2_IRQ	ADC Scan end interrupt
129	ADC_RCOIRQ	ADC Range comparator interrupt
130	ADC_ADPIRQ	ADC pulse detection interrupt
131	EIRQ_0	external IRQ pin 0 interrupt
132	EIRQ_1	external IRQ pin 1 interrupt
133	EIRQ_2	external IRQ pin 2 interrupt
134	EIRQ_3	external IRQ pin 3 interrupt
135	EIRQ_4	external IRQ pin 4 interrupt
136	EIRQ_5	external IRQ pin 5 interrupt
137	EIRQ_6	external IRQ pin 6 interrupt
138	EIRQ_7	external IRQ pin 7 interrupt
139	ESPI_RX	External device SPI Reception interrupt
140	ESPI_TX	External device SPI Transmission interrupt
141	ESPI_FAULT	External device SPI Fault interrupt
142	IRS_PE_SC0	Iris-MVL pixel engine sequence complete (synchronizer 0)
143	IRS_PE_SC1	Iris-MVL pixel engine sequence complete (synchronizer 1)
144	IRS_PE_FC0	Iris-MVL pixel engine frame complete (extdst 0)
145	IRS_PE_FC1	Iris-MVL pixel engine frame complete (extdst 1)
146	IRS_LB0_SL	Iris-MVL layer blend 0 shadow register loaded
147	IRS_LB1_SL	Iris-MVL layer blend1shadow loaded
148	IRS_DE_SL	Iris-MVL display engine top shadow loaded
149	IRS_DE_SC	Iris-MVL display engine sequence complete
150	IRS_FG_P0	Iris-MVL framegeneratorprogrammable interrupt 0
151	IRS_FG_P1	Iris-MVL framegeneratorprogrammable interrupt 1
152	IRS_FG_P2	Iris-MVL framegeneratorprogrammable interrupt 2

Table 4-7: Event messages (Continued)

Event	Name	Description
153	IRS_FG_P3	Iris-MVL framegeneratorprogrammable interrupt 3
154	IRS_FG_SL_P	Iris-MVL frame generator shadow register loaded (primary input)
155	IRS_FG_SL_S	Iris-MVL frame generator shadow register loaded (secondary input)
156	IRS_SIG0_SL	Iris-MVL signature unit 0 shadow loaded
157	IRS_SIG0_RDY	Iris-MVL signature unit 0 measurement complete
158	IRS_SIG0_ERR	Iris-MVL signature unit 0 signature error
159	IRS_SIG1_SL	Iris-MVL signature unit 1 shadow loaded
160	IRS_SIG1_RDY	Iris-MVL signature unit 1 measurement complete
161	IRS_SIG1_ERR	Iris-MVL signature unit 1 signature error
162	IRS_SIG2_SL	Iris-MVL signature unit 2 shadow loaded
163	IRS_SIG2_RDY	Iris-MVL signature unit 2 measurement complete
164	IRS_SIG2_ERR	Iris-MVL signature unit 2 signature error
165	IRS_SIG3_SL	Iris-MVL signature unit 3 shadow loaded
166	IRS_SIG3_RDY	Iris-MVL signature unit 3 measurement complete
167	IRS_SIG3_ERR	Iris-MVL signature unit 3 signature error
168	IRS_FG_SYNC_P	Iris-MVL frame generator synchronization stable (primary input)
169	IRS_FG_SYNCERR_P	Iris-MVL frame generator synchronization loss (primary input)
170	IRS_FG_SYNC_S	Iris-MVL frame generator synchronization stable (secondary input)
171	IRS_FG_SYNCERR_S	Iris-MVL frame generator synchronization loss (secondary input)
172	IRS_FC_SYNC	Iris-MVL frame capture synchronization stable
173	IRS_FC_SYNCERR	Iris-MVL frame capture synchronization loss
174	CMDSEQ_WDG	Command Sequencer watchdog interrupt (watchdog status)
175	CMDSEQ_SWINT	Command Sequencer software interrupt
176	CMDSEQ_LWM	Command Sequencer command buffer lowwatermark interrupt (counter reaches low water mark)
177	CMDSEQ_HWM	Command Sequencer command buffer high watermark interrupt (counter reaches high water mark)
178	CMDSEQ_ERROR	Command Sequencer error interrupt (error on illegal instruction)
179	CMDSEQ_HALT	Command Sequencer halt interrupt (core is in halt state)
180	CMDSEQ_EMPTY	Command Sequencer command buffer fifo empty interrupt
181	CMDSEQ_FULL	Command Sequencer command buffer fifo full interrupt
182	GC_ALV	Global Control Alive sender IRQ
183	GC_WDG	System Watchdog interrupt
184	LVD_L_R	Low voltage detection core voltage low threshold comparator going high interrupt
185	LVD_L_F	Low voltage detection core voltage low threshold comparator going low interrupt
186	LVD_H_R	Low voltage detection core voltage high threshold comparator going high interrupt
187	LVD_H_F	Low voltage detection core voltage high threshold comparator going low interrupt
188	PANIC_SWITCH	Panic switch was asserted
189	HIFC	Host interface AHB bus error interrupt
190	FLSH	Flashinterfaceinterrupt (ready, hang or single bit error)
191	DMAC_DIRQ	DMA Controller single ORed output of all the DIRQx generated from each Channel
192	DMAC_DIRQ0	DMA Controller end of DMA transfer channel 0
193	DMAC_DIRQ1	DMA Controller end of DMA transfer channel 1
194	DMAC_EIRQ	DMA Controller single ORed output of all the EIRQx generated from each Channel
195	DMAC_EIRQ0	DMA Controller error DMA channel 0
196	DMAC_EIRQ1	DMA Controller error DMA channel 1
197	FSPI_RX	External Flash SPI Reception interrupt

Table 4-7: Event messages (Continued)

Event	Name	Description
198	FSPI_TX	External Flash SPI Transmission interrupt
199	FSPI_FAULT	External Flash SPI Fault interrupt
200	PRGCRC_IRQ	Programmable CRC completion interrupt
201	RBUS_BUSERR	RBUS interconnect error (signaled by RBUS error collection unit)
202	ERBUS_BUSERR	eRBUS interconnect error (signaled by eRBUS error collection unit)
203	EXTIRQ_BUSERR	External IRQ unit signals AHB interface error
204	ESPI_BUSERR	External device SPI unit signals AHB interface error
205	FSPI_BUSERR	External Flash SPI unit signals AHB interface error
206	PRGCRC_BUSERR	Programmable CRC unit signals AHB interface error
207	IFLASH_BUSERR	Internal Flash interface signals AHB interface error
208	IFLASH_TCBUSERR	Internal Flash interface signals TC interface error

4.6.7 AShell Remote Handler Register Overview

Table 4-8: Registers Overview

Base Address(es)	**Instance no 0: BASEADDR0="00022000"** **Instance no 1: BASEADDR1="00024000"**	
Absolute Address	**Register Name**	**Register Description**
BASEADDRx + 0x0000	RH_CTRL	Remote Handler Control Register
BASEADDRx + 0x0004	RH_STAT	Remote Handler Status Register
BASEADDRx + 0x0008	RESET_CTRL	Reset Control Register
BASEADDRx + 0x000C	AHBM_LOCK	AHB Master Lock Register
BASEADDRx + 0x0010	BASE_ADDR_WRITE	Base Address Register for Write Message
BASEADDRx + 0x0014	BASE_ADDR_READ	Base Address Register for Read Message
BASEADDRx + 0x0018	BASE_ADDR_EVENT	Base Address Register for Event and Push Message
BASEADDRx + 0x001C	FIFO_THRESH_VAL	FIFO Threshold Register
BASEADDRx + 0x0020	FIFO_STAT	FIFO Status Register
BASEADDRx + 0x0024	RH_IRQ_EN	Remote Handler Interrupt Enable
BASEADDRx + 0x0028	Reserved	Do not modify
BASEADDRx + 0x002C	MAILBOX	Send an event message to the host
BASEADDRx + 0x0030	PUSH_MODE	Push Mode Register
BASEADDRx + 0x0034	PUSH_MSG	Push Message Data
BASEADDRx + 0x0038	PUSH_INDEX	Setup the index identifier of push message
BASEADDRx + 0x003C	PUSH_TYPE	Setup the type of the push message
BASEADDRx + 0x0040	PUSH_TID	PUSH message target ID setup
BASEADDRx + 0x0044	ASHELL_FLUSH	AShell Message Flush
BASEADDRx + 0x0048	EVENT_STAT0	Event Status Register 0
BASEADDRx + 0x004C	EVENT_STAT1	Event Status Register 1
BASEADDRx + 0x0050	EVENT_STAT2	Event Status Register 2
BASEADDRx + 0x0054	EVENT_STAT3	Event Status Register 3
BASEADDRx + 0x0058	EVENT_STAT4	Event Status Register 4
BASEADDRx + 0x005C	EVENT_STAT5	Event Status Register 5
BASEADDRx + 0x0060	EVENT_STAT6	Event Status Register 6
BASEADDRx + 0x0064	EVENT_STAT7	Event Status Register 7
BASEADDRx + 0x0068	EVENT_EN0	Event Enable Register 0
BASEADDRx + 0x006C	EVENT_EN1	Event Enable Register 1
BASEADDRx + 0x0070	EVENT_EN2	Event Enable Register 2
BASEADDRx + 0x0074	EVENT_EN3	Event Enable Register 3
BASEADDRx + 0x0078	EVENT_EN4	Event Enable Register 4
BASEADDRx + 0x007C	EVENT_EN5	Event Enable Register 5
BASEADDRx + 0x0080	EVENT_EN6	Event Enable Register 6
BASEADDRx + 0x0084	EVENT_EN7	Event Enable Register 7
BASEADDRx + 0x0088	AHB_WRERR_ADDR	AHB Write-Error Address Register
BASEADDRx + 0x008C	AHB_RDERR_ADDR	AHB Read-Error Address Register
BASEADDRx + 0x0400	EVENT_MSG_TABLE	Event Lookup Table

4.7 Embedded Ethernet

The objective of Embedded Ethernet block (E2IP) is to enable small, so called satellite or register controllers for Ethernet/IP based communication. E2IP is implemented in hardware without the need of a microcontroller core or any specific software overhead. In order for the E2IP hardware block to efficiently deal with the incoming Ethernet Frame, the exact byte-wise layout is specified.

The E2IP extracts the payload from the Ethernet packet and forwards the payload (AShell messages) to a exclusive Remote Handler which only take care about request from E2IP.

NOTE This functionality is not implemented in MB88F335 ‘Indigo2-S’

4.7.1 Features

- Ethernet
  - E2IP is a hardware Ethernet MAC
  - 10MBit and 100MBit is supported
  - Ethernet communication to two predefined hosts
  - Full and half duplex for external PHY
  - Full duplex for Ethernet-via-APIX in non daisy chain mode
  - Full duplex with shared medium for Ethernet-via-APIX in daisy chain mode
  - Receive unicast Ethernet frames
  - Receive multicast Ethernet frames for ARP only
  - Transmit unicast and multicast Ethernet frames
  - Host MAC address learning (no ARP)
- Protocols
  - ARP request and reply
  - ICMP echo requests and reply
  - IPv4
  - UDP
  - AUTOSAR
  - Remote Handler Messages (AUTOSAR payload)
- Others
  - Dataflow handling
  - Drop counter for unexpected drops of Ethernet frames
  - Complex trigger scheme fro transmission

- Limitations:
  - ICMP echo reply limited to last 64-bytes
  - UDP contains a single AUTOSAR packet
  - AUTOSAR payload limited to 32 Remote Handler Messages

4.7.2 Block Diagram

The following figure shows the Embedded Ethernet block within its environment. This figure shall provide a brief overview of the Embedded Ethernet block and the connected Remote Handler.

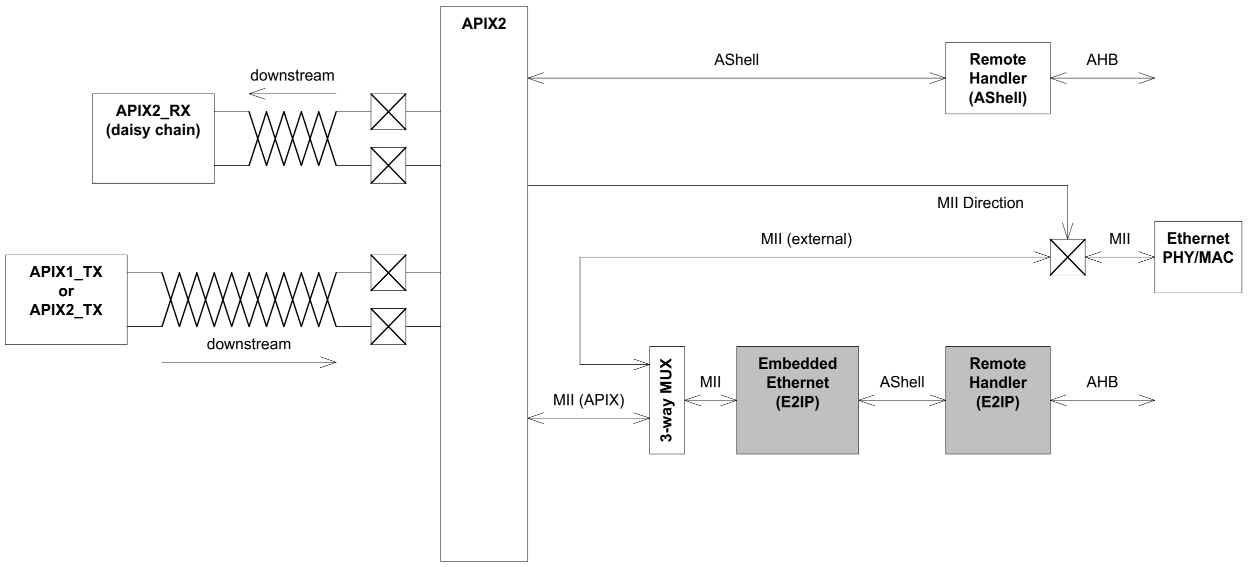


Figure 4-13: Block diagram

4.7.3 Functional Description

The following figures shows the major data paths of the Embedded Ethernet block and the connected Remote Handler.

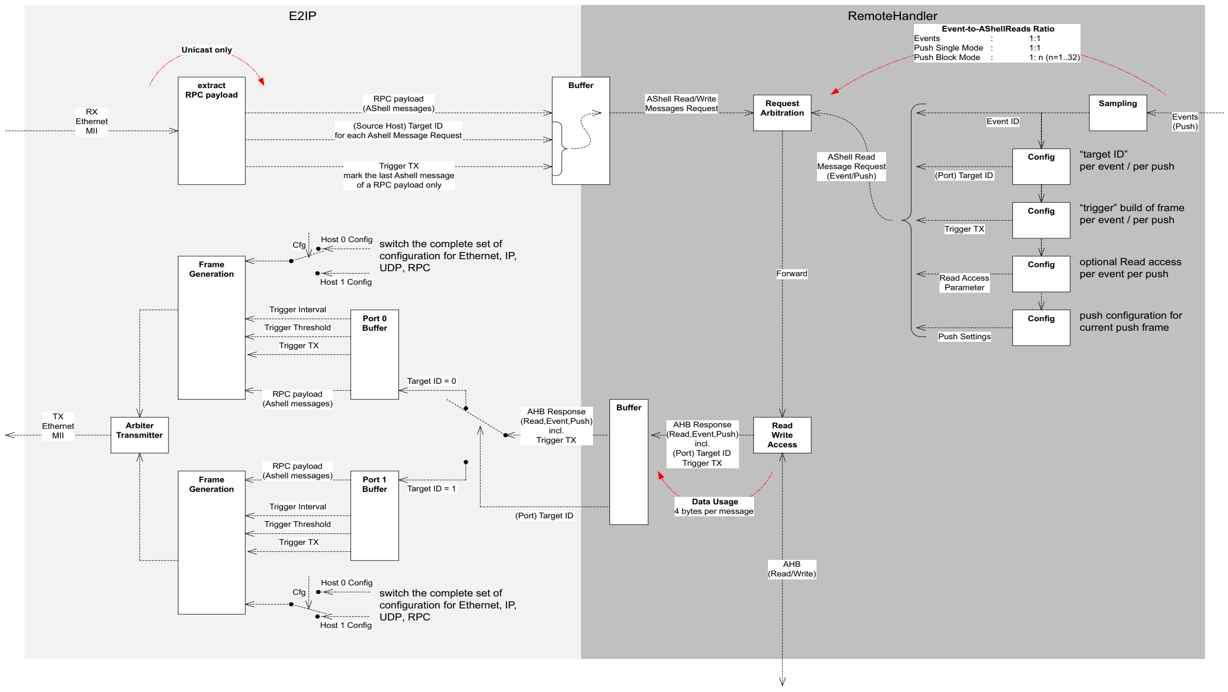

E2IP
RemoteHandler
Unicast only
extract RPC payload
RX Ethernet MII
RPC payload (AShell messages)
(Source Host) Target ID for each Ashell Message Request
Trigger TX mark the last Ashell message of a RPC payload only
Buffer
AShell Read/Write Messages Request
Request Arbitration
Event-to-AShellReads Ratio
Events : 1:1
Push Single Mode : 1:1
Push Block Mode : 1: n (n=1..32)
Sampling
Events (Push)
Event ID
AShell Read Message Request (Event/Push)
Config
"target ID" per event / per push
(Port) Target ID
Config
"trigger" build of frame per event / per push
Trigger TX
Config
optional Read access per event per push
Read Access Parameter
Config
push configuration for current push frame
Push Settings
Forward
Cfg
Host 0 Config
Host 1 Config
switch the complete set of configuration for Ethernet, IP, UDP, RPC
Frame Generation
Trigger Interval
Trigger Threshold
Trigger TX
Port 0 Buffer
Target ID = 0
RPC payload (Ashell messages)
TX Ethernet MII
Arbiter Transmitter
AHB Response (Read,Event,Push) incl. Trigger TX
Buffer
AHB Response (Read,Event,Push) incl. (Port) Target ID Trigger TX
Read Write Access
RPC payload (Ashell messages)
Frame Generation
Port 1 Buffer
Target ID = 1
(Port) Target ID
Data Usage 4 bytes per message
Trigger Interval
Trigger Threshold
Trigger TX
AHB (Read/Write)
Cfg
Host 0 Config
Host 1 Config
switch the complete set of configuration for Ethernet, IP, UDP, RPC

Figure 4-14: Major data path

The following figure describes the supported frame formats on the RX side of the Embedded Ethernet block.

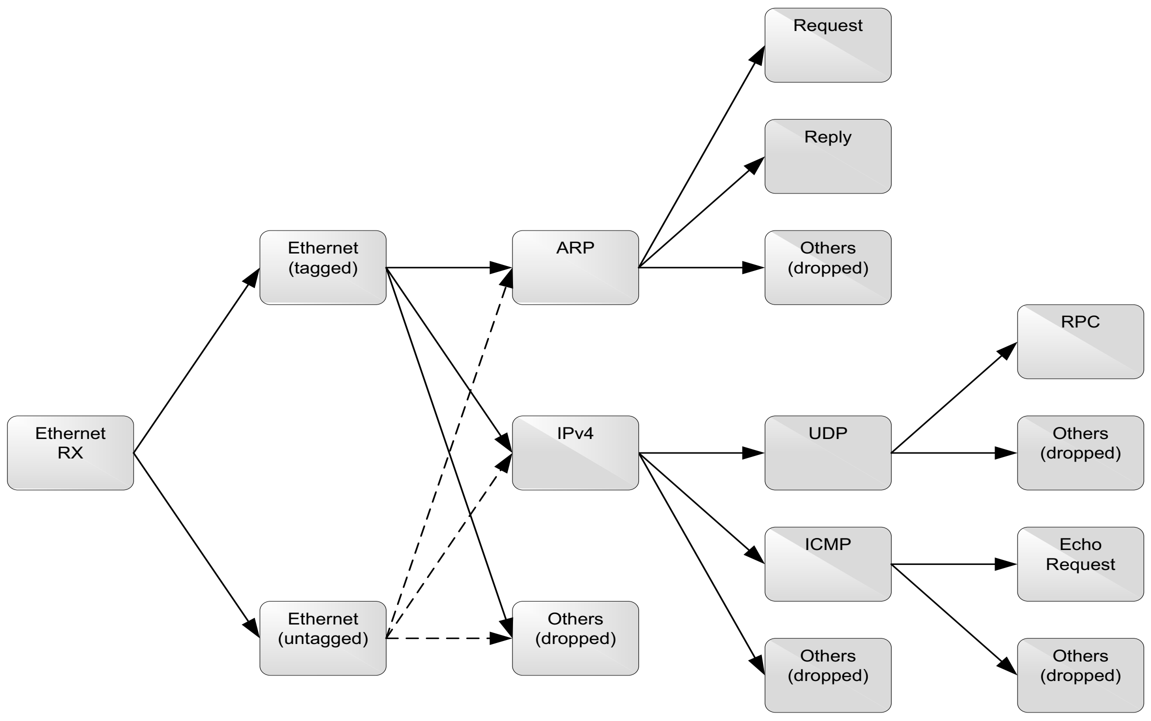


Figure 4-15: Supported frame formats on the RX side

The following figure describes the supported frame formats on the TX side of the Embedded Ethernet block.

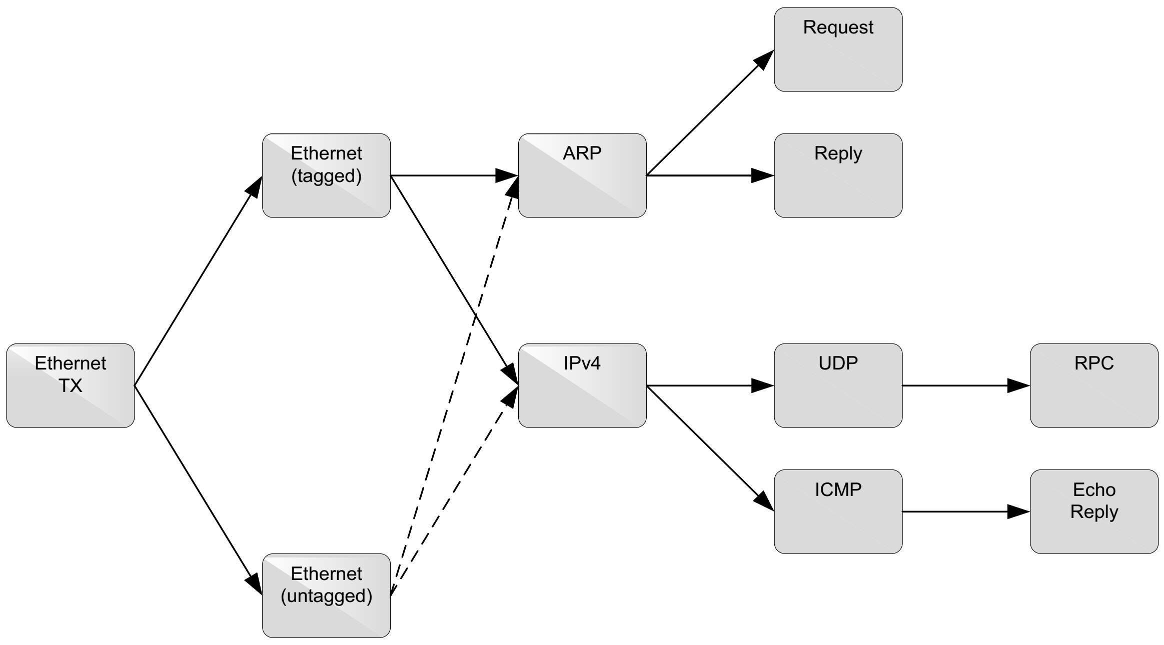
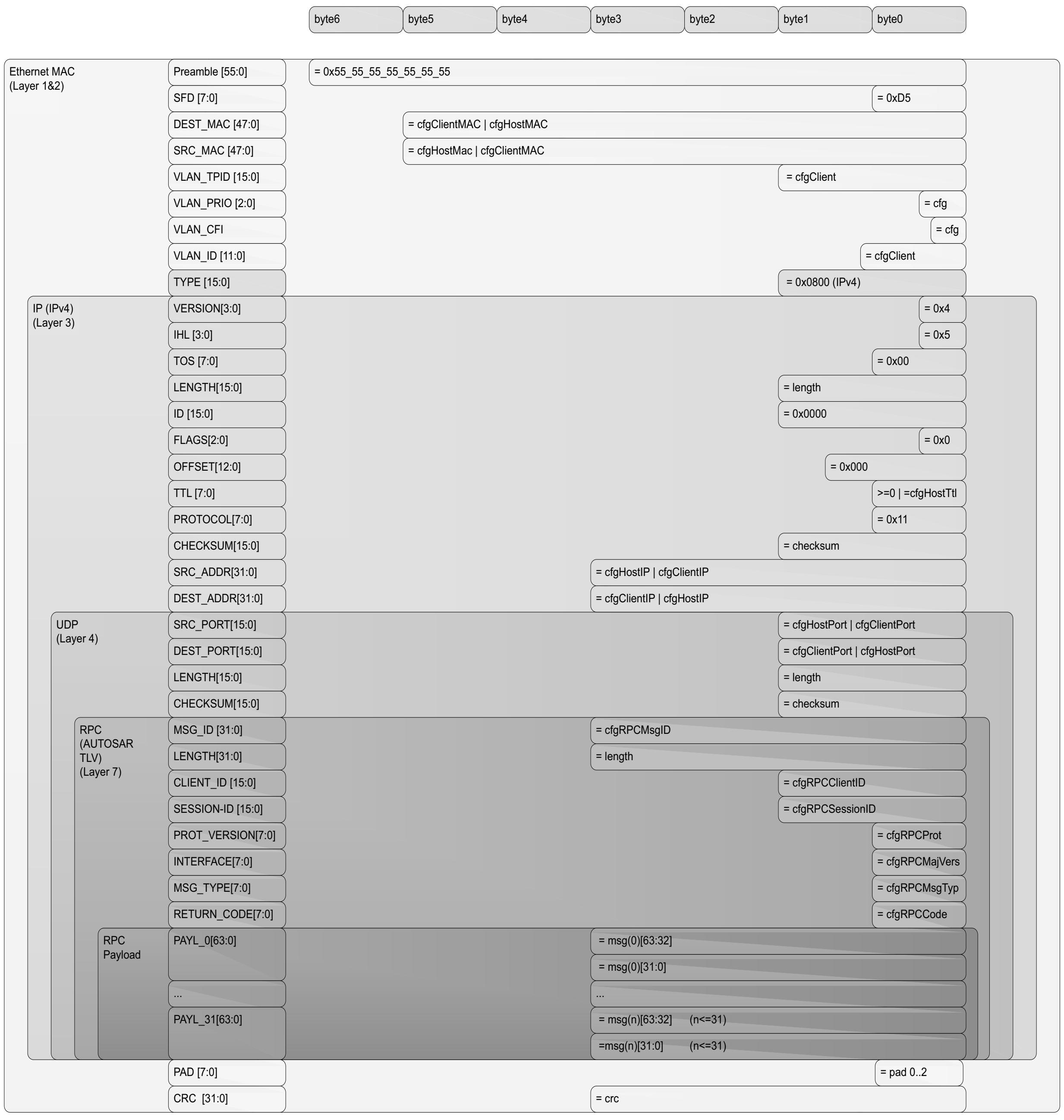


Figure 4-16: Supported formats on the TX side

4.7.4 Operation

4.7.4.1 RPC (AUTOSAR)

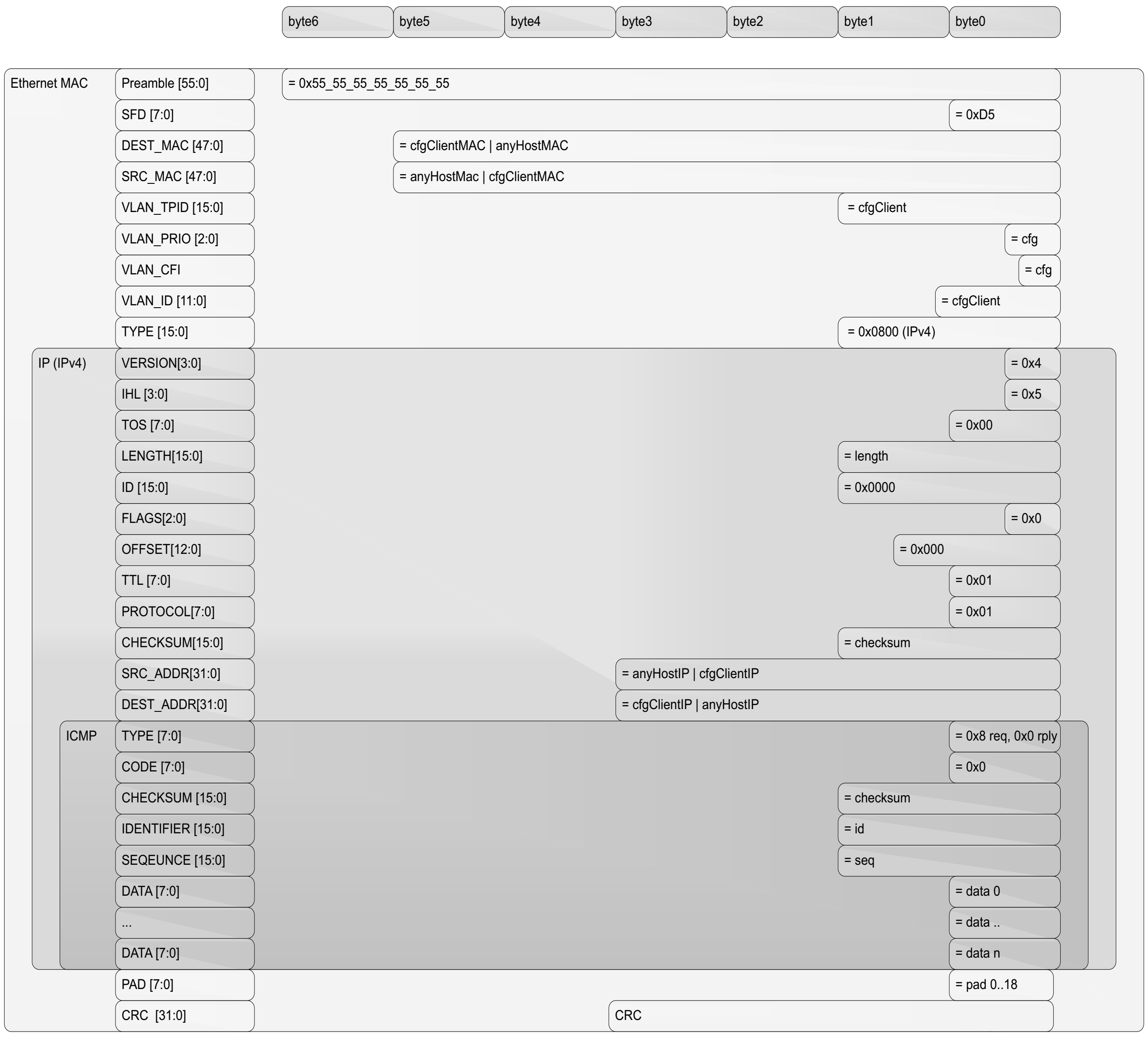
The following figure describes an Ethernet frame of a RPC (AUTOSAR) packet.



4.7.4.2 ICMP Frame Format

The following figures describes an Ethernet frame of an ICMP message.

In general the Embedded Ethernet block is able to receive ICMP echo request and is able to transmit a corresponding echo reply. The echo reply is limited to the last 64-bytes of the echo request.



4.7.4.3 ARP Frame Format

The following figures describes an Ethernet frame of an ARP frame

.

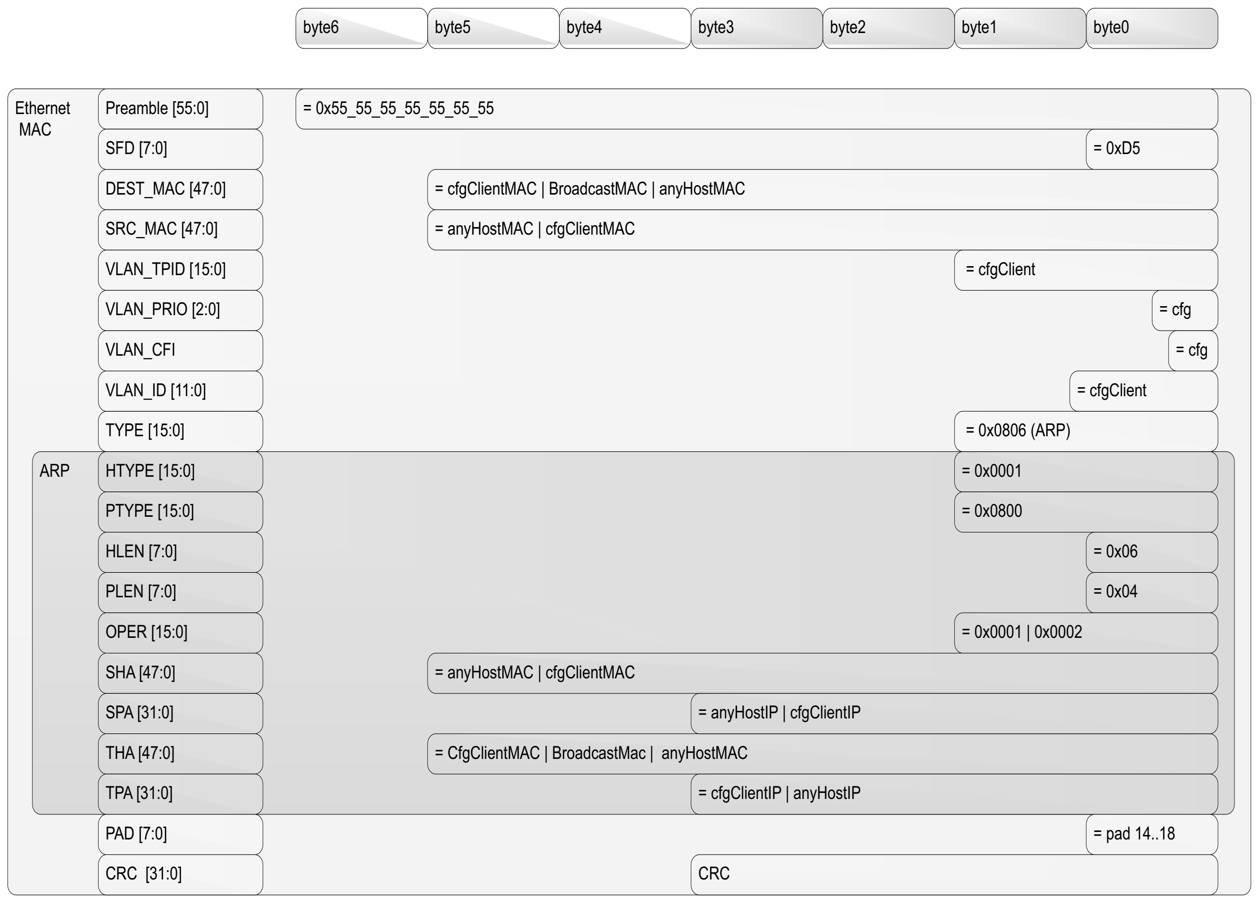


Figure 4-17: ARP frame format

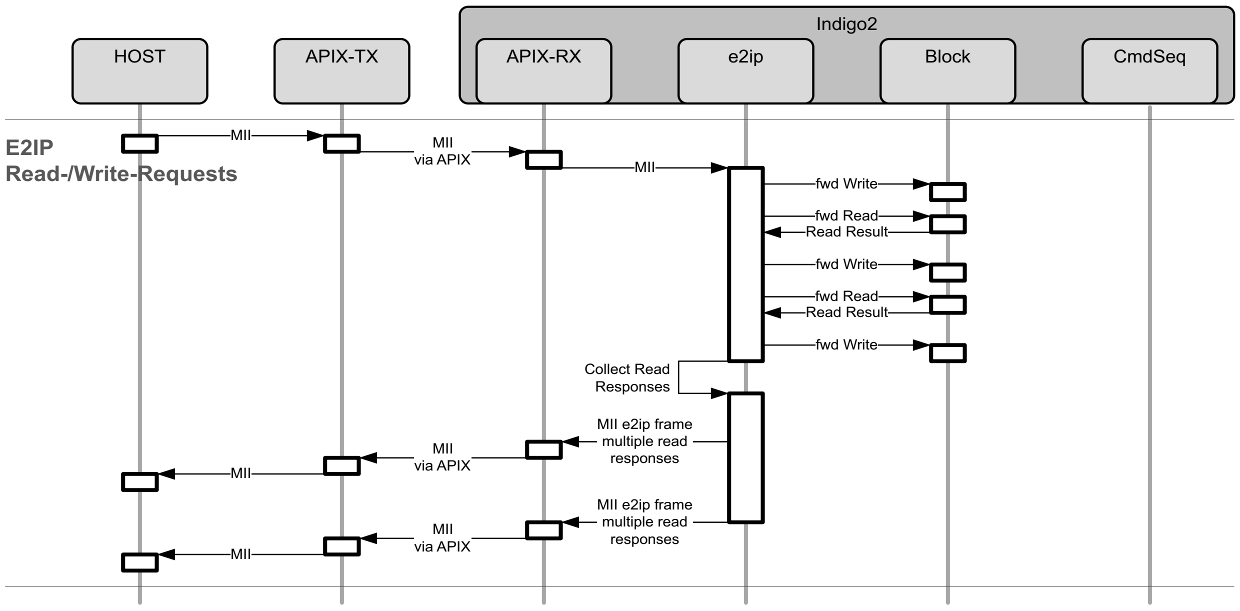
4.7.5 Control Flow

4.7.5.1 Extract and Collect RPC (AUTOSAR) Payload

4.7.5.1.1 Remote Handler Read- and Write Messages

The following figures describes how the RPC payload is extracted from a RPC (AUTOSAR) frame. The RPC payload contains Remote Handler read- and write messages. These Remote Handler read- and write messages are forwarded to the Remote Handler which corresponds to the Embedded Ethernet block.

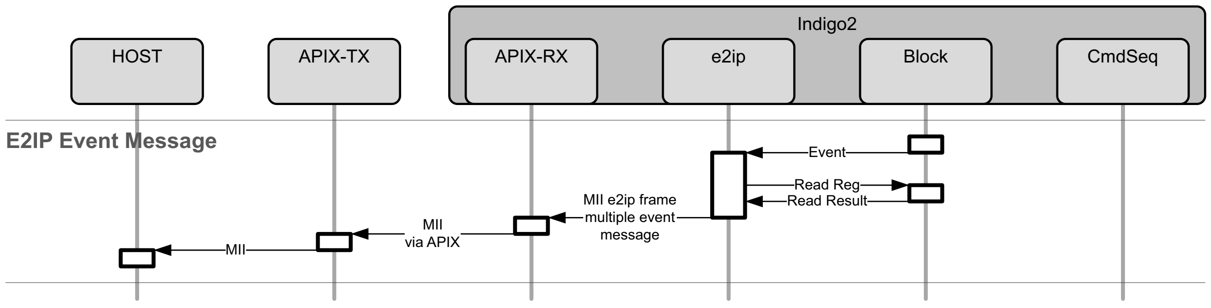
The read-response messages get collected within the Embedded Ethernet block. If the condition of transmission has been reached, the Embedded Ethernet block will combine these messages into one or more RPC payloads. The Embedded Ethernet block will generate the appropriate framing to transmit these payloads via Ethernet. These RPC frames are forwards to the corresponding Ethernet host, from which the request was initiated.



4.7.5.1.2 Remote Handler Event Messages

The following figures describes how Remote Handler event messages are forwarded to an Ethernet host. Remote Handler event messages are generated messages from Indigo2. They are not initiated by the host CPU. Remote Handler event messages might imply an additional AHB read for further diagnostics.

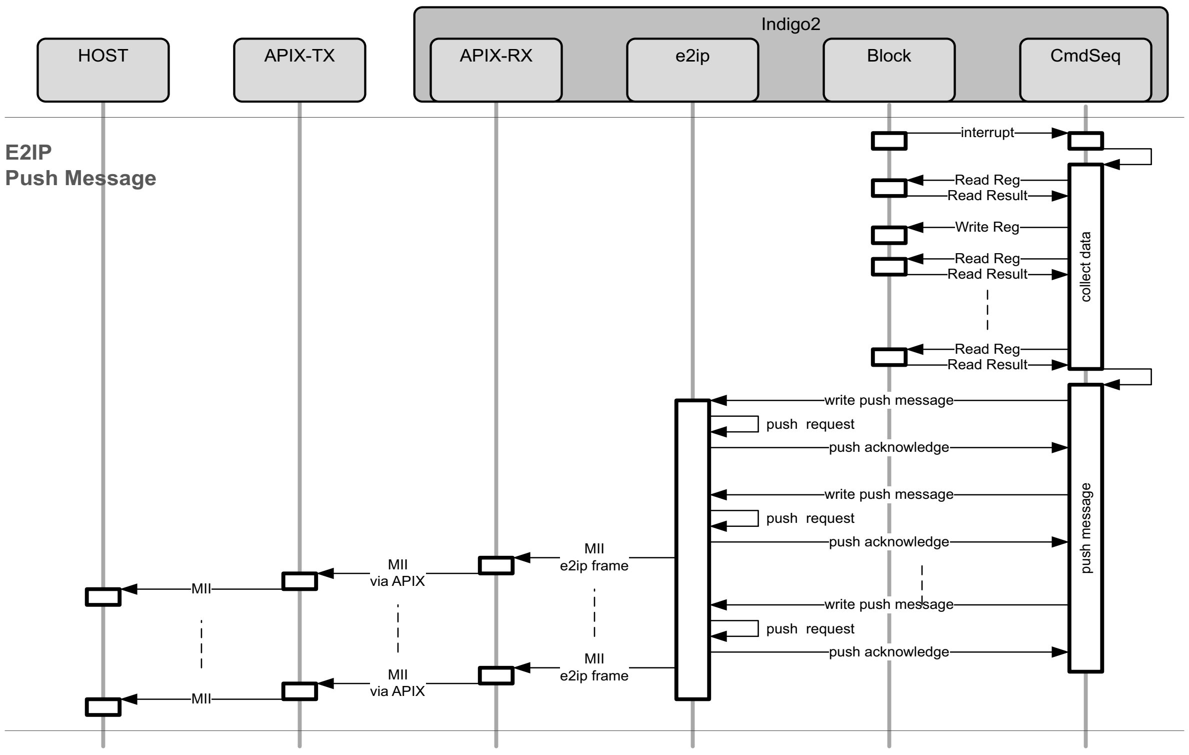
The event messages get collected within the Embedded Ethernet block. If the condition of transmission has been reached, the Embedded Ethernet block will combine messages into one or more RPC payloads. The Embedded Ethernet block will generate the appropriate framing to transmit these payloads via Ethernet. These RPC frames are forwards to the selected Ethernet host.



4.7.5.1.3 Remote Handler Push Messages

The following figures describes how Remote Handler push messages are forwarded to an Ethernet host. Remote Handler push messages are generated messages from Indigo2. They are not initiated by the host CPU. Remote Handler push frame is build out of one or more push messages. Like Remote Handler event messages, Remote Handler push messages will initiate a read request on the AHB interface.

The push messages get collected within the Embedded Ethernet block. If the condition of transmission has been reached, the Embedded Ethernet block will combine messages into one or more RPC payloads. A complete push frame, which consists out of one more push messages, might be split over several RPC payloads. The Embedded Ethernet block will generate the appropriate framing to transmit these payloads via Ethernet. These RPC frames are forwards to the selected Ethernet host.



4.7.5.1.4 Occurrence of Messages

- Remote Handler read response messages are initiated by read request messages at the host CPU.
- Remote Handler event messages are initiated by an internal state (e.g. interrupt) of Indigo2.
- Remote Handler push messages are initiated based on an internal or external event.

These three activities are non-synchronized tasks. They all occur in concurrency. The push frame is processed like a event message. All events (including push messages) are scheduled on a fixed priority arbitration scheme. This all implies, that a RPC payload, which contains the Remote Handler messages, is ordered in the way of the message occurrence. They are not sorted in terms of the message type. Please refer to the transmit trigger scheme. Still the order of read responses is kept by the Embedded Ethernet block as well as the remote Handler.

4.7.5.2 Transmit Trigger Scheme

The following trigger schemes are available and fully controllable via configuration either within the Embedded Ethernet block or the Remote Handler.

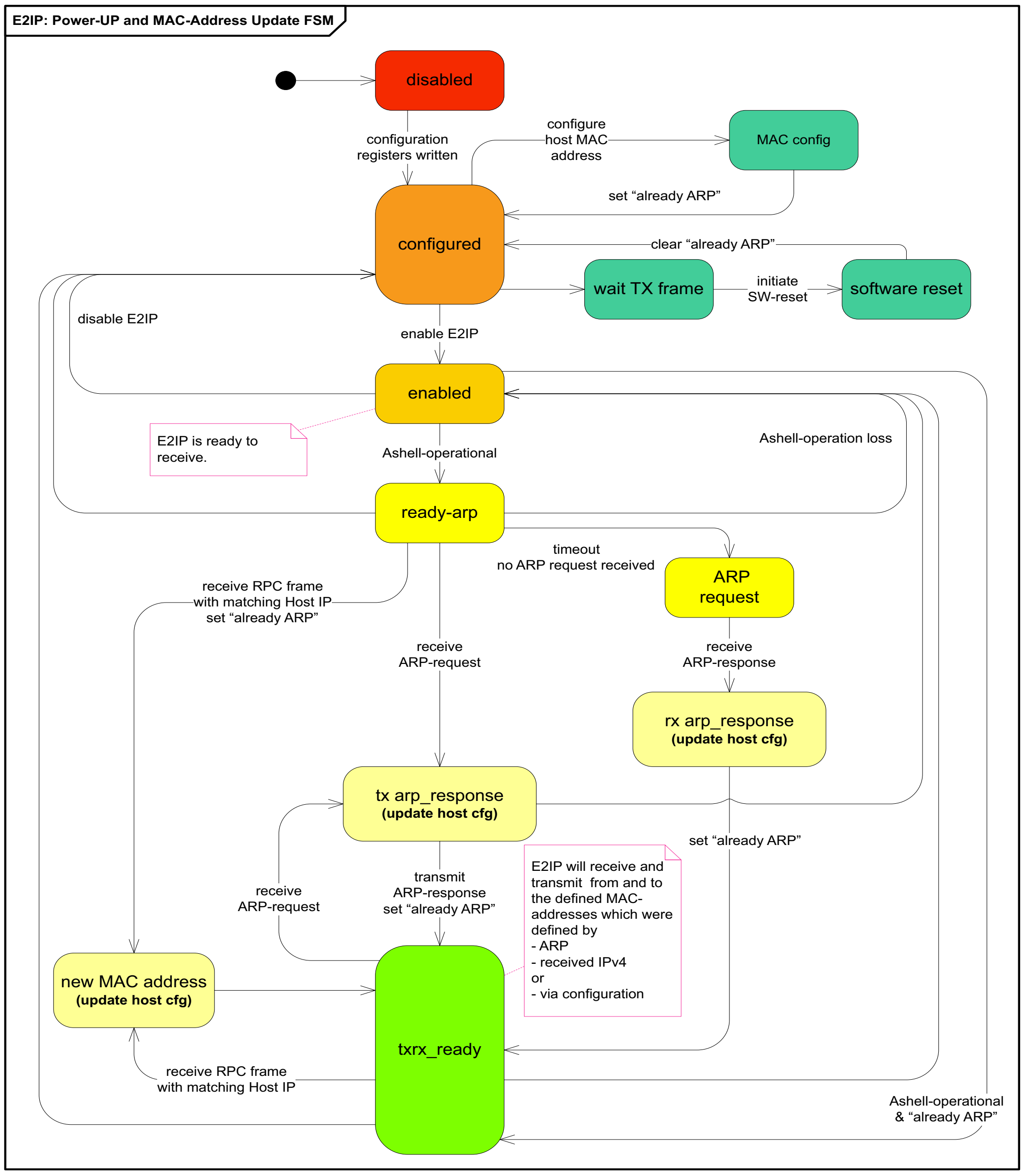
1. 1:1 communication between the host CPU and Indigo2
   The last Remote Handler message, which belong to a Ethernet frame RPC payload, is marked with a ‘last’ flag internally. When this ‘last’ message was processed by the Remote Handler, it will trigger the reply of a RPC payload. Then the Ethernet frame will be transmitted to the corresponding host CPU.
2. Periodical update of host CPU
   The Embedded Ethernet block will trigger the transmission of a RPC payload, if a defined time has been elapsed. This periodic interval will restart autonomously, after the trigger has been generated.
3. Threshold based update of host CPU
   The Embedded Ethernet block will trigger the transmission of a RPC payload, if a defined number of Remote Handler messages are available within the transmit buffers. The Embedded Ethernet block will automatically trigger the transmission, if the RPC payload has reached the maximum among of Remote Handler messages.
4. Request-based update of host CPU
   The host CPU itself is able to trigger the transmission. This is a SW trigger.
5. Message-based update of the host CPU.
   Each Remote Handler event or push message may configured to trigger the transmission of the already collected Remote Handler messages within the transmit buffer.

In general it is possible, that a single trigger will initiate more than one Ethernet frame.

4.7.5.3 Host MAC Address Exchange

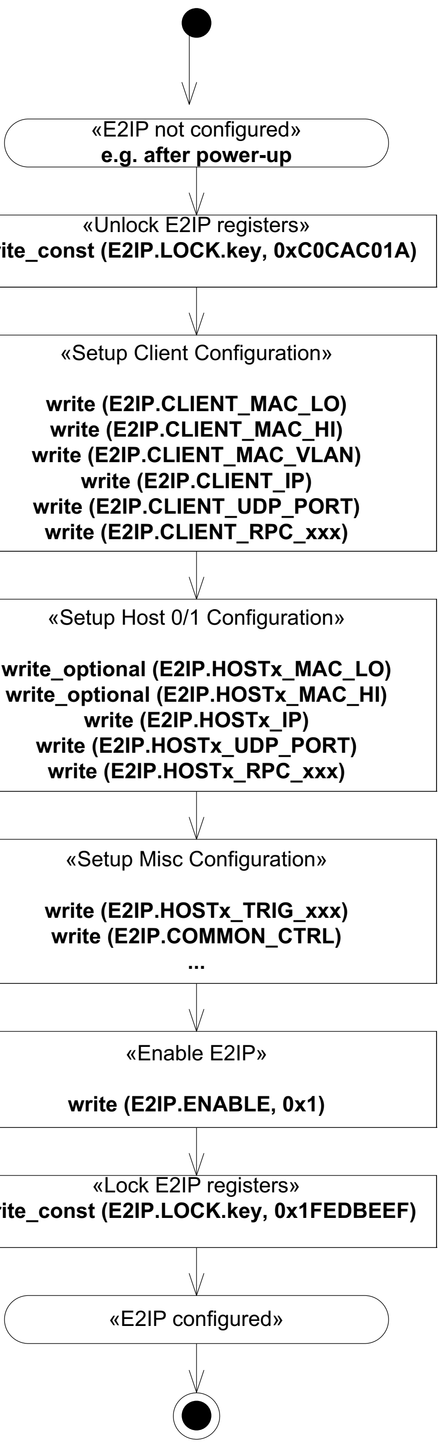
In general the Embedded Ethernet block will not being configured for the host CPU MAC addresses. The host CPU MAC address will be communicated by either the address resolution protocol (ARP) or by every other valid IP frame from the predefined host CPU. This also implies IP frames, which are received but not address to the Indigo2. The Embedded Ethernet block will not transmit any Ethernet frame to a host CPU until the MAC address has been updated by the above methods.

The following figure shows how the MAC address is being updated, as well the stage in which the Embedded Ethernet block is ready to transmit and receive.


E2IP: Power-UP and MAC-Address Update FSM
disabled
configuration registers written
configure host MAC address
MAC config
set "already ARP"
configured
clear "already ARP"
wait TX frame
initiate SW-reset
software reset
disable E2IP
enable E2IP
enabled
E2IP is ready to receive.
Ashell-operational
Ashell-operation loss
ready-arp
timeout no ARP request received
ARP request
receive RPC frame with matching Host IP set "already ARP"
receive ARP-request
receive ARP-response
rx arp_response (update host cfg)
tx arp_response (update host cfg)
set "already ARP"
receive ARP-request
transmit ARP-response set "already ARP"
E2IP will receive and transmit from and to the defined MAC-addresses which were defined by - ARP - received IPv4 or - via configuration
new MAC address (update host cfg)
txrx_ready
receive RPC frame with matching Host IP
Ashell-operational & "already ARP"

4.7.5.4 Power-Up Initialization

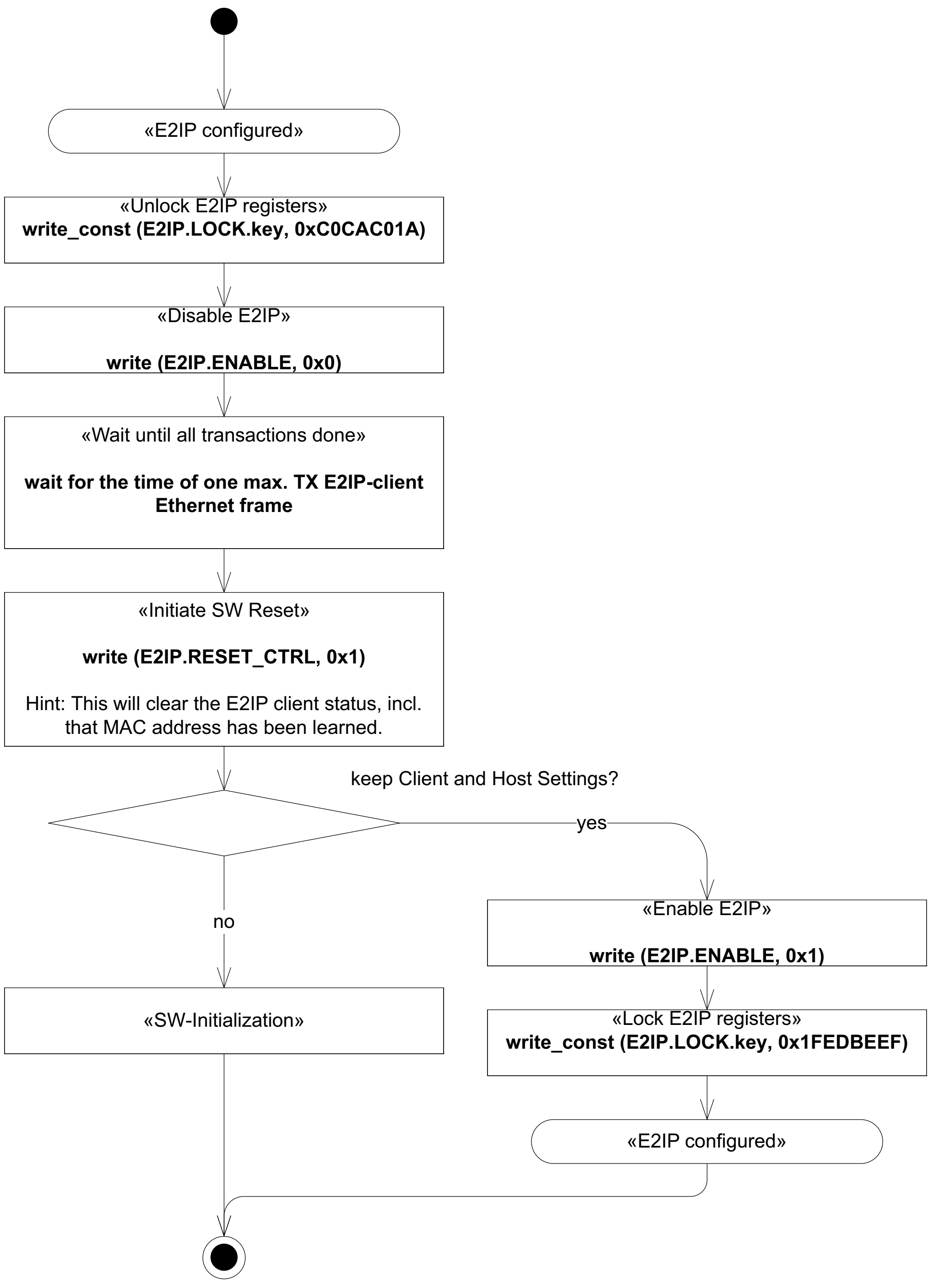
The following picture defines the minimal set of register, which need to be configured after power-up.


«E2IP not configured»
e.g. after power-up
«Unlock E2IP registers»
write_const (E2IP.LOCK.key, 0xC0CAC01A)
«Setup Client Configuration»
write (E2IP.CLIENT_MAC_LO)
write (E2IP.CLIENT_MAC_HI)
write (E2IP.CLIENT_MAC_VLAN)
write (E2IP.CLIENT_IP)
write (E2IP.CLIENT_UDP_PORT)
write (E2IP.CLIENT_RPC_xxx)
«Setup Host 0/1 Configuration»
write_optional (E2IP.HOSTx_MAC_LO)
write_optional (E2IP.HOSTx_MAC_HI)
write (E2IP.HOSTx_IP)
write (E2IP.HOSTx_UDP_PORT)
write (E2IP.HOSTx_RPC_xxx)
«Setup Misc Configuration»
write (E2IP.HOSTx_TRIG_xxx)
write (E2IP.COMMON_CTRL)
...
«Enable E2IP»
write (E2IP.ENABLE, 0x1)
«Lock E2IP registers»
write_const (E2IP.LOCK.key, 0x1FEDBEEF)
«E2IP configured»

NOTE When Indigo2 is started in Bootmode 2 or Bootmode 3 the E2IP block is already configured. (see Application Note “MB88F33X 'INDIGO2(-X)' Using Botmode 2 & 3). The reconfiguration procedure has to be used for changing the configuration.

4.7.5.5 Software Reset Procedure or Reconfiguration Procedure

The following picture defines the procedure to apply software reset.


«E2IP configured»
«Unlock E2IP registers»
write_const (E2IP.LOCK.key, 0xC0CAC01A)
«Disable E2IP»
write (E2IP.ENABLE, 0x0)
«Wait until all transactions done»
wait for the time of one max. TX E2IP-client Ethernet frame
«Initiate SW Reset»
write (E2IP.RESET_CTRL, 0x1)
Hint: This will clear the E2IP client status, incl. that MAC address has been learned.
keep Client and Host Settings?
yes
no
«Enable E2IP»
write (E2IP.ENABLE, 0x1)
«SW-Initialization»
«Lock E2IP registers»
write_const (E2IP.LOCK.key, 0x1FEDBEEF)
«E2IP configured»

4.7.6 Embedded Ethernet Register Overview

Table 4-9: Registers Overview

Base Address(es)	**Instance no 0: BASEADDR="00023000"**	
Absolute Address	**Register Name**	**Register Description**
BASEADDR + 0x0000	LOCK	Key Register for Lock and Unlock
BASEADDR + 0x0004	LOCK_STAT	Lock Status Register
BASEADDR + 0x0008	ENABLE 🔒	Enable Register
BASEADDR + 0x000C	RESET_CTRL 🔒	Reset Control Register
BASEADDR + 0x0010	CLEAR_CNT	Clear Error Counter Register
BASEADDR + 0x0014	RX_CNT_0	RX Error Counter 0 (total of dropped frames)
BASEADDR + 0x0018	RX_CNT_1	RX Error Counter 1
BASEADDR + 0x001C	RX_CNT_2	RX Error Counter 2
BASEADDR + 0x0020	RX_CNT_3	RX Error Counter 3
BASEADDR + 0x0024	RX_CNT_4	RX Error Counter 4
BASEADDR + 0x0028	TX_CNT_0	Transmit Counter Register 0
BASEADDR + 0x002C	TX_CNT_1	Transmit Counter Register 1
BASEADDR + 0x0030	SW_FLUSH	Software Transmit Trigger Register
BASEADDR + 0x0034	HOST0_TRIG_CTRL 🔒	Transmit Trigger for Host 0 Register
BASEADDR + 0x0038	HOST0_TRIG_VAL 🔒	Trigger Values for Host 0 Register
BASEADDR + 0x003C	HOST1_TRIG_CTRL 🔒	Transmit Trigger for Host 0 Register
BASEADDR + 0x0040	HOST1_TRIG_VAL 🔒	Trigger Values for Host 0 Register
BASEADDR + 0x0044	IRQ_EN	E2IP Interrupt Enable
BASEADDR + 0x0048	Reserved	Do not modify
BASEADDR + 0x004C	COMMON_CTRL 🔒	Common Control/Mode Register
BASEADDR + 0x0050	TX_RX_TARGETID 🔒	APIX Channel Destination Register
BASEADDR + 0x0054	TX_CFG 🔒	Transmitter Config Register
BASEADDR + 0x0058	CLIENT_MAC_LO 🔒	E2IP Client MAC Address Register
BASEADDR + 0x005C	CLIENT_MAC_HI 🔒	E2IP Client MAC Address Register
BASEADDR + 0x0060	CLIENT_MAC_VLAN 🔒	E2IP Client MAC VLAN Register
BASEADDR + 0x0064	CLIENT_IP 🔒	E2IP Client IP Address Register
BASEADDR + 0x0068	CLIENT_UDP_PORT 🔒	E2IP Client UDP Port Register
BASEADDR + 0x006C	CLIENT_RPC_MSGID 🔒	E2IP Client RPC Message-ID Register
BASEADDR + 0x0070	CLIENT_RPC_ID 🔒	E2IP Client RPC ID Register
BASEADDR + 0x0074	CLIENT_RPC_MISC 🔒	E2IP Client RPC Header Register
BASEADDR + 0x0078	HOST_MAC_VLD	Host MAC Address Validation Register
BASEADDR + 0x007C	HOST0_MAC_LO 🔒	E2IP Host 0 MAC Address Register
BASEADDR + 0x0080	HOST0_MAC_HI 🔒	E2IP Host 0 MAC Address Register
BASEADDR + 0x0084	HOST0_IP 🔒	E2IP Host 0 IP Address Register
BASEADDR + 0x0088	HOST0_IP_TTL 🔒	E2IP Host 0 IP Time-to-live Register
BASEADDR + 0x008C	HOST0_UDP 🔒	E2IP Host 0 UDP Port Register
BASEADDR + 0x0090	HOST0_RPC_MSGID 🔒	E2IP Host 0 RPC Message-ID Register
BASEADDR + 0x0094	HOST0_RPC_ID 🔒	E2IP Host 0 RPC ID Register
BASEADDR + 0x0098	HOST0_RPC_MISC 🔒	E2IP Host 0 RPC Header Register
BASEADDR + 0x009C	HOST1_MAC_LO 🔒	E2IP Host 1 Mac Address Register
BASEADDR + 0x00A0	HOST1_MAC_HI 🔒	E2IP Host 1 Mac Address Register
BASEADDR + 0x00A4	HOST1_IP 🔒	E2IP Host 1 IP Address Register
BASEADDR + 0x00A8	HOST1_IP_TTL 🔒	E2IP Host 1 IP Time-to-life Register
BASEADDR + 0x00AC	HOST1_UDP 🔒	E2IP Host 1 UDP Port Register
BASEADDR + 0x00B0	HOST1_RPC_MSGID 🔒	E2IP Host 1 RPC Message-ID Register
BASEADDR + 0x00B4	HOST1_RPC_ID 🔒	E2IP Host 1 RPC ID Register
BASEADDR + 0x00B8	HOST1_RPC_MISC 🔒	E2IP Host 1 RPC Header Register

Chapter 5: IRIS-MVL

5.1 General

The Iris-MVL unit is the central pixel processing unit of the Indigo2 device and consists of three main interlinked functional blocks:

- Capture Engine
- Pixel Engine
- Display Engine

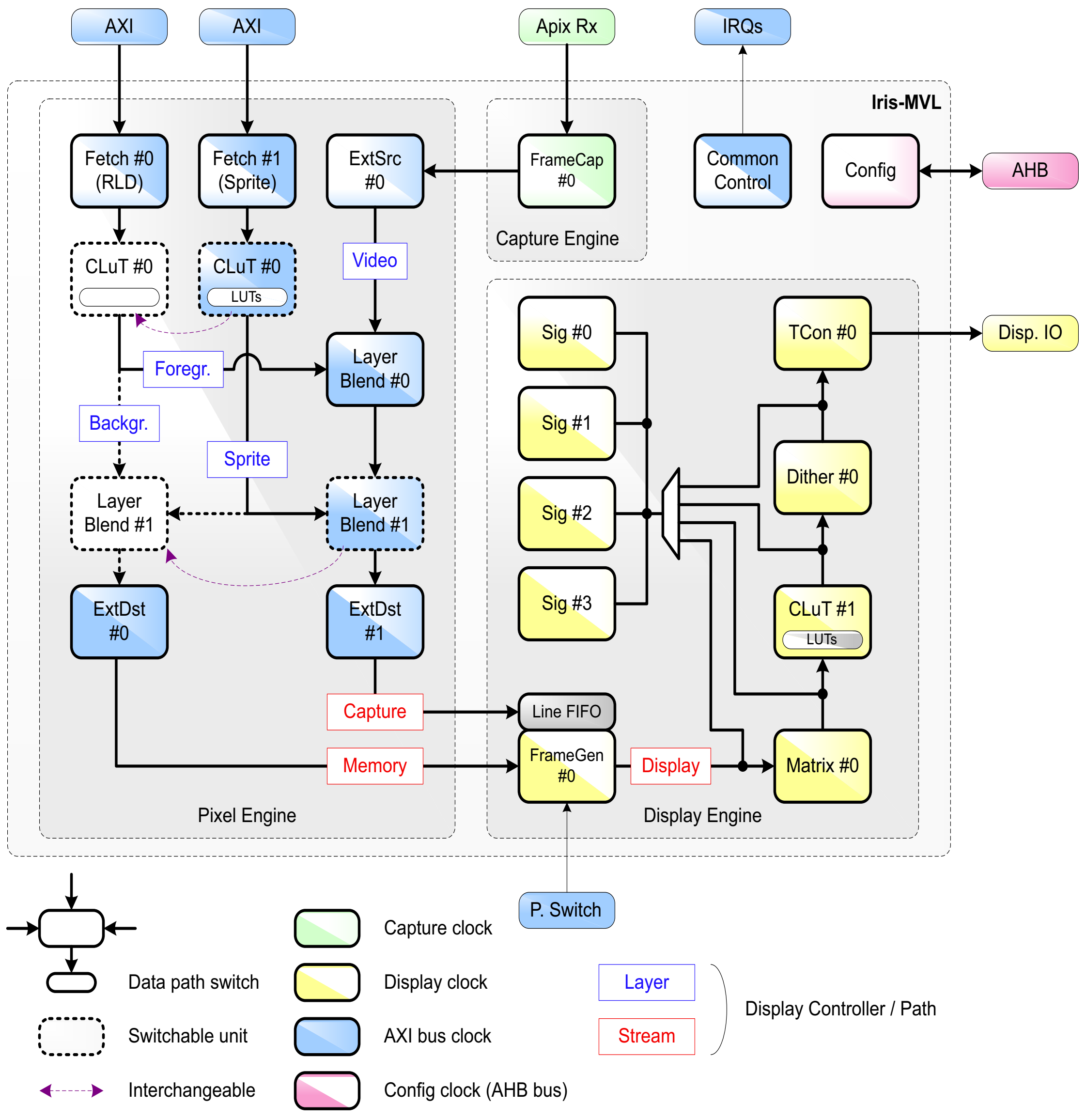


Figure 5-1: Iris-MVL Block Diagram

Iris-MVL can generate three display layers:

- Background or Foreground layer (Fetch#0)
- Sprite layer (Fetch#1)
- Video layer (FrameCap#0)

These layers are combined to two video streams by alpha blending:

- Memory stream (ExtDst#0)
- Capture stream (ExtDst#1)

These streams are combined with a constant color background by overlay to one

- Display stream (FrameGen#0)

The data paths for stream composition are partly configurable for different kind of use cases.

The following processing units exist:

- **FrameCap** - Receives data from an external video source and synchronizes it to the internal processing pipeline.
- **ExtSrc** - Implements some basic features for frame format conversion.
- **Fetch** - Reads a frame from a source buffer in memory and convert it to the internal processing format. Derivatives: (RLD) supports run-length decoding of compressed buffers, (Sprite) output layer is composed from up to 16 sprites.
- **CLuT** - For non-linear color corrections (e.g. gamma) or for color palettes (index formats).
- **Matrix** - For linear color transformations (e.g. standard color controls or color space conversion).
- **LayerBlend** - Can alpha blend a foreground layer onto the background display layer at a variable position.
- **ExtDst** - Prepare frame data for display output.
- **FrameGen** - Generates the display output timing and overlays the input stream on it.
- **Dither** - Enables display of full color range on panels with lower physical color resolution.
- **TCon** - Freely programmable timing generator to control column and row drivers of an external panel.
- **Sig** - Can compute a signature value for each display output frame and compare it against a pre-computed reference value in order to detect display of corrupt data.

The Iris-MVL unit can be operated in one of three modes:

- TTL mode
- RSDS mode
- LVDS mode

5.2 Features

5.2.1 Display Controller

- 1 display output stream (progressive scan).
- 1 display input stream (capture):
  - Video background layer from capture input.
  - 1 alpha blend-able sprite layer from memory.
  - 1 alpha blend-able foreground layer from memory (only if memory input stream is not used).
- 1 display input stream (memory):
  - Memory background layer.
  - 1 alpha blend-able sprite layer (only if not used for capture stream).

- Max. 144 MHz pixel clock (capture input and display output). **(**MB88F335 ‘Indigo2-S’ $\rightarrow$ **max. 35 MHz pixel clock = e.g., 960x 480@60 Hz)**
- Max. 1920 x 1024 pixels with 24-bit RGB display output.
- Support for single channel TTL or dual channel, RSDS and LVDS display output.
- Dual channel modes interleaved (even/odd column index) or split (left/right display half).
- 12 freely-programmable timing generators.
- Simultaneous operation of both input streams (overlay, seamless switching).
- Automatic synchronization of display to capture timing up to 10% difference in refresh rate (blanking adjustment).
- Vertical letterboxing (horizontal resolution of capture smaller than display).
- Up to 16 simultaneous sprites with individual size from 1 x 1 to 1920 x 1024 pixels.
- Pixel sizes:
  - Video layer: 1 .. 24 bpp.
  - Memory layers: 1, 2, 4, 8, 16, 24, 32 bpp (all packed in memory).
- Color formats: RGBA, Grey scale.
- Pixel formats: Individual size and position for each RGBA component (0..10 bits).
- Constant color for null-size components.
- Alpha options: per-pixel, constant (global), transparent color.
- Pre-multiply modes (memory layers): constant alpha on per-pixel alpha, per-pixel alpha on RGB, constant on RGBA.
- Run-length decompression for background / foreground layer.
- Clip and skip window (memory layers).
- Scan directions (memory layers): 90/180/270° rotation, horizontal/vertical flip.
- Color conversions (display output): Linear (RGB matrix) and non-linear (RGB look-up).
- Spatial and temporal dithering: 10 bits (virtual) to 5, 6, 7 or 8 bits (physical) per component.
- 4 signature unit for display check against reference (Sum or CRC).
- Panic mode to turn display off or switch display stream in case of signature violation.
- Alpha mask mode (enables color conversion and/or signature computation per pixel).

5.3 System Setup MB88F33x ‘Indigo2(-x)’

5.3.1 Clock Setup of Iris-MVL

The correct configuration of the system clock signals for the Iris-MVL unit is required to enable it to generate the display clock in its display engine subunit (dsp_clk), as well as the bit clock for the panel interface (bit_clk). These clock signals are internally generated based on the incoming vid_clk signal from the APIX unit, in conjunction with either a PLL or various clock dividers (whereby the PLL and dividers can also be bypassed). The following diagram shows the configuration path and shows the names of the corresponding registers and bitfields (format: register.bitfield) which control the clock configuration.

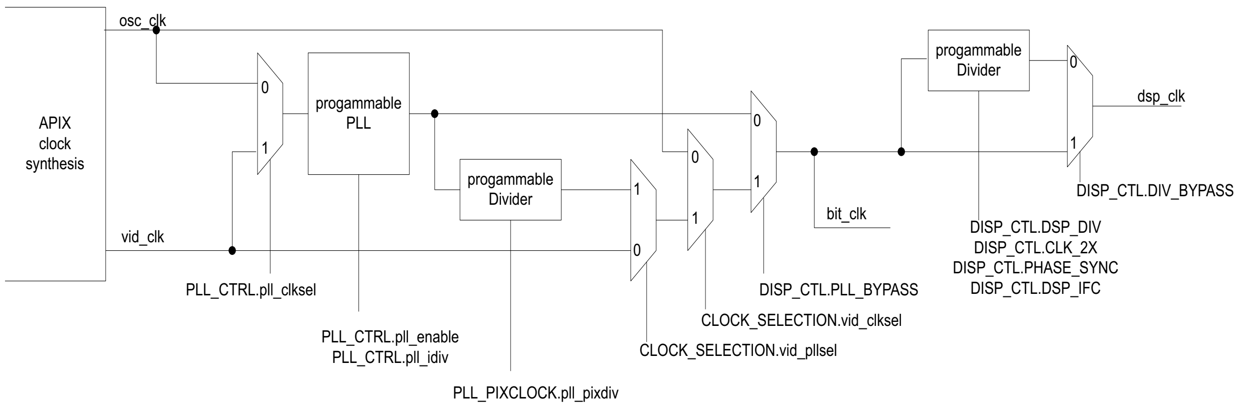


Figure 5-2: Display Clock generation

The clock signal configuration varies according to the panel interface/mode you intend to use (TTL, RSDS or LVDS). The table below summarizes the clock signal setup requirements for each of the panel interface types and modes.

Table 5-1: Clock Requirements for Display Interface Modes

Mode	bit_clk	dsp_clk	Comment
TTL single channel	2x pix_clk ... 600 MHz. Even integer multiple of pix_clk. Maximum is 32x pix_clk	2x pix_clk	Maximum pix_clk = 85MHz
RSDS single channel	2x pix_clk ... 600 MHz. Even integer multiple of pix_clk. Maximum is 32x pix_clk	2x pix_clk	Maximum pix_clk = 85MHz
RSDS dual channel	pix_clk ... 600 MHz. Integer multiple of pix_clk. Maximum is 16x pix_clk	pix_clk	Maximum pix_clk = 144MHz
LVDS single channel	7x pix_clk	pix_clk	Maximum pix_clk = 75MHz
LVDS dual channel	3.5x pix_clk	pix_clk	Maximum pix_clk = 144MHz

NOTE MB88F335 ’Indigo2-S’ only supports max 35MHz pix_clk.
RSDS, dual and LVDS is not implemented in MB88F335 ’Indigo2-S’

5.3.1.1 Pixel Clock

The pix_clk (pixel clock) is the pixel clock signal, whose frequency matches the selected display resolution.

5.3.1.2 Video Clock

The vid_clk (video clock) is the video clock generated by the APIX macro. This clock is the reference for all video clocks. If it is spread spectrum modulated all the clocks of the display subsystem will be spread spectrum modulated. The maximum clock frequency which can be generated is 170 MHz.
For setup of this clock see “2.4.5 Clock Synthesis”and “2.4.6 Clock Modulation / Spread Spectrum”.

5.3.1.3 Bit Clock

The bit_clk (bit clock) is the output signal shifting frequency clock signal in TTL and RSDS mode and the bit clock in LVDS mode. There are three possible ways to generate the bit_clk signal:

- use the vid_clk from the APIX clock generation unit or
- use the clock output from the PLL or
- divide the PLL clock and use this value

The three options are shown in the following diagrams.

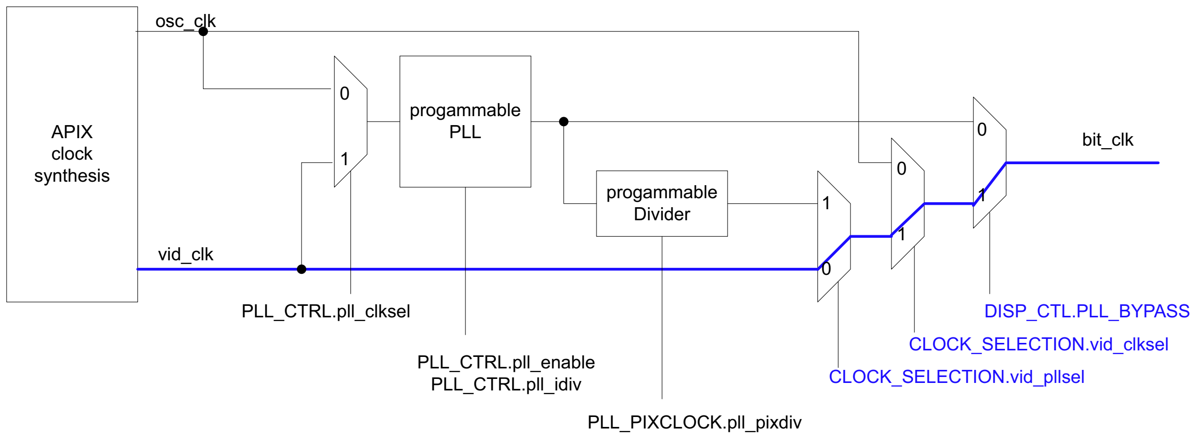


Figure 5-3: APIX clock (vid_clk) determines bit_clk frequency

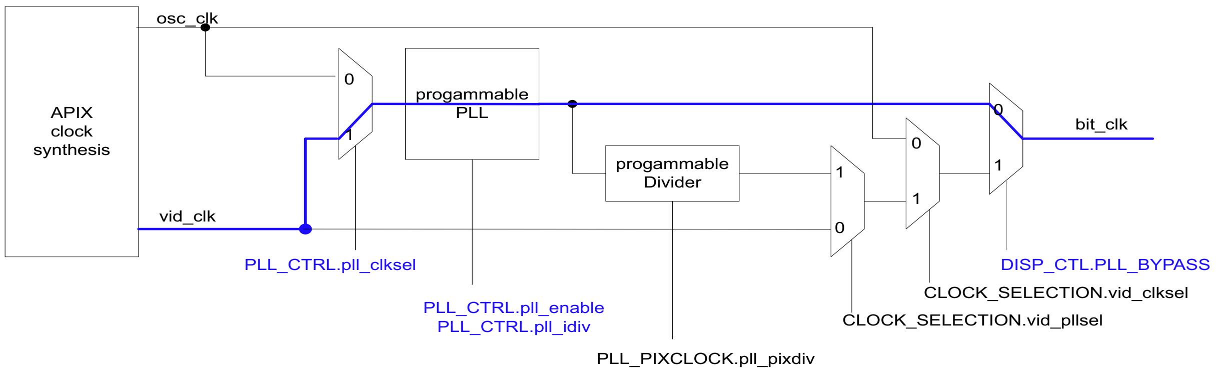


Figure 5-4: APIX clock and PLL determines bit_clk frequency

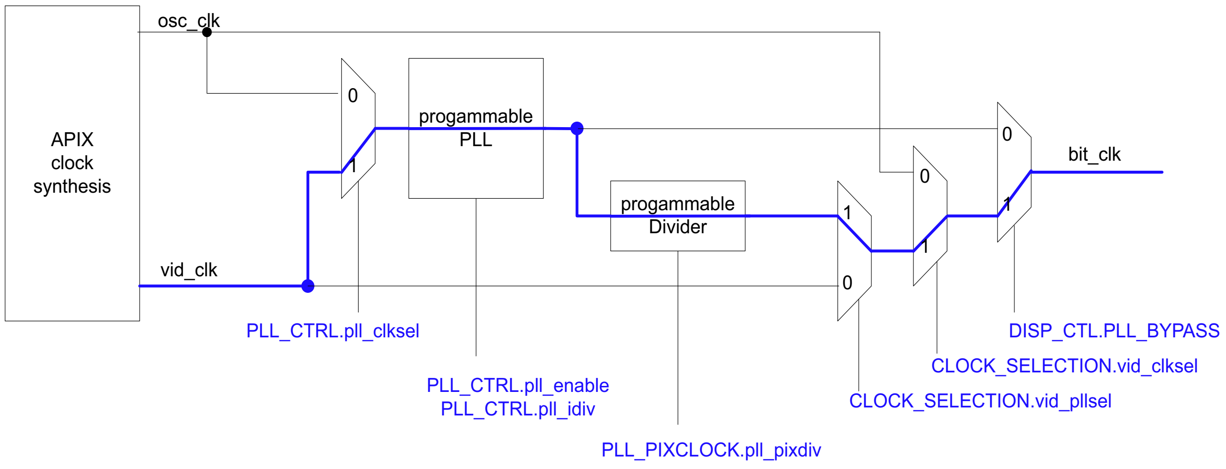


Figure 5-5: APIX clock and divided PLL determines bit_clk frequency

5.3.1.4 Display Clock

The dsp_clk (display clock) is the clock signal whose frequency determines the operation frequency of the Iris display subsystem. The dsp_clk is either the bit_clk or a divided version of the bit_clk. For TTL and RSDS mode the divider has to be an integer and can be freely selected up to the factor 16.
For LVDS the divider has to be 3.5. The value of 3.5 has to be set with a divider of 7 and by enabling the clock doubling.

The table below shows the register settings required for a specific display panel interface mode.

Table 5-2: Display Clock Setup

Mode	DISP_CTL. DSP_DIV	DISP_CTL. CLK_2x	DISP_CTL. PHASE_SYNC	DISP_CTL. LVDS_2X_MODE
TTL, RSDS	2 ... 15	0	0	0
LVDS single channel	6	1	1	0
LVDS dual channel	6	1	1	0

5.3.1.5 Iris Clock Control

The Iris graphics display controller must ‘know’ if the display clock (dsp_clk) is running at single or double pixel clock frequency. This in turn, depends on the selected display panel interface for which the clock frequency has to be set.

Table 5-3: Iris Clock Setup

Mode	dsp_clk	ClockCtrl DspClkDivide	Max. pix_clk
TTL single channel	2x pix_clk	DIV2	85 MHz
RSDS single channel	2x pix_clk	DIV2	85 MHz
RSDS dual channel	pix_clk	DIV1	144 MHz
LVDS single channel	pix_clk	DIV2	75 MHz
LVDS dual channel	pix_clk	DIV1	144 MHz

5.3.2 Configuring Display Output Pins (Multiplexing)

It is necessary to configure the display output pins (‘pin control’) correctly for the respective display modes. Start by ‘enabling’ the correct number of display pin pairs in the global pin multiplexer, i.e. the display pin pairs must be set to ‘display mode’.

NOTE In the following descriptions, [I] is the variable interface number and [O] is the variable output pair number.

The disp[I]p[O]_mode and disp[I]n[O]_mode in the DISP[I]_[O]_CTL registers have to be set to DISP[I]_[O]P or DISP[I]_[O]N. The following table shows which (and how many) display pin pairs required for the respective display interface mode.

Table 5-4: Display Output Pin Pairs

Mode	Pair Count	Display Pairs to use
TTL single channel (24 bit)	13	DISP0_0 ... DISP0_12
TTL single channel (18 bit)	10	DISP0_3 ... DISP0_12
RSDS single channel (24 bit)	13	DISP0_0 ... DISP0_12
RSDS single channel (18 bit)	10	DISP0_3 ... DISP0_12
RSDS dual channel (2x 24 bit)	26	DISP0_0 ... DISP0_12 and DISP1_0 ... DISP1_12
RSDS dual channel (2x 18 bit)	20	DISP0_3 ... DISP0_12 and 10 pairs out of DISP1_0 ... DISP1_12
LVDS single channel (24 bit)	5	DISP0_8 ... DISP0_12
LVDS single channel (18 bit)	4	DISP0_8 ... DISP0_11
LVDS dual channel (2x 24 bit)	10	DISP0_3 ... DISP0_12
LVDS dual channel (2x 18 bit)	8	DISP0_4 ... DISP0_11

The next step is to select a clock output pair. In single channel mode, one output pair has to act as a clock output. In dual channel mode, two output pairs have to act as clock outputs. The clock output pairs, can be selected using the D[I]_CLKSEL_[O] fields of the DISP[I]_PN[O]_CTL registers.

The remaining pairs are used for the pixel data output, whereby the mapping of the pixel data delivered by these can be configured in the TCON.

5.3.3 Shifting Output Data for Display

In TTL or RSDS mode, the edges of the data and clock signal outputs can be shifted out at any rising or falling edge of the bit_clk clock signal. The display output can be shifted out of the panel interface on any multiple of the dsp_clk clock signal and by any multiple of the bit_clk clock signal.

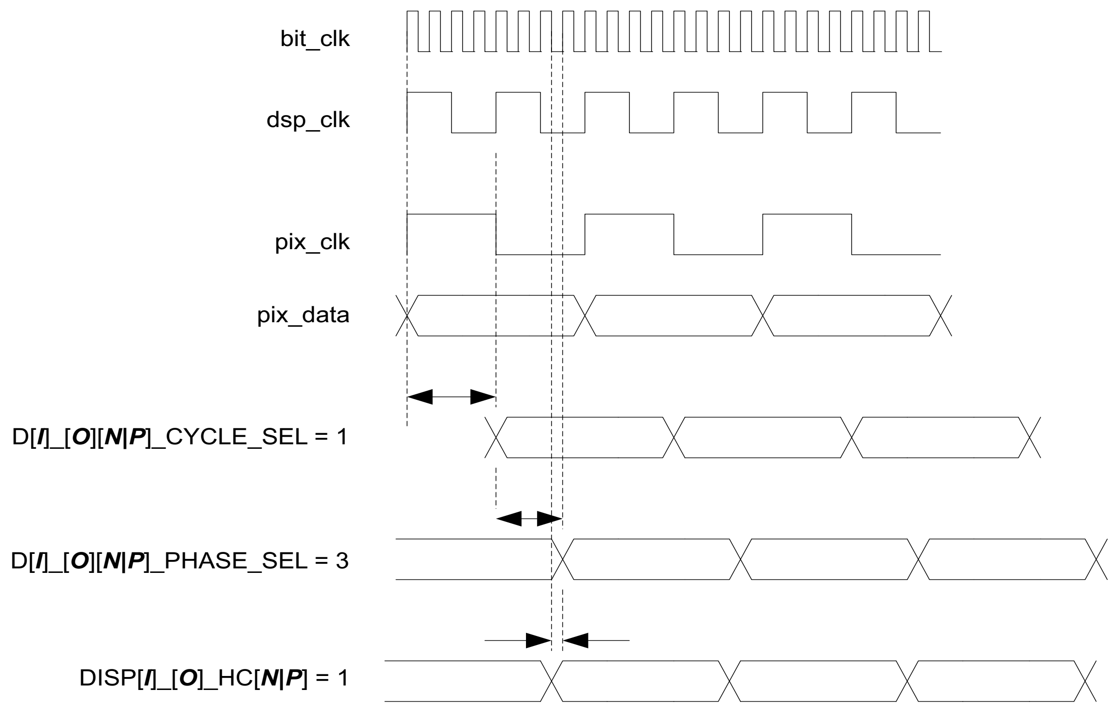

bit_clk
dsp_clk
pix_clk
pix_data
D[*I*]_[*O*][*N*|*P*]_CYCLE_SEL = 1
D[*I*]_[*O*][*N*|*P*]_PHASE_SEL = 3
DISP[*I*]_[*O*]_HC[*N*|*P*] = 1

Figure 5-6: Output Shifting Example

- Shifting the panel data by one dsp_clk cycle is configured with D[I]_[O][N|P]_CYCLE_SEL = 1.
- Shifting the panel data by three bit_clk cycles is configured with D[I]_[O][N|P]_PHASE_SEL = 3.
- Shifting the panel data by -0.5 bit_clk cycles is configured with DISP[I]_[O]_HC[N|P] = 1.

Whereby [I] is the interface number, [O] is the output pair number and [N|P] defines if the N or P output is used.

The D[I]_[O]P_CYCLE_SEL, D[I]_[O]P_PHASE_SEL and DISP[I]_[O]_HCP registers define the shifting of the output pair data in RSDS mode.

In LVDS mode, only a shifting value of -0.5 can be enabled (using DISP[I]_[O]_HCP).

5.4 IRIS-MVL Functions

5.4.1 Use Cases

The following applications are supported:

- **Display of content in memory**
  A display buffer and multiple sprites are read from memory (Fetch units via AXI), combined and complemented with a freely programmable display timing.
- **Display of direct capture input**
  An external video source (typically driven by an APIX receiver module in the system) is synchronized to an output timing with similar vertical refresh rate and directly displayed.

Both applications can be combined in three different basic configurations:

Table 5-5: Basic Configurations

	Capture Stream	Memory Stream	Color Palette
A	- Video layer - Sprite layer (blended)	- Background layer	Background or sprite layer
B	- Video layer	- Background layer - Sprite layer (blended)	Background or sprite layer
C	- Video layer - Foreground layer (blended) - Sprite layer (blended)		Foreground or sprite layer

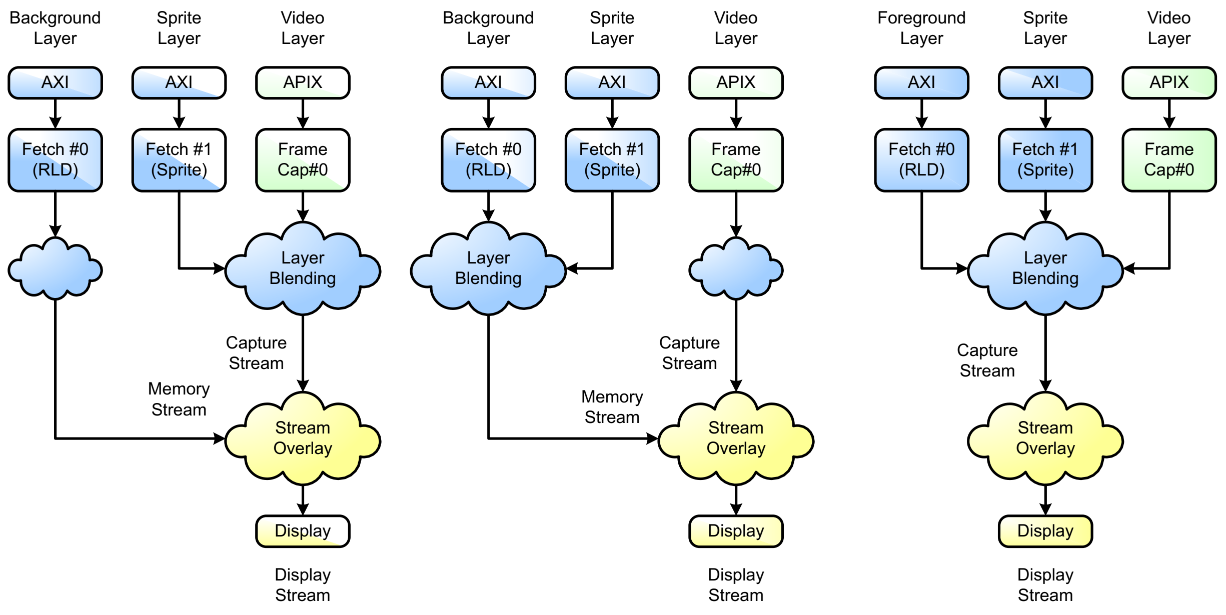


Figure 5-7: Basic Configurations (A, B, C)

The two input streams can be combined with a constant color layer by overlay for each configuration:

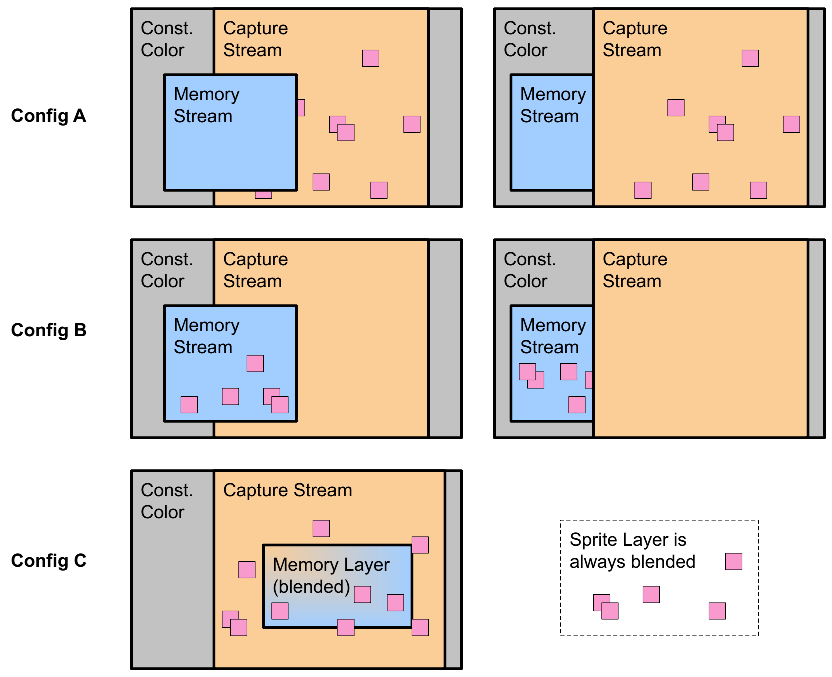


Figure 5-8: Display Stream Layout

Any of the streams in any use case shown above can also be turned off.

Both display streams operate independently and simultaneously. Also the different display mode for each base configuration can be changed during operation. This particularly makes it possible to setup the following use case:

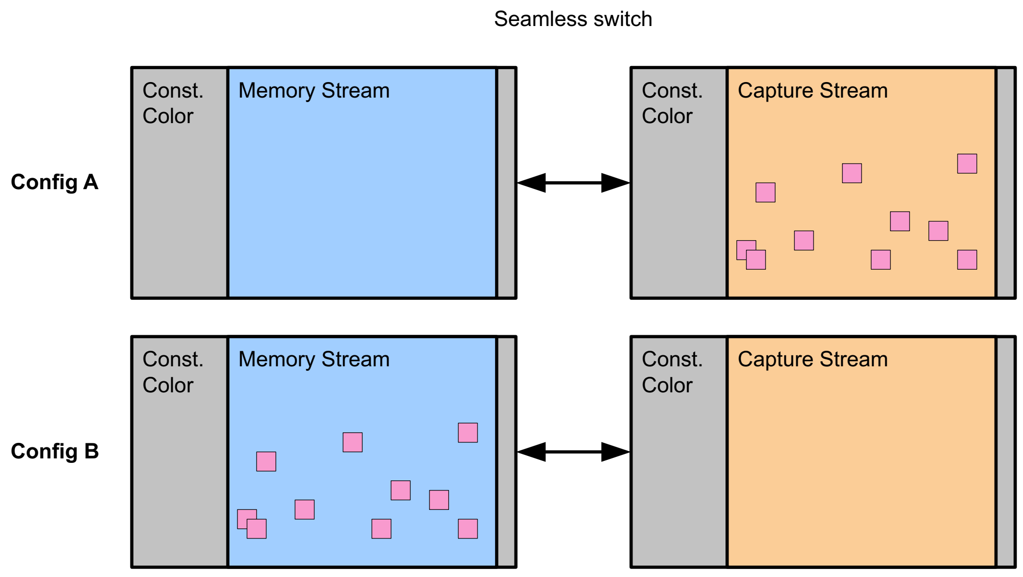


Figure 5-9: Display Stream Switching

Seamless means, that from one output frame to the next one display changes between memory and capture content without any corrupt or black intermediate frames becoming visible.

This makes it possible, for example, to show content from memory while a video link is established and synchronized. Once video and display timings are locked, the software can switch to capture display. Also it can immediately switch back to memory content in case of some error condition (e.g. capture link is lost).

5.4.2 Limitations

5.4.2.1 Display Controller

Table 5-6: Display Controller Limitations

Display Output Video Mode	
Pixel clock frequency (*pix_clk*) Spread spectrum modulation amplitude Modulation frequency	1.00 .. 144 MHz max 5% center-spread min 44 kHz
Active area width (*hact*)	320 .. 1920 pixels
Active area height (*vact*)	160 .. 1024 pixels
Horizontal blanking width (*hblank*) Front porch / Sync pulse / Back porch	5% ... 40% of hact min 1 / 1 / 1 pixel
Vertical blanking height (*vblank*) Front porch / Sync pulse / Back porch	2% ... 30% of *vact* min 1 / 1 / 1 pixel
Width of left and right display side in dual channel split mode	max. 1280 pixels
Capture Input Video Mode	
Pixel clock frequency (virtual clock) [a]	*pix_clk* ± 10%
Active area width	60 .. 100% of *hact*
Active area height	= *vact*
Horizontal blanking width	60 .. 100% of *hblank*
Vertical blanking height	= *vblank*

a) Does not correspond to the data clock of the input interface (= capture clock) but to the pixel clock of the transmitted video mode as it would appear on the display output for the same mode.

This makes it possible to setup, for example, the following modes:

- 1600x600@100Hz with reduced blanking according to APIX standard (15% hor, 4% vert).
- 1920x768@60Hz with reduced blanking (15% hor, 4% vert).
- 1280x1024@60Hz with standard blanking (25% hor, 15% vert).
- 800x600@60Hz video to 1280x600@60Hz display (letterboxed).

5.4.3 Basic Functions

5.4.3.1 Register Access

All processing units are setup by addressable 32-bit configuration registers. These can be read and written via the AHB slave port.

5.4.3.2 Shadow Registers

All configuration registers, which it makes sense to change for different kinds of operations, are shadowed. The software can write to these shadows without having an immediate effect.

The software can generate a shadow load token at certain points in the processing pipeline. Such a token is automatically synchronized to the start of the next frame and will consecutively load shadow registers into the active configuration for all subsequent units until a token end-point is reached. This is signalled by a shadow load interrupt to the software, which in response can start to write another setup into the shadows.

For display operation, it makes it possible to update settings, which are spread over several processing units, consistently for the same output frame.

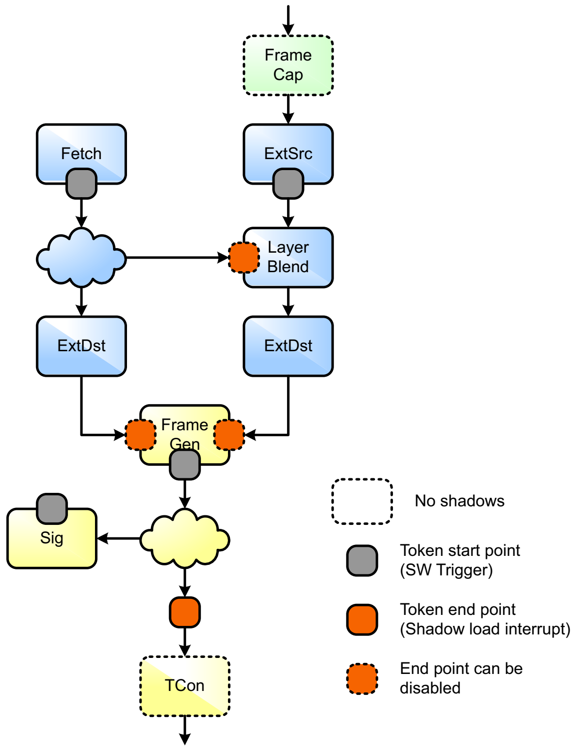

Frame Cap
Fetch
ExtSrc
Layer Blend
ExtDst
ExtDst
Frame Gen
Sig
TCon
No shadows
Token start point (SW Trigger)
Token end point (Shadow load interrupt)
End point can be disabled

Figure 5-10: Shadow Load Tokens

For processing paths with more than one source, the software trigger can be consistently set for all Fetch and ExtSrc units by a dedicated synchronizer unit of the Pixel Engine. The software needs to write one trigger bit only in that case.

The Layer Blend unit has an optional end-point at its secondary input (= foreground layer that is blended on the primary input). By this two different software threads can asynchronously change settings for different display layers without interfering. If enabled a token on the secondary input will end and generate a shadow load interrupt, while those from the primary input are passed up to the next end-point. Independent from that it can be selected if the shadows of the LayerBlend unit itself are loaded with the primary or secondary token.

The Frame Generator has both an optional end-point at its inputs and a start-point. When the end-points are enabled, the start-point must be used to generate a token for the following pipeline. This makes it possible to separate front and back part setup of the display path from a software point of view. Also it is required when the Generator is in free-running mode without frame input data. Independent from that it can be selected if the shadows of the FrameGen itself are loaded with the primary, secondary or its own token.

The Signature Unit is a special case, because it is not an active part of the processing pipeline, but only monitoring it. It can detect a shadow load token in the pipeline or generate its own token to separate the signature setup from the rest (useful for safety aspects of software architectures). However, it is not a token end-point, consequently its shadow load interrupt will only occur when a shadow load was actually done and not in general when a token from the pipeline has been received.

5.4.3.3 Register Locking

For safety reasons, access to certain registers of Common Control and Signature Unit is locked by default. A constant key value must be written to a dedicated register in order to unlock respectively re-lock access.

This minimizes the probability that the software will disable on a malfunction, for example, the signature check of display output by accident.

Note, that this is not a security feature to prevent unauthorized access to those registers.

5.4.3.4 Interrupt Controller

Iris provides a built-in interrupt controller with the following features for all relevant hardware events:

- Enable bit (mask)
- Status bit (set by an hardware event)
- Preset bit (can be used by the software to set status)
- Clear bit (used by the software to reset the status)

The trigger signals and relevant status signals are provided to the MB88F33x 'Indigo2(-x)' Interrupt controller (please refer to chapter "2.8 Interrupt Controller"). The built-in interrupt controller can be used for polling of interrupt status flags.

5.4.3.5 Power Optimization

To reduce power consumption, the MB88F33x 'Indigo2(-x)' may reduce the AXI clock frequency. In case of display operation, however, this must not violate the Display Controller Limitations.

If this is not possible, alternatively Clock Throttling can be activated for the Pixel Engine clock domain individually for both the capture and memory stream, dividing the effective clock frequency by a fixed ratio.

For power considerations see also chapter "7.3.1 VDD Supply Current (Note 1)"

5.4.4 Display Controller

The Display Controller can generate the output timing for one display (display stream) with image content read from any memory resource (memory stream) and directly from a captured video timing (capture stream).

5.4.4.1 Display Stream

Independent of any input data, a free-running display timing with constant colored active area, horizontal and vertical blanking intervals and synchronization pulses can be generated. Both the memory and the capture input stream can be overlaid with any size and at any position inside the output frames, including the blanking areas.

The refresh rate of the output timing can be synchronized to the capture stream by dynamically adjusting the blanking size.

Initial synchronization can be done in background while the memory stream is displayed. Once synchronized a seamless switch between memory and capture stream display is possible. Alternatively a fast synchronization mode is supported to speed up the initial procedure to a few frames only.

In order to directly control row and column drivers of a panel, which require more complex synchronization signals than just a horizontal and vertical sync pulse, freely programmable timing generators can be used. Output signal generation is capable of driving display IO cells in a system in different modes.

Pixel colors of the active output area can be adjusted with a linear transformation by RGB matrix and offset (e.g. brightness, contrast, saturation) and a non-linear transformation by RGB look-up (e.g. gamma correction).

By spatial or temporal dithering a virtual color resolution of 10 bits per channel can be achieved on panels with physical resolutions from 5 to 8 bit.

Color transformations and dithering can be limited to either individual pixels only by using a bit mask in memory or to areas that correspond to specific foreground or background layers (alpha masking).

Related topics: Frame Generator, Color Matrix, Color Lookup Table, Dither Unit, Timing Controller.

5.4.4.2 Memory Stream

The memory input stream reads frame buffer content re iteratively from any resource in a system that can be addressed via AXI bus. It is connected to the display output stream.

It can operate in single or double (front and back) buffer mode. Modifications of a single buffer can be synchronized to the vertical blanking interval of the display in order to avoid tearing artifacts.

Image content of the memory stream is composed from the following layers by alpha blending:

- Background Layer
  Supports all common Iris buffer formats. Can optionally be run-length encoded (RLE) or can use a color palette (Note: only if palette is not used for sprite layer) to save memory size and bandwidth.
- Sprite Layer (Note: available only if sprite layer is not used for capture input stream)
  Is composed from up to 16 different images, laid onto a transparent background. The result is alpha blended onto the background layer (note that sprites cannot be blended onto other sprites, a fixed z-order is applied instead). Sprites can use up to 16 different color palettes (Note: only if palette is not used for background layer) to save memory size and bandwidth.

Related topics: “5.4.5.1 Fetch Unit”, “5.4.5.9 Color Lookup Table”.

5.4.4.3 Capture Stream

The capture input stream is driven by an external source, from which it must receive a consecutive sequence of frames. Most typically this is some video interface or APIX receiver module. It is connected to the display output stream.

Image content of the capture stream is composed from the following layers by alpha blending:

- Video Layer
  Builds the background of the capture stream and contains the frames as received on the capture input interface.
- Foreground Layer (Note: only available if memory input stream is not used)
  Supports all common Iris buffer formats in memory. Can be alpha blended with any size and at any position onto the video layer.
- Sprite Layer (Note: available only if sprite layer is not used for memory input stream)
  Same capabilities as sprite layer for the memory stream.

Related topics: Frame Capture Unit, External Source Interface, Layer Blend Unit.

5.4.4.4 Safety Features

In order to detect output of corrupt images, a signature value can be computed for each frame and checked against a reference value.

The signature can be computed at four different points in the display processing path, before or after certain color transformations. This allows a balance between safety aspects and complexity of the reference value determination.

The computation can be limited to rectangular sub areas or to any shaped regions using a bit mask.

Four different rectangular areas can be monitored against reference values simultaneously.

In response to a signature violation the following is possible:

- Interrupt signal to the software.
- The hardware switches autonomously and instantly the monitored region of the corresponding Signature Unit to a constant color.
- The hardware switches autonomously and instantly the display mode of the Frame Generator. This allows different scenarios like disabling or enabling certain input streams.

Settings that control these feature are protected against getting modified accidently (see Register Locking). This prevents display of corrupt content in case of major malfunction of a system.

It helps to full-fill the requirements of safety standards (e.g. Automotive Safety Integrity Level, ASIL).

Related topics: Frame Generator, Signature Unit.

5.4.5 Processing Units

5.4.5.1 Fetch Unit

The Fetch Unit is the interface between the AXI bus for source buffer access and the internal pixel processing pipeline, which is 30-bit RGB plus 8-bit Alpha.

5.4.5.1.1 AXI Settings

Access to the AXI bus can be configured in terms of burst length that is used for transactions.

5.4.5.1.2 Source Buffer Formats

32-bit base address.

Any buffer dimension between 1 and 16,384 pixels in both horizontal and vertical directions.

Total size of one pixel in memory: 1, 2, 4, 8, 16, 24 or 32 bits per pixel (bpp).
All of these are packed with a memory and bandwidth utilization of 100%.

Stride (= offset in bytes between two lines in memory) can be setup independent from dimension and pixel size.

Alignment restrictions:

– 32 and 24 bpp: Base address and stride must be multiple of 4 bytes
– 16 bpp: Base address and stride must be multiple of 2 bytes
– others: any byte alignment allowed

5.4.5.1.3 Pixel Formats

The width of color components as stored in memory can be setup individually to any value between

– 0 and 10 bits for Red, Green and Blue
– 0 and 8 bits for Alpha

Any bit position within the pixel word can be configured individually for each component. There are no restrictions regarding sequence or overlaps.

Example for RGB565 (16 bpp):

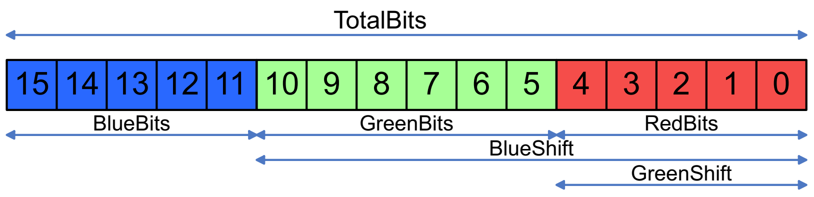


Figure 5-11: Generic Pixel Format

The value for components that are setup to null size is taken from a programmable constant color.

Components which are smaller than the internal processing width are up-scaled accordingly in order to keep black (input 0 always maps to output 0) and white level (input max code always maps to output max code).

Gray scale in all bit widths is supported by replicating pixel data into R, G and B component.

Depending on the subsequent operations, also other formats like YUV instead of RGB can be processed.

Optionally the red channel can be used as index value to address a color palette (max 8 bits). Alpha can be part of the value from the palette or explicitly coded in the pixel word together with the index value.

Optionally the alpha channel can be used as a bit mask to enable certain features in downstream processing for each pixel individually.

5.4.5.1.4 Clip and Skip Window

NOTE The following cannot be used for run length - encoded buffers.

The output frame dimension does not need to match the source buffer dimension. Any other size between 1 and 16’384 pixels in horizontal and vertical direction is allowed. For that a clip window size and position relative to the source buffer can be configured:

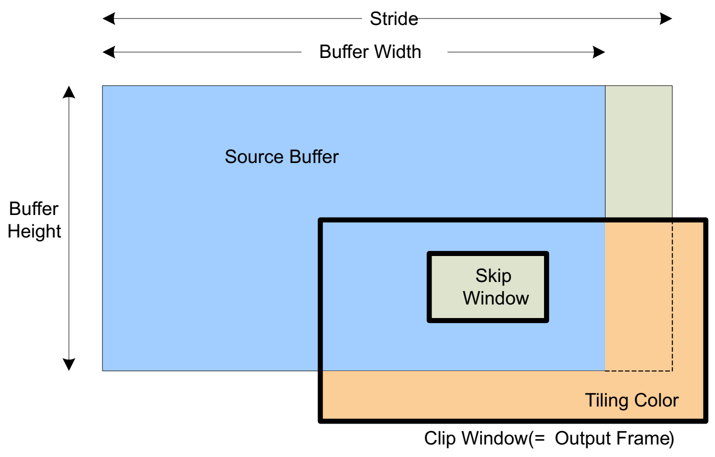


Figure 5-12: Fetch Clip and Skip Window

Those regions of the clip window, that lie outside the source buffer, are filled with the tiling color. This can be black, the programmable constant color (which is also used for null size components) or the color of the nearest border pixel of the source buffer.

Additionally, a skip window can be setup inside the clip window. It is filled with either black or the constant color. In that skip region no source buffer pixels are read. This can save memory bandwidth, for example, in case that this area is overlaid anyway by some other layer in subsequent processing.

Optionally the effect of the skip window can be inverted, making it effectively a clip window that is independent from the output dimension:

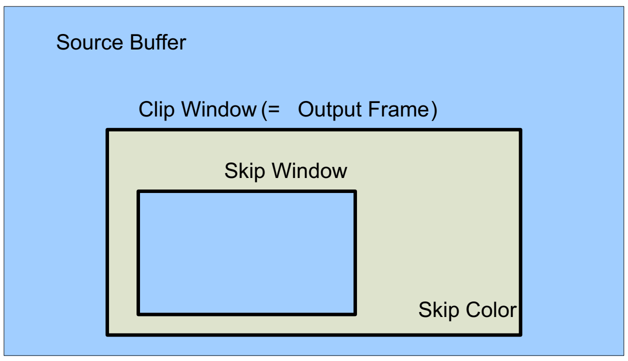


Figure 5-13: Skip Window Inversion

5.4.5.1.5 Global Alpha

The alpha value that is read for each pixel from memory can be multiplied with the programmable constant alpha value. This allows for fade-in/out effects of transparent graphics with a single operation.

5.4.5.1.6 Transparent Color

A transparent color can be defined. The alpha value for pixels is then set to 0 (fully transparent), if the pixel’s RGB value matches this color, or 255 (fully opaque) otherwise.

5.4.5.1.7 Multiply Modes

The R, G and B values of each pixel can be multiplied with its alpha value (pre-multiplied alpha). This is the recommended output format for any sort of filter operation in downstream processing such as scaling, otherwise less visible pixels would have the same contribution to filtered output pixels than others.

Alternatively R, G and B can be multiplied with different constant factors between 0 and 1, allowing to individually scale certain color channels.

5.4.5.2 Frame Generator

A frame with constant color can be generated without access to any buffer in memory.

Simple Scaling

NOTE The following cannot be used for run length - encoded buffers.

The source image can be resized with factors 4, 2, 0.5 or 0.25 by repetition respectively dropping of source buffer pixels. This can be setup individually for horizontal and vertical direction. Start offset for the repetition or drop pattern is configurable.

5.4.5.2.1 Scan Directions

NOTE The following cannot be used for run length - encoded buffers.

The scan direction on the source buffer is freely programmable. This makes it possible to horizontally or vertically flip the image data and/or to rotate it by 90, 180 or 270 degree. This allows all combinations of the following examples:

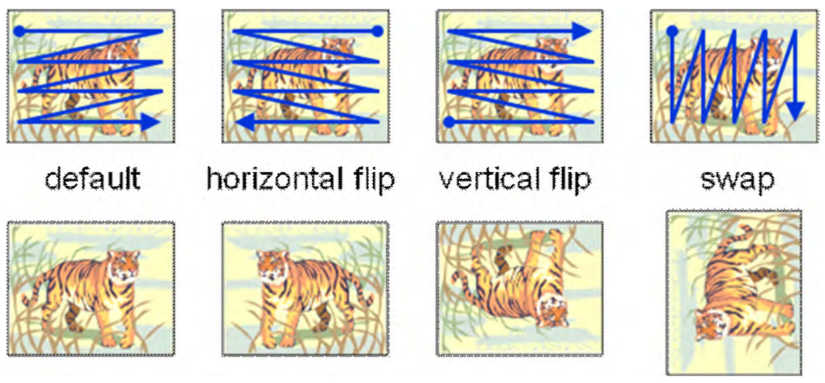


Figure 5-14: Scan Directions

5.4.5.2.2 Run-Length Decoder

NOTE The following is available for Fetch RLD only.

The source buffer can be run-length encoded in order to save memory size and bandwidth. Supported formats:

- According to the TGA File Specification
- Customization to the format above regarding header word format and endianness in the encoded data stream.

All pixel sizes and color formats as for uncompressed buffers are supported.

5.4.5.2.3 Sprites

NOTE The following is available for Fetch Sprite only.

The Fetch Sprite unit can generate a constant color frame (which is typically set to transparent) with up to 32 different image overlays on it (sprites). The resulting image is called a sprite layer and is typically alpha blended onto some other layer.

The base address in memory, any dimension between 1 x 1 and 2048 x 2048 pixels and the position can be set individually for each sprite.

All pixel sizes and color formats as for source buffers are supported. These must be same for all sprites.

Overlapping sprites are painted according to a fixed z-order. Sprites can only be alpha blended onto a background layer, not on other sprites.

For indexed color formats a subsequent color palette (CLuT unit) can be split into several smaller palettes for different groups of sprites ranging from 1 palette with 256 entries shared by all sprites to 32 palettes with 8 entries for each sprite.

5.4.5.3 Frame Capture Unit

The unit interfaces an external input stream of frame data (capture input) to the Iris internal format. It operates on pixels completely independent from a specific color format.

It can handle any sort of corrupt input data (e.g. missing syncs or pixels) while keeping Iris processing in a defined, robust and evaluable state.

Limit for frame dimension is 16’384 x 16’384 pixels including blank intervals.

5.4.5.4 External Source Interface

The unit interfaces between a capture and a Pixel Engine. It converts pixel input data to the internal pixel processing format, which is 30-bit RGB plus 8-bit Alpha.

5.4.5.4.1 Pixel Formats

Same as Fetch Unit - Pixel Formats

5.4.5.4.2 Clip Window

NOTE This feature is not supported by this Iris derivative.

5.4.5.4.3 Transparent Color

Same as Fetch Unit - Transparent Color.

5.4.5.5 Layer Blend Unit

Combines two input frames to a single output frame.

5.4.5.5.1 Overlay

The output dimension corresponds to the primary input (in block diagrams the top side input), the secondary input can have smaller size. It is positioned at any point inside the primary frame:

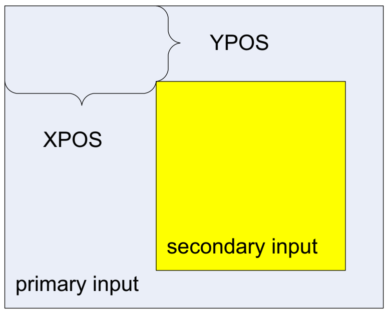


Figure 5-15: LayerBlend Overlay

5.4.5.5.2 Blending

The pixels of the secondary overlay area can be computed by alpha blending. The following blend functions can be set up individually for primary and secondary input and individually for RGB and Alpha:

Table 5-7: LayerBlend Blend Functions

ZERO	$C/\alpha_{blend} = 0 * C/\alpha_{in}$;
ONE	$C/\alpha_{blend} = 1 * C/\alpha_{in}$;
PRIM_ALPHA	$C/\alpha_{blend} = \alpha_{prim} * C/\alpha_{in}$;
ONE_MINUS_PRIM_ALPHA	$C/\alpha_{blend} = (1-\alpha_{prim}) * C/\alpha_{in}$;
SEC_ALPHA	$C/\alpha_{blend} = \alpha_{sec} * C/\alpha_{in}$;
ONE_MINUS_SEC_ALPHA	$C/\alpha_{blend} = (1-\alpha_{sec}) * C/\alpha_{in}$;
CONSTANT_ALPHA	$C/\alpha_{blend} = \alpha_{const} * C/\alpha_{in}$;
ONE_MINUS_CONSTANT_ALPHA	$C/\alpha_{blend} = (1-\alpha_{const}) * C/\alpha_{in}$;

The output color is the sum of the primary and secondary blend functions.

Instead of a per-pixel alpha, a global value can be used (α_{const}).

5.4.5.5.3 Packing

The unit can be used to merge multiple color planes to a packed format (e.g. RGB and separate alpha or separate YUV planes).

5.4.5.5.4 Alpha Mask Generation

- A special mode to generate the output alpha value can be enabled. It uses the information if a pixel is inside the secondary overlay area and optionally the alpha value of the primary input (assuming that this is an alpha mask generated by a previous LayerBlend unit). From that an output alpha of 0 or 255 is computed by one of the following patterns:

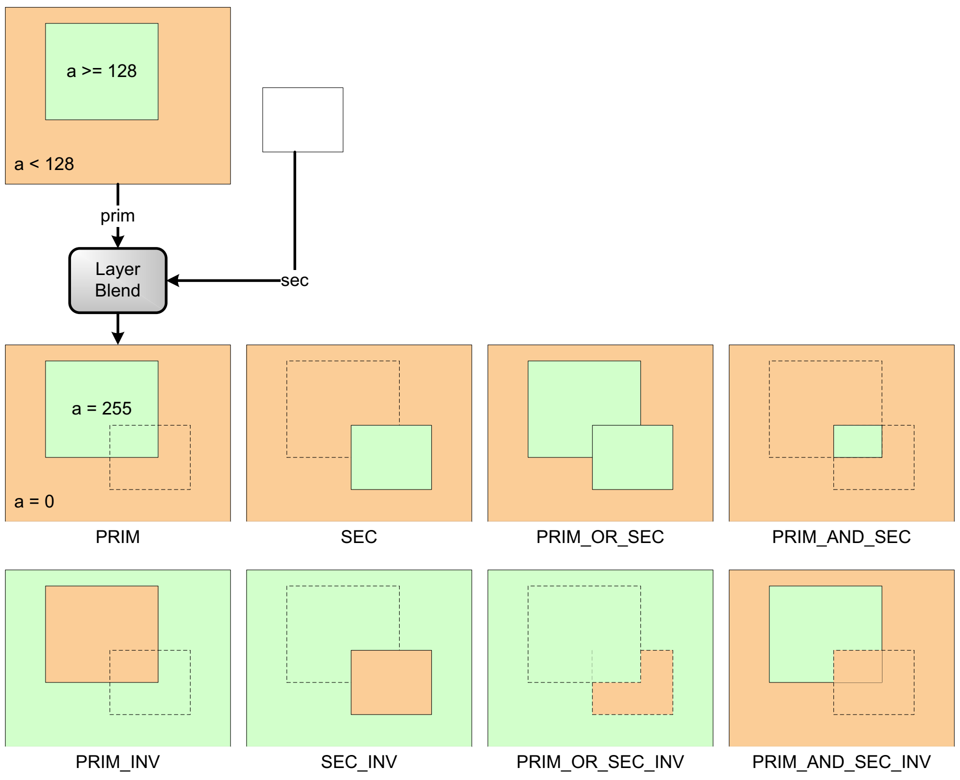


Figure 5-16: LayerBlend Alpha Mask Generation

This is useful when subsequent units for color transformations (e.g. a color matrix or look-up table) or signature computation operate in alpha mask mode. These operations can then automatically be limited to the area of certain layers or layer combinations.

5.4.5.6 External Destination Interface

The unit interfaces between a Pixel and a Display Engine. It behaves neutral on frame data.

5.4.5.6.1 Performance Counter

A performance counter is provided by which an application can determine the actual pixel rate that a system can provide for the display controller. This make it possible to evaluate the robustness of a setup for tearing-free display operation. The display itself must be turned off during this kind of measurement.

5.4.5.7 Frame Generator

5.4.5.7.1 Timing Generator

Generates a free-running display timing with active area, blanking intervals and synchronization pulses up to 16,384 x 16,384 pixels (including blank).

Any constant color can be setup to be used for active pixels.

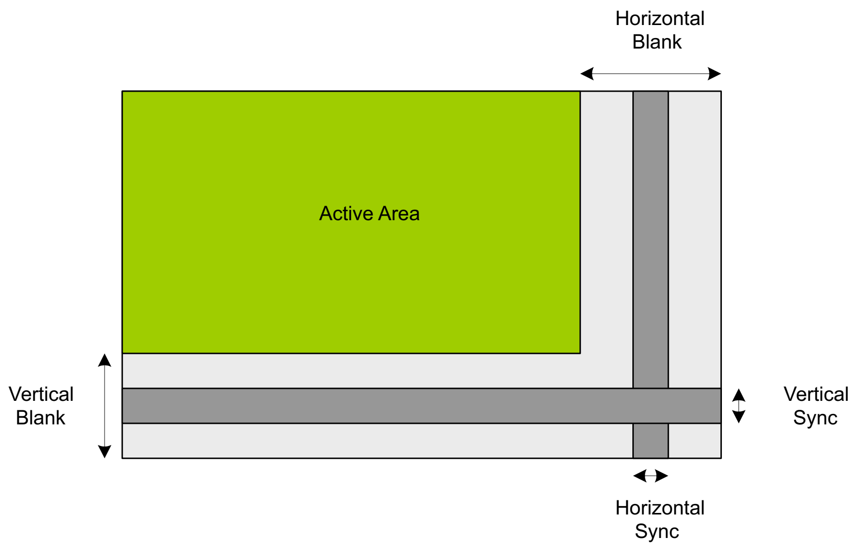


Figure 5-17: Frame Generator Output Timing

5.4.5.7.2 Stream Overlay

- Up to two input streams can be overlaid at any position inside the generated output frame (including the blanking areas).

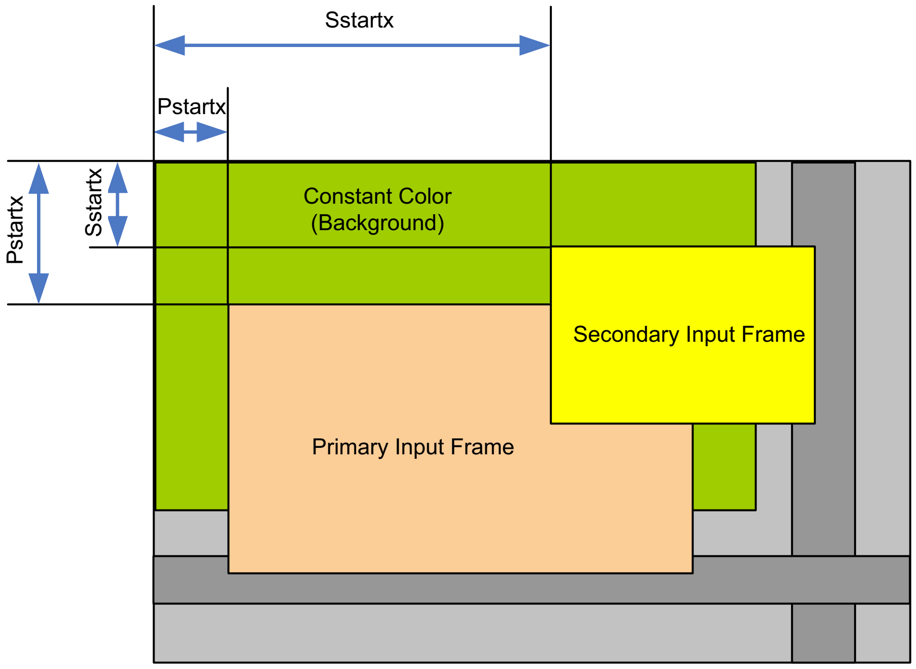


Figure 5-18: Frame Generator Stream Overlay

Overlay of the streams can be individually switched on and off. Also the one to display on top can be selected. Seamless switching between these settings can be done during display operation.

Regarding the allowed sizes for input and output streams and their relation, refer to Display Controller - Limitations.

In case of tearing on any of the input streams (e.g. because an external capture link is broken or available memory bandwidth in a system is not sufficient), the Frame Generator stays in a defined, robust and evaluable state. The constant color background is then drawn instead of the input stream until data is available again. In any case input pixels are displayed at correct positions only on the display in order to reduce visibility of tearing artifacts.

5.4.5.7.3 Timing Synchronization

The vertical refresh rate of the output timing can be automatically synchronized to the secondary input stream. This makes it possible to directly link a video source to the display.

The initial (and fast) phase alignment is done by inserting lines into the vertical front porch during blanking interval. The fine adjustment during normal operation is done by inserting or dropping pixels in the horizontal front porch during the blanking interval.

The initial alignment can be done in background while the display is operating and showing the primary stream. When synchronized a seamless switch between the two is possible.

Regarding the allowed differences in refresh rate between the output and secondary input timing, refer to Display Controller - Limitations.

5.4.5.7.4 Programmable Interrupts

Four different interrupt trigger events can be setup at any position relative to the output timing. These can be issued once per frame (VSync interrupt) or once per line (HSync interrupt).

5.4.5.7.5 Panic Mode

The display mode that controls which of the input streams to show, can automatically switch to another setting in response to a hardware event.

One possible event is a signature violation (see Signature Unit). Other events can be defined by the Panic Switch module (please refer to chapter “2.7.1 Panic Switch”).

In general, this mode is used to prevent display of corrupt output data in case of a malfunction in the system. So, for example, in case a capture link brakes the Frame Generator can automatically switch display to a static image from local memory without the need of software interaction.

5.4.5.8 Color Matrix

5.4.5.8.1 Linear Color Transformation

Can perform a linear color transformation on the color vector of each input pixel according to the following operation (alpha value is not modified):

$$\begin{bmatrix} \text{red\_out} \\ \text{green\_out} \\ \text{blue\_out} \end{bmatrix} = \begin{bmatrix} a11 & a12 & a13 \\ a21 & a22 & a23 \\ a31 & a32 & a33 \end{bmatrix} \cdot \begin{bmatrix} \text{red\_in} \\ \text{green\_in} \\ \text{blue\_in} \end{bmatrix} + \begin{bmatrix} c1 \\ c2 \\ c3 \end{bmatrix}$$

5.4.5.8.2 Alpha Pre-Multiply

Alternatively the red, green and blue can be multiplied with the incoming alpha value (where 0 maps to factor 0.0 and 255 to 1.0). This results in output format “pre-multiplied alpha”, which is recommended for any sort re-sampling operation on RGBA images (e.g. scaling or warping). The alpha value itself is not modified.

5.4.5.8.3 Alpha Masking

Linear color transformation can optionally be enabled for individual pixels only. The mask can be activated for pixels with alpha value <128 or ≥128. Color of masked pixels is not changed.

5.4.5.8.4 Display Specifics

Matrix operation is generally disabled for blanking pixels when the unit is inside a Display Engine.

5.4.5.9 Color Lookup Table

The unit implements 3 look-up tables with 256 x 10 bit entries each. These can be used for different kind of applications.

5.4.5.9.1 Color Lookup

For non-linear color transformations (e.g. gamma correction) the color components R, G and B are processed independently. The input code is used as table index, the table entry as 10-bit output color code. Input alpha is by-passed unchanged.

5.4.5.9.2 Index Lookup

For color palettes the red input channel is interpreted as index value and used to address all tables. The index size can range from 0 to 8 bits. The table entries build the 30-bit RGB output color. Input alpha is by-passed unchanged.

Alternatively the alpha value can be stored in the palette. In that case the 30-bit vector is interpreted as RGBA8886 and the component values up-scaled to 10-bit RGB and 8-bit alpha.

5.4.5.9.3 Dithering

The 10-bit RGB output values can be spatially dithered to 8-bit resolution.

5.4.5.9.4 Alpha Masking

The look-up operation can be individually enabled or disabled for each pixel using the most significant bit of the input alpha code. If disabled for a pixel, the RGBA input is passed unchanged.

5.4.5.9.5 Display Specifics

Look-up is generally disabled for blanking pixels when the CLuT is inside a Display Engine.

5.4.5.10 Dither Unit

The unit can increase the physical color resolution of a display from 5, 6, 7 or 8 bits per RGB channel to a virtual resolution of 10 bits. The physical resolution can be set individually for each channel.

5.4.5.10.1 Dithering

The resolution is increased by mixing the two physical colors that are nearest to the virtual color code in a variable ratio either by time (temporal dithering) or by position (spatial dithering).

An optimized algorithm for temporal dithering minimizes noise artifacts on the output image.

5.4.5.10.2 Alpha Masking

The dither operation can be individually enabled or disabled for each pixel using the alpha input bit.

5.4.5.11 Timing Controller

The TCon can generate a wide range of customized synchronization signals for direct interfacing to the column and row drivers of most panel types.

Mode control of the correlated IO cells is done with the related Global Control register.

5.4.5.11.1 Control Signals

12 control signals can be generated with freely programmable waveforms:

- 12 pulse generators.
- 1 signal sequencer with up to 64 signal transitions.
- 12 signal mixers with programmable function table.

5.4.5.11.2 Data Modes

The RGB output signals can drive IO cells in the following modes:

- Single-ended TTL
  (24-bit data output at pixel clock frequency)
- Low Voltage Differential Swing according to the RSDS standard
  (12-bit data output at twice the pixel clock frequency)
- LVDS interface according to the LVDS standard

RSDS and LVDS modes can operate in

- Single channel (one pixel output per pixel clock cycle)
- Dual channel (two pixels in parallel at half the pixel clock frequency)

Dual channel mode supports the following schemes:

- Interleaved (pixel pairs on both channels are horizontal neighboring pixels)
- Split (first channel for left and second one for right half of the display). Split position is configurable, allowing to have an asymmetric split (different width of left and right half).

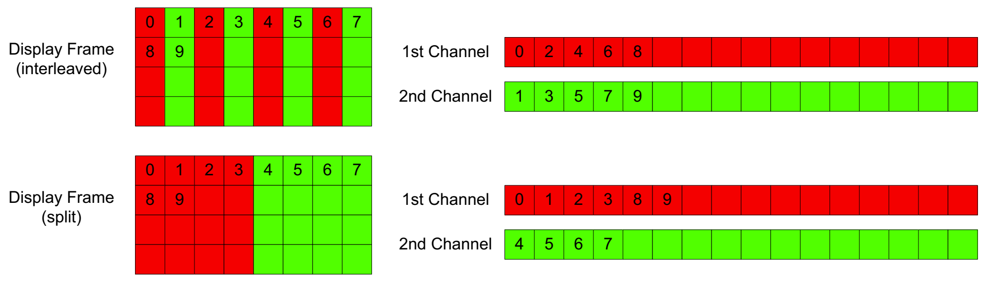


Figure 5-19: Dual Channel Modes

5.4.5.11.3 Data Multiplexing

- Each of the 24 RGB output bits can be mapped to any of the 24 RGB input bits or to constant 0 or 1.
- For LVDS modes, the multiplexing capability also includes timing control signals in addition to RGB data.

- For RSDS operation, each of the 12 signal pairs can be individually swapped, which corresponds to polarity inversion of the corresponding pair of output signals.
- In dual channel modes all can be setup individually for each channel.
- These features can help to optimize board layouts.

5.4.5.11.4 Inversion Control

In order to reduce EMI effects for TTL applications, the TCon can automatically invert the RGB data vector for certain pixels in order to keep the overall toggle rate on these signals as low as possible.

An external port indicates if the data for a pixel is inverted or not for each channel.

5.4.5.12 Signature Unit

In order to control the correctness of display output, a signature value is computed for each frame and can be compared against a reference value. In case of a mismatch (signature violation) a hardware event can be triggered, for example a software interrupt.

5.4.5.12.1 Signature Computation

Two measurement modes are supported:

- Automatic measurement for each frame.
- Single measurement. Must be triggered explicitly by the software.

Valid result is signalled by interrupt for both modes. For each measurement two different kind of signature values are computed:

- Sum of all color values.
- CRC-32 checksum according to IEEE 802.3 with fixed polynomial and start value.

While the second method is more reliable for detection of corrupt frames, the first one makes it possible to specify a tolerance range for the check against the reference value, e.g. to ignore noise artifacts.

A minimum number for consecutive frames with signature violation can be configured, before the violation event is actually triggered.

5.4.5.12.2 Evaluation Window

Two windows can be setup to restrict signature computation to a certain area of the input frame:

- Evaluation window (only pixels inside are considered).
- Skip window (only pixels outside are considered).

So, only green area would be considered in the following example:

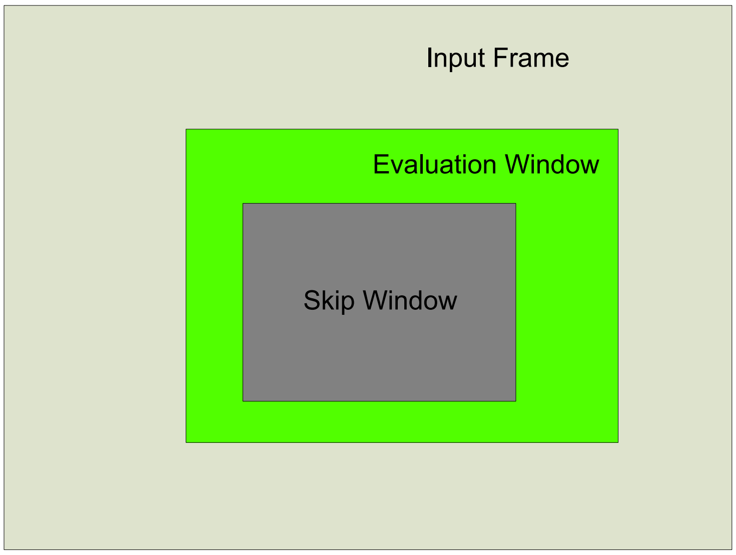


Figure 5-20: Signature Evaluation Window

5.4.5.12.3 Alpha Masking

Alternatively to the evaluation window the 1-bit alpha value of each input pixel can control if it is included into the signature computation. This makes it possible to monitor any kind of shape on the display frame.

5.4.5.12.4 Panic Mode

In response to a signature violation the Signature Unit can automatically and instantly switch the area of the evaluation window to a certain constant.

5.5 IRIS-MVL Map Tables

5.5.1 Interrupt Map

Related topics: “5.4.3.4 Interrupt Controller”, “5.6.2 Interrupts” Setup.

Iris provides two types of sources for interrupts and other synchronization methods: Events and states.

See chapter “2.8 Interrupt Controller” for more information.

Table 5-8: Event Map

ID	Unit	Name	Description
0	PixEng Top	Sequence complete (ExtDst#0 synchronizer)	Processing path for display operation in the Pixel Engine (end point EstDst#0) is idle and no further operation is pending (occurs only if explicitly triggered by SW). Not required for documented use cases.
1	PixEng Top	Sequence complete (ExtDst#1 synchronizer)	Same as above for ExtDst#1.
2	ExtDst#0	Frame complete	The ExtDst#0 unit has completed transmission of a display frame to the Frame Generator. See Display Controller - “5.7.2.2 Dynamic”
3	ExtDst#1	Frame complete	Same as above for ExtDst#1.
4	LayerBlend#0	Shadow load (secondary input)	A shadow load token has reached the LayerBlend#0 unit on its secondary input, all upstream shadow registers have been loaded. Not required for documented use cases.
5	LayerBlend#1	Shadow load (secondary input)	Same as above for LayerBlend#1.
6	DisEng Top	Shadow load (display output)	A shadow load token has reached the display output port, all upstream shadow registers have been loaded. See Display Controller - “5.7.2.2 Dynamic”
7	DisEng Top	Sequence complete (display output)	Display operation has stopped and display processing pipeline is completely flushed. All related processing units are idle. See Display Controller “5.7.2.1 Static” and “5.7.2.2 Dynamic”
8 9 10 11	FrameGen#0	Programmable positions (id 0 to 3)	Occurs each output frame at a programmable position relative to the display timing. See Display Controller - “5.7.2.2 Dynamic” and “5.7.3.7 Programmable Interrupts”
12	FrameGen#0	Shadow load (primary input)	A shadow load token has reached the FrameGen unit on its primary input (ExtDst#0), all upstream shadow registers have been loaded. See Display Controller “5.7.2.1 Static” and “5.7.2.2 Dynamic”
13	FrameGen#0	Shadow load (secondary input)	Same as above for secondary input (ExtDst#1).
14	Sig#0	Shadow load	Shadow registers of the Signature unit have been loaded due to a shadow load token or a dedicated SW trigger. See Signature Setup
15	Sig#0	Measurement complete	Measurement of the signature value for one display frame has been completed and can be read out by SW. See Signature Setup
16	Sig#0	Signature error	The measured signature value of a display frame does not match the reference value provided by SW. See Signature Setup
17	Sig#1	Shadow load	Same as ID 14 for Sig#1.
18	Sig#1	Measurement complete	Same as ID 15 for Sig#1.
19	Sig#1	Signature error	Same as ID 16 for Sig#1.
20	Sig#2	Shadow load	Same as ID 14 for Sig#2.
21	Sig#2	Measurement complete	Same as ID 15 for Sig#2.
22	Sig#2	Signature error	Same as ID 16 for Sig#2.
23	Sig#3	Shadow load	Same as ID 14 for Sig#3.
24	Sig#3	Measurement complete	Same as ID 15 for Sig#3.

Table 5-8: Event Map (Continued)

ID	Unit	Name	Description
25	Sig#3	Signature error	Same as ID 16 for Sig#3.

Table 5-9: Status Map

ID	Unit	Name	Description
0	FrameGen#0	Synchronization status (primary input)	Active when primary input (ExtDst#0) and output timing of the Frame Generator are synchronized and operating stable. See Memory Stream - Timing Setup
1	FrameGen#0	Synchronization status (secondary input)	Same as above for secondary input (ExtDst#1). See Capture Stream - Timing Setup
2	FrameCap#0	Synchronization status	Active when input (video capture) and output timing of the Frame Capture unit are synchronized and operating stable. See Capture Stream Timing Setup

5.5.2 Address Map

Total size of Iris address space is 32K bytes (range = 0x8000).

Access to reserved areas within this space or to areas without registers within the range of a sub unit will generate an error response on the AHB bus.

Table 5-10: Address Map

Base Address	Range	Type	Unit	Register Map
0x0000	0x0800	Configuration	Common Control	ComCtrl
0x0800	0x0400	Configuration	Pixel Engine – Top-Level	Pixelbus
0x0c00	0x0400	Configuration	Pixel Engine – Fetch#0 (RLD)	FetchRLD
0x1000	0x0400	Configuration	Pixel Engine – Fetch#1 (Sprite)	FetchSprite
0x1400	0x0400	Configuration	Pixel Engine – ExtSrc#0	ExtSrc
0x1800	0x0400	Configuration	Pixel Engine – ExtDst#0	ExtDst
0x1c00	0x0400	Configuration	Pixel Engine – ExtDst#1	ExtDst
0x2000	0x0800	Configuration	Pixel Engine – CLuT#0	CLuT
0x2800	0x0400	Configuration	Pixel Engine – LayerBlend#0	LayerBlend
0x2c00	0x0400	Configuration	Pixel Engine – LayerBlend#1	LayerBlend
0x3000	0x0400	Configuration	Display Engine – Top-Level	DispCfg
0x3400	0x0400	Configuration	Display Engine – FrameGen#0	FrameGen_PS
0x3800	0x0400	Configuration	Display Engine – Matrix#0	Matrix
0x3c00	0x0800	Configuration	Display Engine – CLuT#1	CLuT
0x4400	0x0400	Configuration	Display Engine – Dither#0	Dither
0x4800	0x0800	Configuration	Display Engine – TCon#0	TCon
0x5000	0x0400	Configuration	Display Engine – Sig#0	Sig
0x5400	0x0400	Configuration	Display Engine – Sig#1	Sig
0x5800	0x0400	Configuration	Display Engine – Sig#2	Sig
0x5c00	0x0400	Configuration	Display Engine – Sig#3	Sig
0x6000	0x0800	reserved		
0x6800	0x0400	Configuration	Capture Engine – FrameCap#0	FrameCap
0x6c00	0x1400	reserved		

5.5.3 Key Map

Related topic: “5.4.3.3 Register Locking” Function

Table 5-11: Key Map

Units	Key Type	Key Value	Register Map
Common Control	Lock Unlock	0x78631639 0x803fd2da	ComCtrl
Display Engine – Sig#0/1/2/3	Lock Unlock	0xac0b1f35 0x1f24a7a4	Sig
Display Engine – FrameGen#0	Lock Unlock	0xd1a375fb 0xb4ac332b	FrameGen_PS

5.6 Basic Setup

5.6.1 IP Identifier

Information about the Iris IP derivative and design revision can be read from the Common Control register IPIdentifier.

5.6.2 Interrupts

Related topic: "5.4.3.4 Interrupt Controller" Function, "5.5.1 Interrupt Map" Table

All registers to setup interrupt control are located in Common Control: Interrupt<*>.

For more information, please refer to chapter "2.8.6 Interrupt Table".

5.6.3 Clock Settings

All clock properties are setup in the global control unit (please refer to chapter "2.4 Clock Structure" and "5.3 System Setup MB88F33x 'Indigo2(-x)'"). For any use case at least the following clocks must be activated:

- AXI bus clock (*axi_clk*).
  Lower or higher frequency to save power or increase performance.
  Optionally spread spectrum can be enabled.
- Configuration clock for AHB bus (*HCLK*)
  Lower or higher frequency to save power or increase performance.
  Optionally spread spectrum can be enabled.
- Display Clock (*dsp_clk*). Depending on DspClkDivide setting:
  DIV1: Frequency = pixel clock frequency (not allowed for single channel operation!)
  DIV2: Frequency = 2 x pixel clock frequency
  Optionally spread spectrum can be enabled.
  Note: The pixel clock frequency above must always match the video mode, not the frequency of the *dsp_clock_ctrl* output signal, which is reduced by factor 2 in dual channel mode.

When the video layer is enabled, additionally:

- Capture clock (*apx_clk*).

For frequency limitations to all clocks refer to chapter Limitations.

5.6.4 Reset Settings

For any use case at least the following resets must be released:

- AXI reset (*axi_reset_n*).
- Configuration reset (*HRESETn*).
- Display reset (*dsp_reset_n*).

When the video layer is enabled, additionally:

- Capture reset (*apx_reset_n*).

Reset must not be released before the corresponding clocks are operating stable.

5.6.5 Power Optimization

Related topic: "5.4.3.5 Power Optimization" Function.

In order to bring the Iris module into an idle state with minimal power consumption:

- Set all <unit> <port> sel fields of the Pixelbus configuration to disable.
- Set SwReset and ClockDisable fields in all Fetch and Store unit configurations to "1".

In order to reduce the effective clock speed for a specific part only of the Pixel Engine, clock throttling can be set up:

- Setup clock divider extdst0_div for the memory stream.
- Setup clock divider extdst1_div for the capture stream.

5.7 Display Controller

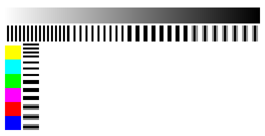
5.7.1 Getting Started

5.7.1.1 Minimal Setup

Assuming all configuration fields being in reset state, the minimal setup to enable display activity is:

- Setup all required system clocks with pixel clock = 6 MHz (QVGA at 60Hz). See Clock Settings.
- Release reset. See Reset Settings.
- Set FgEn of FrameGen#0 to '1'.

This will generate a QVGA timing (320 x 240 active pixels) with high active signals in TCon by-pass mode with white background and the following test image at top-left corner:



Note that when driving a panel directly with this setup, most typically the sync polarities must be changed to low active:

- Set PolHs and PolVs to LOW.

The border pixels of the active area are painted. The border pixels of the active area are painted in constant color in this display mode, which is white per default. To change it:

- Write color into FgCCR register

5.7.1.2 Display Path

In general the processing path must be setup before enabling display operation. That is the number of layers and streams and the kind and sequence of processing units that are active:

- Set *<pu>_<port>_sel* fields of Pixel Engine for all input ports of selected processing units. Start with extdst0_src_sel and extdst1_src_sel, then go upstream up to Fetch or ExtSrc units. Disabled ports are set to '0'.
- Set src_select field of Display Engine to define the tap for signature computation when needed.
- Set all processing units to neutral operation mode that are not needed but cannot be by-passed with available pipeline configurations (this is the reset setting in all cases).
- For all Fetch units in use set fields *SWReset* and *ClockDisable* to *OPERATION*.

For possible paths refer to the Iris Block Diagram. Note, that primary inputs of processing units are always at top side connected in that diagram.

For active processing units it is mandatory to connect the primary input. Secondary and tertiary inputs are optionally to connect.

5.7.2 Control Flow

5.7.2.1 Static

When configuration and memory content can be static during display operation, a simple control flow with disabled shadow registers can be used:

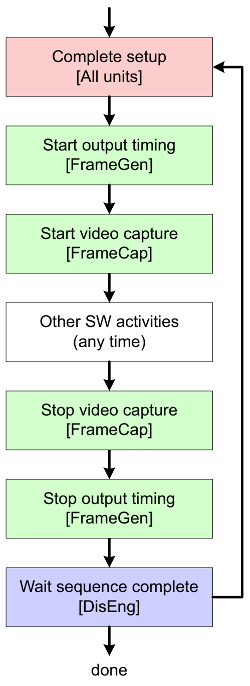

Complete setup [All units]
Start output timing [FrameGen]
Start video capture [FrameCap]
Other SW activities (any time)
Stop video capture [FrameCap]
Stop output timing [FrameGen]
Wait sequence complete [DisEng]
done

Figure 5-21: Static Control Flow

Complete setup:

- Disable all shadows registers for all selected processing units in the display path:
  - *<pu>_shdw* to *WRITETHROUGH* (PixEng top-level)
  - *ShdEn* to *false* (configuration of processing unit)
- Set extdst0_Sync_Mode and extdst1_Sync_Mode of Pixel Engine to *SINGLE*.
- Setup the Display Path.
- Setup the Display Stream, Memory Stream (optionally) and Capture Stream (optionally).

Start display operation:

- Set FgEn of FrameGen#0 to true (starts the display and memory stream).
- Set Cen of FrameCap#0 to true (starts the capture stream). This must not be done before the display stream has been started.

Stop display operation:

- Set *Cen* of FrameCap#0 to *false* (stops the capture stream). Note: This must be done before the display stream is stopped.

- Set *FgEn* of FrameGen#0 to *false*. This will not immediately stop the display stream, but complete all pending frames in the memory and capture streams.
- In order to detect that display operation has completed (output timing stopped and all units are idle), SW can either poll FrameGen#0 status EnSts or wait for the sequence complete interrupt of the Display Engine.

5.7.2.2 Dynamic

When there is a need to modify parts of the configuration or content of display buffers in memory during display operation, the following control flow using shadow registers must be used:

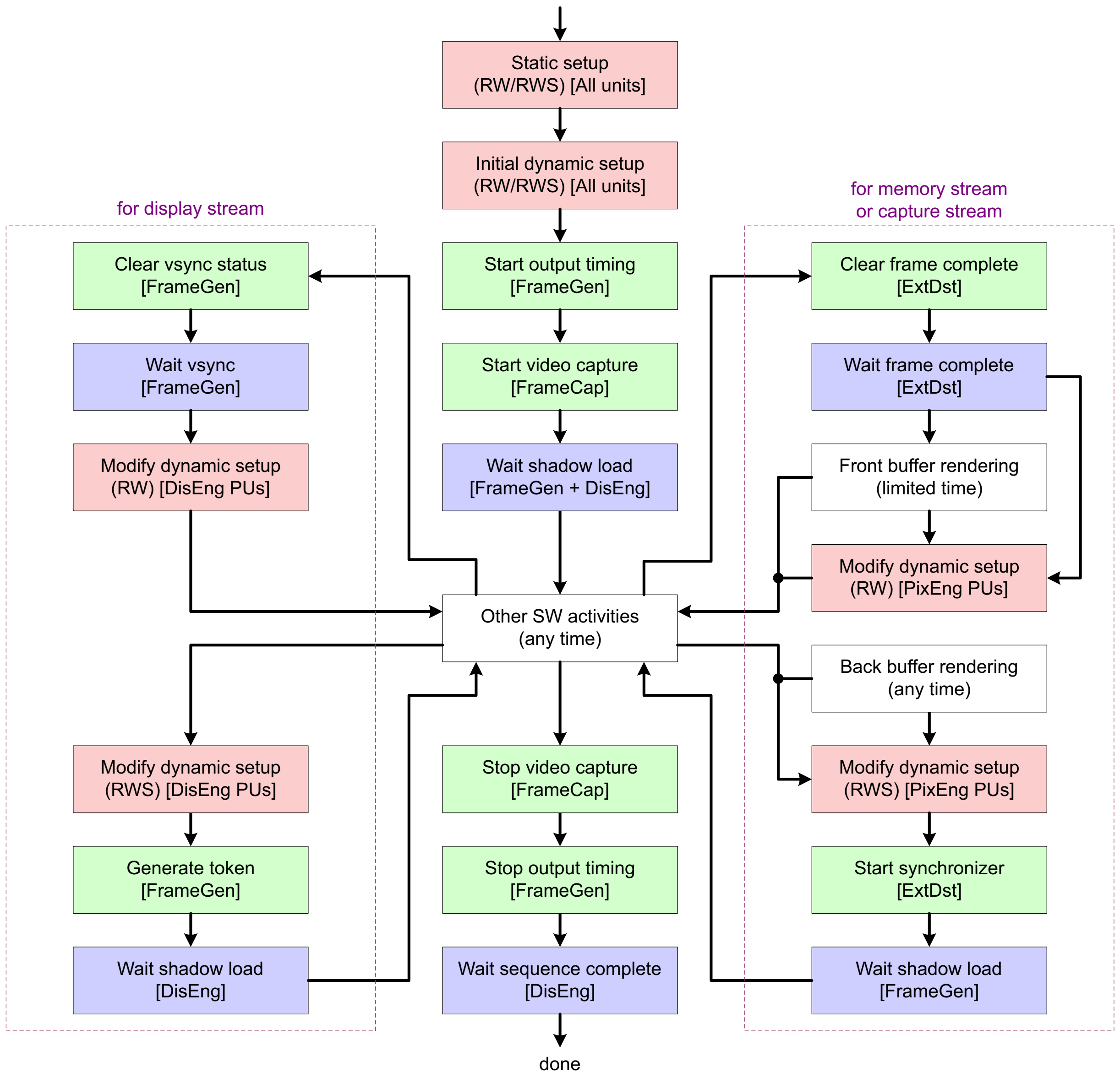

Static setup (RW/RWS) [All units]
Initial dynamic setup (RW/RWS) [All units]
for display stream
for memory stream or capture stream
Clear vsync status [FrameGen]
Start output timing [FrameGen]
Clear frame complete [ExtDst]
Wait vsync [FrameGen]
Start video capture [FrameCap]
Wait frame complete [ExtDst]
Modify dynamic setup (RW) [DisEng PUs]
Wait shadow load [FrameGen + DisEng]
Front buffer rendering (limited time)
Modify dynamic setup (RW) [PixEng PUs]
Other SW activities (any time)
Back buffer rendering (any time)
Modify dynamic setup (RWS) [DisEng PUs]
Stop video capture [FrameCap]
Modify dynamic setup (RWS) [PixEng PUs]
Generate token [FrameGen]
Stop output timing [FrameGen]
Start synchronizer [ExtDst]
Wait shadow load [DisEng]
Wait sequence complete [DisEng]
Wait shadow load [FrameGen]
done

Figure 5-22: Dynamic Control Flow

Configuration fields are grouped into three categories for this flow:

- Static setup (RW or RWS).
  These are all settings that are not changed during operation. This applies to all settings described below (control flow setup) and in general to most other un-shadowed fields (RW), particularly those related to capture and display timing. In addition an application can decide to mark settings static that could be dynamic, but do not required a change.
- Dynamic setup (RWS).
  All settings that are not static and that are shadowed. Modifications are written into the shadows, which are loaded at any time consistently for the same frame in all units.
- Dynamic setup (RW).
  All settings that are not static, but do not have a shadow. In general content of all look-up tables or color palettes (CLuT) belong to this category. Additionally some settings related to sprites (FetchSprite) could be added here. Modifications must be done during vertical blanking period to avoid tearing artifacts on display output.

For display buffers in memory two different setup types are covered:

- Double buffer setup (front and back buffer).
  Display is configured to the front buffer, while the back buffer can be modified without impact on current display. Buffers can be switched at any time when back buffer rendering has completed.
- Single buffer setup (front buffer only).
  Saves memory, however, modifying the buffer content must be done during vertical blanking phase to avoid undefined display output.

Static setup:

- Enable all shadows registers for all selected processing units in the display path:
  - *<pu>_shdw* to *SHADOWED* (PixEng top-level)
  - *ShdEn* to *true* (configuration of all processing units that have shadow registers)
- Set extdst0_Sync_Mode and extdst1_Sync_Mode of Pixel Engine to SINGLE.
- Disable shadow token end-points at LayerBlend units: Set ShdTokEpSec to false.
- Enable shadow token end-point at both FrameGen inputs: Set ShdTokEpPrim, ShdTokEpSec to true and ShdLdSel to TRIGGER.
- Setup all static and initial dynamic settings of the Display Stream, Memory Stream (optionally) and Capture Stream (optionally).
- Generate a shadow load token for all streams: Write ‘1’ to ShdTokGen field of FrameGen (display stream) and to extdst0 Sync trigger (memory stream) or extdst1 Sync trigger (capture stream).
- Wait until all shadows have been loaded (shadow load interrupts of Frame Generator and Display Engine).

Start display operation:

- Set FgEn of FrameGen#0 to true (starts the display and memory stream)
- Set Cen of FrameCap#0 to true (starts the capture stream). This must not be done before the display stream has been started.

Wait until initial setup has been loaded:

- Shadow load interrupts of Frame Generator and Display Engine).

Display is now operating.

Change of dynamic shadowed setup (RWS) in the display stream (Display Engine):

- Write new values to RWS fields.
- Generate a shadow load token for the display stream: Write ‘1’ to ShdTokGen of FrameGen#0.
- Wait until shadows have been loaded (shadow load interrupt of Display Engine).

Change of dynamic un-shadowed setup (RW) in the display stream:

- Clear status of the programmable FrameGen interrupt 2.
- Wait for that interrupt (indicates start of vertical blanking).
- Write RW fields. This must not take longer than the vertical blanking period of the display timing. Its end can be detected with programmable FrameGen interrupt 3. Otherwise tearing artifacts may appear.

Change of dynamic shadowed setup (RWS) in the memory or capture stream:

- Write new values to RWS fields.
- Start synchronizer: Write ‘1’ to extdst0_Sync_trigger (memory stream) or extdst1_Sync_trigger (capture stream). When started the synchronizer will automatically retain pending display frames until the Pixel Engine pipeline is completely flushed and then load all shadow registers from Fetch/ExtSrc units down to ExtDst into active configuration before continuing. So all shadow settings take effect synchronously for the same frame.
- Wait for shadow load interrupt of primary (memory stream) or secondary (capture stream) input of FrameGen.

This procedure is also used to switch front and back buffer by changing *BaseAddress* field of the corresponding Fetch unit.

Change of dynamic un-shadowed setup (RW) in the memory or capture stream (Pixel Engine):

- Clear status of corresponding ExtDst frame complete interrupt.
- Wait for that interrupt.
- Write RW fields. This must be completed until the next frame is kicked, which can be detected with the programmable FrameGen interrupts 0 (memory stream) and 1 (capture stream). Otherwise tearing artifacts may appear.

This procedure is also used to change content of a front buffer.

Stop procedure for display operation is same as described for the Static Control Flow.

5.7.3 Display Stream

Related topic: “5.4.4.1 Display Stream” Function

5.7.3.1 Timing Setup

The Frame Generator is a free-running video timing generator. It can operate without any input streams being active by generating a constant color background.

The following video mode parameters must be setup in FrameGen#0:

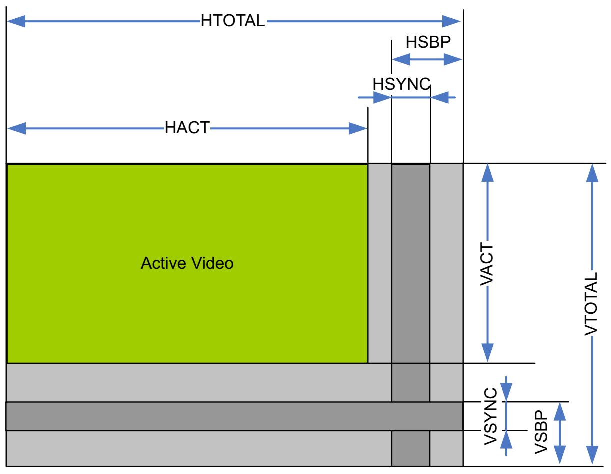


Figure 5-23: Display Mode Parameters

- Frame dimension (mandatory): Hact , Htotal, Vact, Vtotal, Hsbp, Vsbp.
- Sync pulse generation: HsEn, Hsync, VsEn, Vsync (FrameGen)
  Sync pulse polarity: PolHs, PolVs, PolEn, PixInv (DisEng Top-Level).
  Both is only required, if the Timing Controller (TCon#0) operates in by-pass mode.

The setup must comply the Display Controller Limitations for active area, sync and blanking intervals.

The vertical refresh rate implicitly results from the total frame dimension and the pixel clock frequency, which is always half of or equal to the display clock frequency (*dsp_clk*) provided by the MB88F33x ‘Indigo2(-x)’ clocking system. For more information, please refer to chapter “5.3 System Setup MB88F33x ‘Indigo2(-x)’”.

When change of dynamic RW registers in the display stream is required, programmable VSync interrupts can be used to signal begin and end of vertical blanking period:

- Set Int2En to true, Int2HsEn to false, Int2Row to VACT+1 and Int2Col to 1 (mandatory).
- Set Int3En to true, Int3HsEn to false, Int3Row to VTOTAL and Int2Col to HTOTAL (optionally).

To start timing generation:

- Set FgEn to ‘1’.

To stop timing generation:

- Set FgEn to ‘0’. This won’t stop immediately, but continues until all pending frames in the pipeline have been completed. The current status can be detected by either polling EnSts or by waiting for sequence complete interrupt of the Display Engine.

The following table gives some typical examples for video timings according to VESA Coordinated Video Timings (CVT) (note: some register must be configured to table value less 1, see field descriptions; VR = Vertical Refresh rate; AR = Aspect Ratio; RB = Reduced Blanking):

Table 5-12: Typical Video Modes

	VR [Hz]	AR	RB	pix_clk [MHz]	Hact	Vact	Htot	Vtot	Hsync	Vsync	Hsbp	Vsbp
QVGA	60	4:3	no	6.000	320	240	400	253	32	4	72	10
VGA	60	4:3	no	23.750	640	480	800	500	64	4	144	17
NTSC	60	3:2	no	26.750	720	480	896	500	64	10	152	17
PAL	50	4:3	no	28.500	768	576	960	596	72	4	168	17
WVGA	60	15:9	no	29.500	800	480	992	500	72	7	168	17
SVGA	60	4:3	no	38.250	800	600	1024	624	80	4	192	21
XGA	60	4:3	no	63.500	1024	768	1328	798	104	4	256	27
HD720	60	16:9	yes	64.000	1280	720	1440	741	32	5	112	18
WXGA	60	16:10	no	83.500	1280	800	1680	831	128	6	328	28
	60	5:2	yes	98.500	1920	768	2080	790	32	10	112	19
SXGA	60	4:3	no	109.000	1280	1024	1712	1063	136	7	352	36
	100	8:3	yes	110.500	1600	600	1760	629	32	10	112	26

5.7.3.2 Display Modes

The Frame Generator can handle two display streams on its primary (ExtDst#0) and secondary input (ExtDst#1), which are overlaid at any position inside the generated background frame.

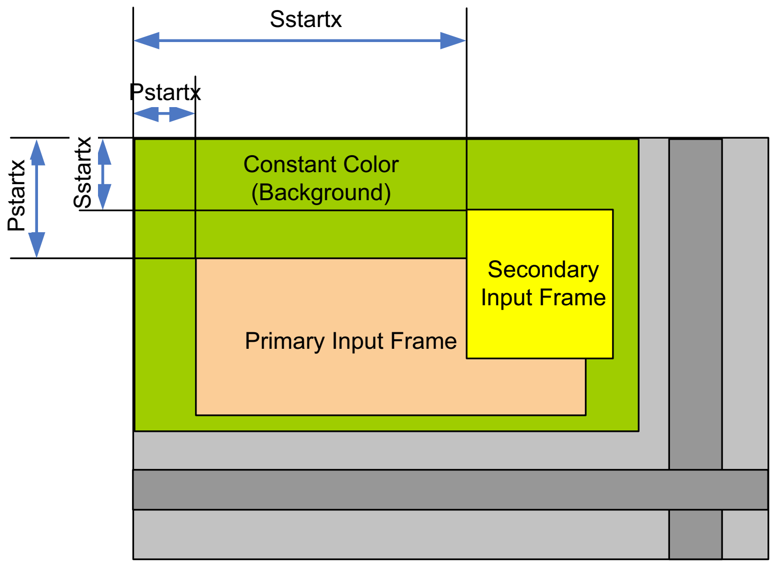


Figure 5-24: Display Stream Overlay

The appearance is controlled with following settings of FrameGen#0:

- Display mode: FgDm.
- Background color for active pixels (visible in areas where no input stream is overlaid): FgCCR. Blanking pixels without input frame overlay are always black.
- Position of the primary stream overlay: Pstartx/Pstarty. When using shadow registers, this position can be modified during operation (tearing-free).
- Horizontal position of the secondary stream overlay: Sstartx (Sstarty must be 0 due to display controller limitations). Should not be changed during operation to avoid tearing artifacts.

Dimension of overlays implicitly results from the size of the corresponding input stream. Overlay area must not exceed the active frame area!

Note, that the display mode can be changed during operation. Also if an input stream is disabled for display, it does not mean that the stream is deactivated. This allows seamless switching between display mdoes.

Particularly the memory stream can be displayed while the secondary input is not yet synchronized to a capture timing. So a typical system boot up sequence may look like this:

- Setup both memory and capture stream with *FgDm* = *CONSTCOL* (constant color is displayed).
- Wait for stable synchronization of primary input stream (see Timing Setup of Memory Stream).
- Change *FgDm* to *PRIM* (buffer from local memory is displayed).
- Wait for stable synchronization of secondary input stream (see Timing Setup of Capture Stream).
- Change *FgDm* to *SEC* (capture input is displayed).

This prevents frame tearing artifacts on the display at any time.

5.7.3.3 Color Transformations

Related topic: “5.7.7.1 Color Palette” Function.

Use Matrix#0 to setup a Linear Transformation.

Use CLuT#1 to setup a Non-linear Transformation.

Both can be controlled by Alpha Masking.

5.7.3.4 Dithering

Related topic: “5.4.5.9.3 Dithering” Function.

When a panel has less resolution than 8 bits per color, the most significant bits of the computed color output are connected.

To achieve a virtual resolution of 10 bits, the physical color code can be automatically varied by dithering.

Dither#0 setup:

- Set mode to ACTIVE.
- Set offset_select (spatial or temporal dithering) and desired method for mapping of dithered color codes (algo_select).
- Set physical color resolution for each channel: red/green/blue_range_select.

The following shows an example how the internal bit width is handled in case of dithering for one color channel, assuming a 5-bit source driving a 6-bit destination:

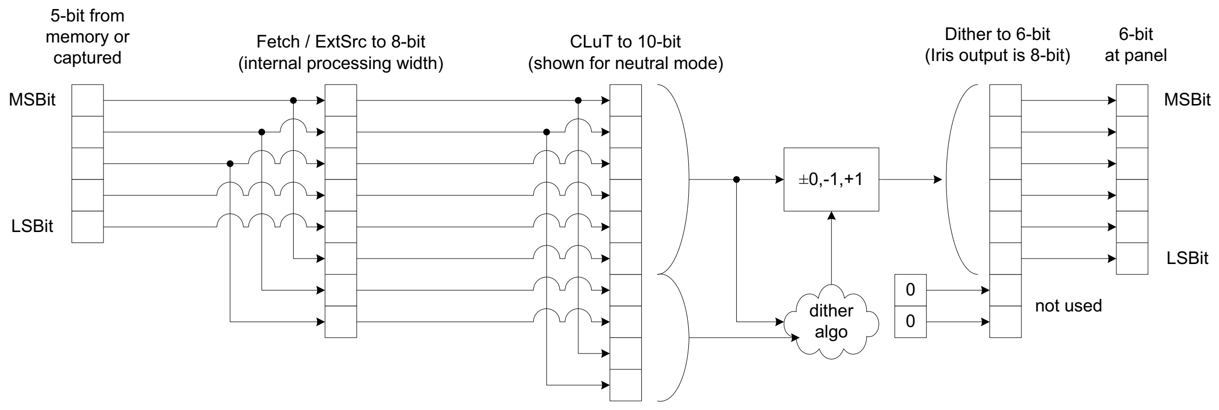

5-bit from memory or captured
Fetch / ExtSrc to 8-bit (internal processing width)
CLuT to 10-bit (shown for neutral mode)
Dither to 6-bit (Iris output is 8-bit)
6-bit at panel
MSBit
LSBit
±0,-1,+1
dither algo
0
0
not used
MSBit
LSBit

Figure 5-25: Bit Mapping with Dithering

Typically temporal dithering achieves better quality, however, may introduce minor noise artifacts. In contrast the output of spatial dithering is static, but dither patterns may become visible on areas of constant color.

When signature shall be checked after the dithering stage, spatial dithering must be selected.

5.7.3.5 By-pass Mode

In this mode, the programmable timing generators are disabled and the synchronization signals and pixel data as generated by the Frame Generator are directly routed to outputs with the following mapping:

Table 5-13: By-pass Mode Control Signal Mapping

Display interface output signal	Internal signals from the Frame Generator
TSIG[0,3,6]	HSync (horizontal sync pulse)
TSIG[1,4,7]	VSync (vertical sync pulse)
TSIG[2,5,8]	Enable (active pixels)
TSIG[9]	HLast (last pixel of total line)
TSIG[10]	VLast (all pixels of last line of total frame)
TSIG[11]	reserved

Setup:

- Bypass of TCon to BYPASS_MODE.
- Polarity of output signals: See PolarityCtrl Register of DisEng.
- Inversion control (display interface output signal dsp_invers_ctrl): InvCtrEn.
- Bit mapping of 24 bit RGB vector (R[0..7], G[8..15], B[16..23]) to display interface output signals (dsp_data_a0/a1/d or dsp_lvds): MapBit<0..27> of TCon.
- Optionally for RSDS use cases: RSDS_Inv (can individually swap each pair of signals driving an RSDS I/O cell; effectively this changes the polarity of the corresponding differential output signal).
- ChannelMode must be SINGLE.

Note, that there is no difference in TCon setup between a TTL, RSDS or LVDS use case. Signals for all modes are provided in parallel, the mode must be selected in the configuration of the MB88F33x ‘Indigo2(-x)’ system (please refer to “5.3.2 Configuring Display Output Pins (Multiplexing)”).

5.7.3.6 TCon Mode

Related topic: “5.4.5.11 Timing Controller” Function

In this mode a much more customized output timing can be generated. Sync signals from the Frame Generator are ignored.

Initial TCon#0 setup is same as for By-pass Mode, except:

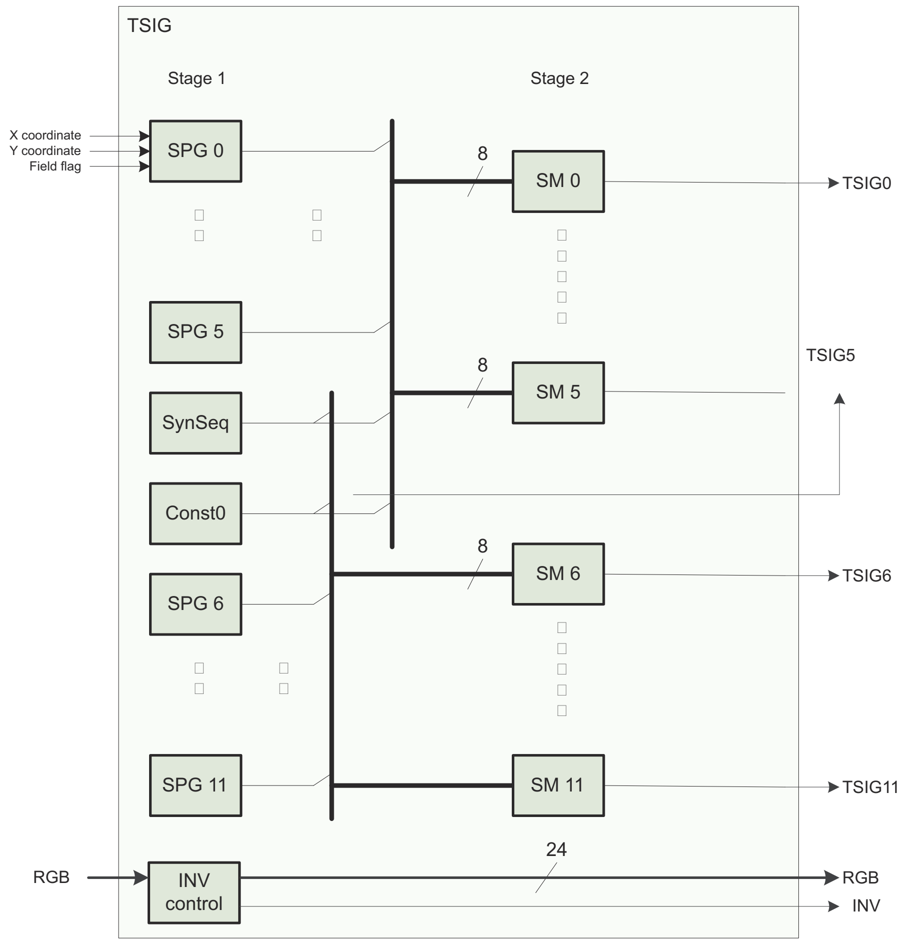
- Bypass to TCON_MODE.
- tcon_sync field: Can be used to take the sync signals as generated by the FrameGen as reference point for the timing generators instead of the last pixel/line of a line/frame. Effect: When the display timing is synchronized to capture input in *H_VLast* mode (default), the distance between generated control signals to the next first pixel of a line/frame (back porch) can jitter due to dynamic modifications in the hor/ver blanking length. In *H_VSync* mode it is the distance from the control signals to the previous first pixel of a line/frame that jitters (front porch). This may help to solve problems with panels being sensitive to that kind of sync jitter. NOTE: In *H_VSync* mode the sync polarity must be set to low active (PolHs and PolVs to LOW).

For dual channel operation additionally:

- ChannelMode to DUAL_INTERLEAVED or DUAL_SPLIT.
- PolEn to HIGH
- SplitPosition when channel mode is DUAL_SPLIT.
- MapBit_<0..27>_Dual and RSDS_Inv_Dual for second channel.

Reset settings of the timing generators are configured for a standard QVGA timing according to by-pass mode mapping for TSIG[0,1,2].

Sync signals are generated using a two stage approach in order to achieve maximum flexibility. In the first stage, signals are generated which carry positional timing information. Two methods are used to create these signals. The second stage combines them to form more complex waveforms.



SPG x = Sync Pulse Generator x
SM x = Sync Multiplexer x

Figure 5-26: TCon Timing Generators

One way to form the first stage signals is to use simple position matching to trigger an RS flip-flop or a toggle flip-flop. This is done using an array of twelve identical Sync Pulse Generators (SPG’s).

- Setup position for switch on (SPGPSON_X<0..11>/SPGPSON_Y<0..11>) and switch off positions (SPGPSOFF_X<0..11>/SPGPSOFF_Y<0..11>) for all pulse generators needed. The coordinate space must be satisfied the following condition:
  - $0 \leq X <$ HTOTAL (horizontal total pixels)
  - $0 \leq Y <$ VTOTAL (vertical total raster)
  - For dual channel modes (RSDS, and LVDS), X must be dividable by two and the corresponding bit in the SPGMKON<0..11> and SPGMKOFF<0..11> masks (bit 16) needs to be zero

- Set toggle modes SPGPSON_TOGGLE<0..11> and SPGPSOFF_TOGGLE<0..11> to
  - disable: Output of a sync pulse generator is set (switch on) or reset (switch off) if the current position equals the programmable position in all bits for which its don’t-care-vector (SPGMKON<0..11>/SPGMKOFF<0..11>) contains zeros. The Off matching is dominant, i.e. when both On and Off positions are matched at the same time, the output of the sync pulse generator is reset.
  - enable: The output toggles if the current position matches the programmable position in all bits for which its don’t-care-vector (SPGMKON<0..11>/SPGMKOFF<0..11>) contains zeros.. Toggle mode allows e.g. frame wise toggling signals. Set/Reset overrides toggle, and if both positions match and toggle, they cancel each other out.
- Since input is always progressive, SPGPON_FIELD<0..11>/SPGPOFF_FIELD<0..11> must be ‘0’.

A more sophisticated and powerful approach to creating first-stage signals is the use of a sequencer RAM to match a whole sequence of positions. A sync sequencer (SyncSeq) follows an arbitrary sequence of timing positions and generates an appropriate output signal. The length of the sequence as well as the content of the RAM, consisting of the position and the assigned output value are programmable. Operation is as follows: To start, the address counter is reset to zero and the RAM outputs the first position that matches and the output value for this position. If the comparator signals match, the RAM address is incremented, the preset output value (bit 31) is propagated and the RAM then outputs the next position to match. This match/address increment cycle continues until the programmed sequence length is reached. If the last position is matched, the address counter is reset to zero again and the cycle starts again. It is thus possible to generate arbitrarily complex waveforms with up to 64 edges (which is the maximum sequence length).

- Program sequence length to SSQCYCLE.
- For each item of the sequence setup all fields of register SSqCnts[0..63] (starting with index 0) analogously to the pulse generator setup.

Combing first stage sync signals: As shown above, there are twelve sync pulse generator outputs and one sync sequencer output. To obtain more complex waveforms, these signals can be combined in a second stage. Here, an array of twelve sync mixers (SMx) is used to calculate Boolean functions of first-stage signals. Each sync mixer can form any Boolean function on up to five inputs. The basic structure of one such mixer is depicted in the following diagram:

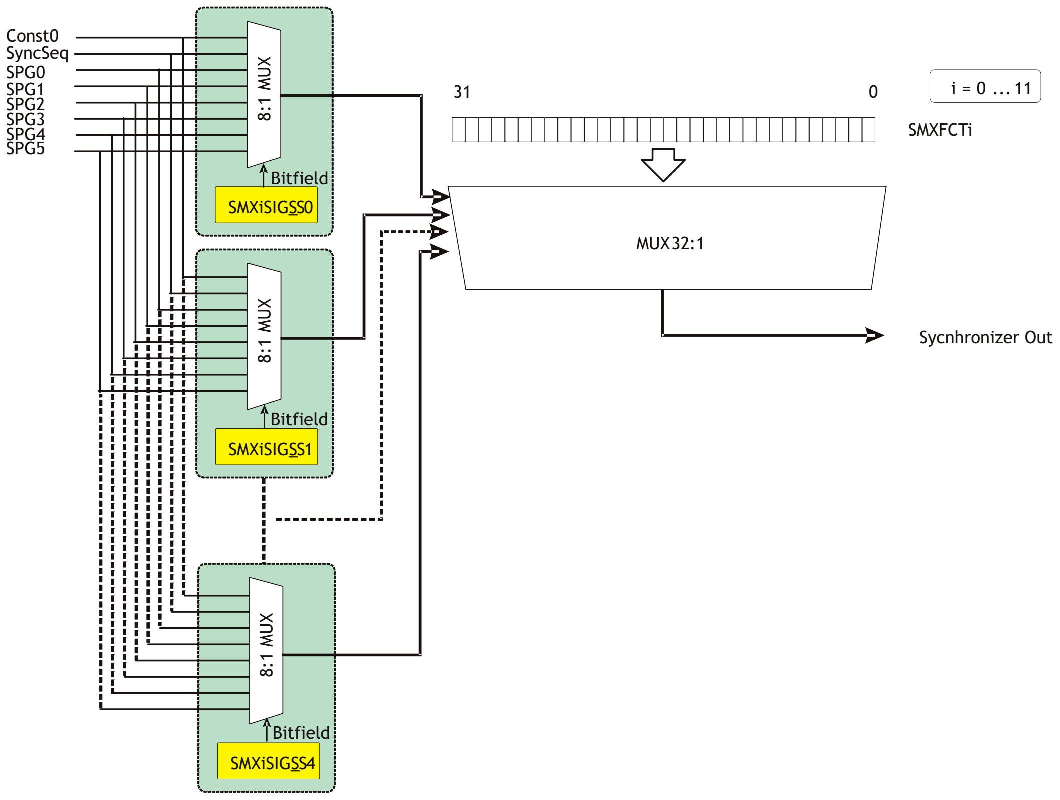


Figure 5-27: TCon Sync Mixer

Basic structure of a Sync Mixer: Each of the five address lines of the 32 to 1 multiplexer can be individually selected from any of the first-stage signals or constant ‘0’. The output is the result of a table look-up. The register *FctTable* contains the truth table of the Boolean function calculated:

- For each of the 12 control output signals (*TSIG[0..11]*) that are needed, setup the corresponding sync mixer (registers SMx<0..11>Sigs and SMx<0..11>FctTable).

The reset setting directly routes the 12 sync pulse generator outputs to the corresponding *TSIG* signal.

The concept of the sync mixers needs some explanation. In a first step the signals to be combined are selected. These are referred to as S0…S4 and form the address for the function table. This function table is used to look up the result of the Boolean operation the five selected signals shall be subject to.

An example may help understand the topic. Assuming the outputs of three Sync Pulse Generators shall form a combined signal with the function, one would proceed as follows: At first, the Sync Mixer signals S0…S4 are assigned the Sync Pulse Generator outputs or constant zero by programming the respective multiplexers. The next step is to build the function’s truth table. As the intended function has only three inputs, only eight entries need to be specified.

It is recommended that S4…S0 are listed in order of binary number representation. This makes it possible to use the function result row directly as register contents for the Sync Mixer function table, i.e. the last row is interpreted as binary 32 bit number with the LSB in the first row and the MSB in the last. For the example this would be [xxxx xxxx xxxx xxxx xxxx xxxx 0000 1000] binary, with x’s denoting arbitrarily set or reset bits, since these will never be read out of the function table.

Table 5-14: TCon Sync Mixer Example

Selected first-stage signals					Desired Output
S4 = 0	**S3 = 0**	**S2 = SPG1**	**S1 = SPG0**	**S0 = SyncSeq**	**SMx = f(S0..S4)**
0	0	0	0	0	0

Table 5-14: TCon Sync Mixer Example

Selected first-stage signals					Desired Output
S4 = 0	**S3 = 0**	**S2 = SPG1**	**S1 = SPG0**	**S0 = SyncSeq**	**SMx = f(S0..S4)**
0	0	0	0	1	0
0	0	0	1	0	0
0	0	0	1	1	1
0	0	1	0	0	0
0	0	1	0	1	0
0	0	1	1	0	0
0	0	1	1	1	0
Combinations [S4..S0] = 10000..11111 can never occur since S4 adn S3 are selected constantly zero.					does not need to be specified

NOTE For LVDS modes only outputs from Sync Mixer 0 to Sync Mixer 5 can be used.

NOTE For LVDS modes TSIG[2] is necessary to output a Enable signal

5.7.3.7 Programmable Interrupts

The Frame Generator provides a set of interrupts that are triggered once per vertical or horizontal refresh of the display output at any position in relation to the output frame timing:

- Registers Int0Config, Int1Config, Int2Config, Int3Config of FrameGen#0.

These are intended for general purpose.

5.7.4 Memory Stream

Related topic: “5.4.4.2 Memory Stream” Function

5.7.4.1 Timing Setup

Synchronization of the memory stream to the display stream is handled by the Frame Generator.

The reference timing is the display output. In relation to that a kick signal is generated once per frame. This will trigger start of fetching data from memory for the next frame.

By default the kick position is set to the last active display pixel. When changes to un-shadowed registers or buffer content in memory during display operation is required in the memory stream, then the kick signal can be delayed to the end of the vertical blanking period. By that a defined period arises where the Pixel Engine modules are completely idle. However, a minimum distance to the first active pixel of the next frame must be kept (kick ahead period).

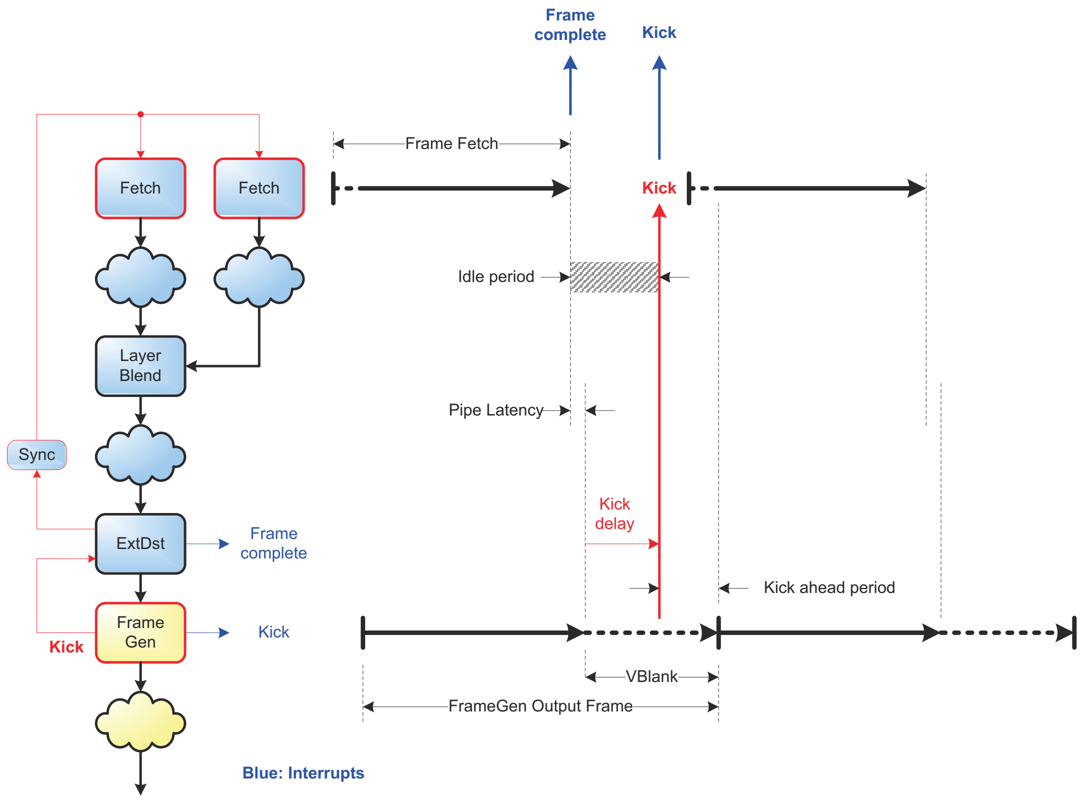


Figure 5-28: Kick Position Memory Stream

The minimum kick ahead period must consider the following effects to ensure that the fetched pixel data is available at the Frame Generator before the active display area starts:

- Pipeline latency of the kick signal and pixel data between FrameGen and Fetch units
- Memory read access latency and possible timeouts on the AXI bus
- Pipeline stalls due to run-in delays at start of line of layers and sprites

A kick position setup calculator is provided along with this document (Setup Tools for Iris-MVL).

FrameGen#0 setup using values from above calculator:

- PKickEn to '1' (enables kick generation).
- PKickRow to kick_row+1 and PKickCol to kick_col+1. When no idle period is required for re-configuration, it can simply be set to the position of the last active pixel instead.

NOTE When timing regulation is enabled for the Capture Stream (see Skew setup), then the programmed row and column must be clamped to HTotalMin/VTotalMin

- Int0En to PKickInt0En to true (optionally; allows SW to detect end of idle period).

ExtDst#0 setup:

- KICK_MODE to EXTERNAL.

SW can detect stable operation by monitoring the following status flags:

- PrimSyncStat of FrameGen#0 (or corresponding interrupt signal). Reasons for synchronization loss:
  - PFifoEmpty: Data stream from a Fetch unit (e.g. AXI bandwidth not sufficient) fell down.

When synchronization is lost, this won't affect the display output timing. Instead of primary input pixels the configured background color is displayed then until input is synchronized again.

The robustness of a system setup for tearing-free operation can be evaluated by measuring the maximum pixel rate that the memory stream could provide if it wasn’t limited to the display’s refresh rate:

- Set PerfCountMode to ENABLE and KICK_MODE to SOFTWARE in ExtDst.
- Keep the Frame Generator turned off (FgEn = 0).
- Kick generation of a single display frame (write to KICK field of ExtDst).
- Wait for frame complete interrupt of ExtDst.
- Read out PerfResult of ExtDst.

This measurement should be repeated for a longer period to eliminate bandwidth variations of the system. The effective pixel rate for a single frame calculates to

- Fill rate [MPix/s] = freq_axi_clk [MHz] x (frame_width x frame_height) / PerfResult

It should be clearly above the target pixel clock for the display for all frames.

5.7.4.2 Background Layer

Setup a Display Buffer for Fetch#0 (at least buffer/pixel format and clip window).

Fetch#0 and CLuT#0 can be used to setup a Color Palette.

5.7.4.3 Sprite Layer

Setup a Display Buffer for Fetch#1 with Sprites.

Fetch#1 and CLuT#0 can be used to setup a Color Palette.

The sprite (= secondary) layer is blended onto the background (= primary) layer by Blend Operation with LayerBlend#1.

5.7.5 Capture Stream

Related topic: “5.4.4.3 Capture Stream” Function

5.7.5.1 Timing Setup

Synchronization of the capture stream to the display stream is handled by the Frame Generator.

The reference timing is the capture input. In relation to that a kick signal is generated once per frame. This will trigger start of fetching data from memory for the next frame and allow passing of captured frames into the Pixel Engine (ExtSrc).

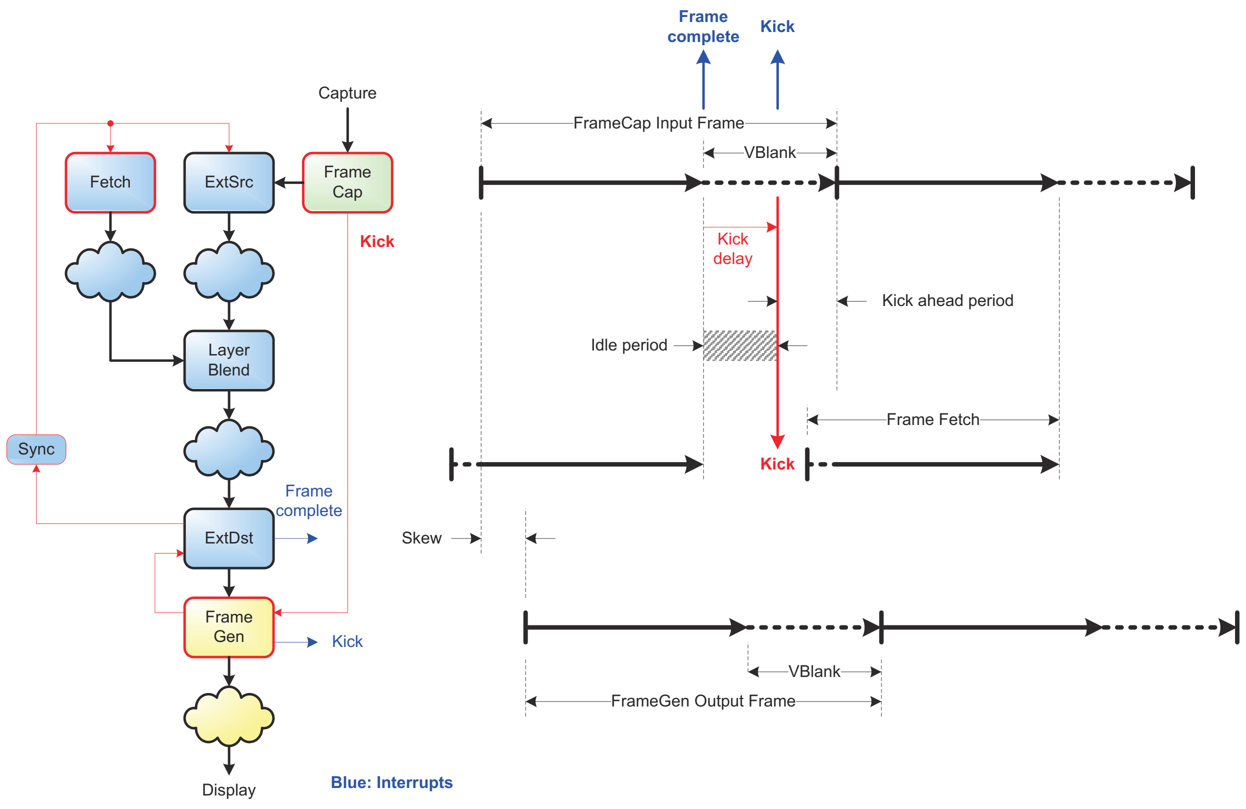


Figure 5-29: Kick Position Capture Stream

Kick positioning is computed analog to the Memory Stream Timing Setup, except that it is in relation to the capture timing and that the following effects must be additionally considered for the minimum kick ahead period:

■ Irregularities in the captured data stream due to burst-like transmission patterns

A kick position setup calculator is provided along with this document (see Setup Tools for Iris-MVL).

FrameCap#0 setup using values from above calculator:

■ KickDel to kick_delay. When no idle period is required for re-configuration, it can simply be set to 0.

FrameGen#0 setup:

■ SKickEn to '1' and SKickTrig to EXTERNAL.
■ Int1En to SKickInt1En to true (optionally; allows SW to detect end of idle period).

ExtDst#1 setup:

■ KICK_MODE to EXTERNAL.

ExtSrc#0 setup:

■ StartSel to LOCAL.

SW can detect stable operation by monitoring the following status flags:

■ SecSyncStat of FrameGen#0 (or corresponding interrupt signal). Reasons for synchronization loss:
  - SFifoEmpty: Data stream from either a Fetch unit (e.g. AXI bandwidth not sufficient) or from the Frame Capture unit (e.g. transmission error) fell down.
  - SkewRangeErr: Skew setup is wrong or video timings are violating the given limitations (see Skew Setup).

SyncStat of FrameCap#0 (or corresponding interrupt signal). Reasons for synchronization loss:

- VsEarly or VsLate: Transmission error on capture input or Width/Height setup does not match the transmitted video mode.
- FifoFull: Back stall from either a Fetch unit (e.g. AXI bandwidth not sufficient) or from the Frame Generator (e.g. when not enabled).

When synchronization is lost, this won't affect the display output timing. Instead of secondary input pixels, the configured background color is displayed until input is synchronized again.

5.7.5.2 Skew Setup

Independent from the kick mechanism the Frame Generator can regulate the display timing to a constant skew in relation to the capture timing by modifying the size of blanking intervals.

In general the input mode can differ from the output mode in both resolution and refresh rate as long as all constraints given in the Display Controller Limitations are considered. Otherwise it might not be possible to realize tearing free display of the capture stream.

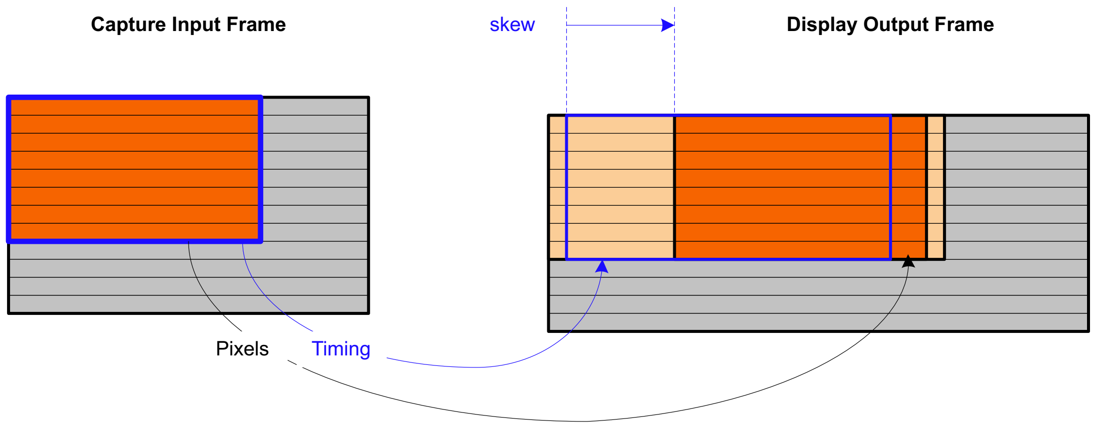


Figure 5-30: Capture and Display Frame

Differences in refresh rate are balanced by dropping or inserting blanking pixels/lines in the horizontal/vertical front porch of the blanking intervals of the display output timing. Regulation in the horizontal front porch allows more frequent and precise corrections, which are required to support all specified use cases. Regulation in the vertical front porch allows faster synchronization of the display timing to a new video source. If disabled this may take a noticeable time.

For the regulation process to work correctly, SW must configure a target skew value. This skew is the time in number of display pixel clock cycles from the moment where the first pixel of an active line from the captured frame is written into the FIFO of the Frame Generator until the moment where this pixel is read from the FIFO for display output. When the actual skew value gets negative, the FIFO runs empty. If it gets too large it may run full. Both results in frame tearing (loss of synchronization).

FrameGen#0 setup, assuming nearly identical video modes for capture and display:

- Set SREn to '1' (enables skew measurement).
- Set SRMode to BOTH (enables regulation of horizontal and vertical front porch).
- Set allowed range for total line width and count according to tolerance of the connected panel:
  HTotalMin ≤ Htotal ≤ HTotalMax and VTotalMin ≤ Vtotal ≤ VTotalMax
  Allowed range for horizontal regulation must be sufficient to compensate differences in line frequency. Both min values must consider a required minimum length of 1 for the front porch.

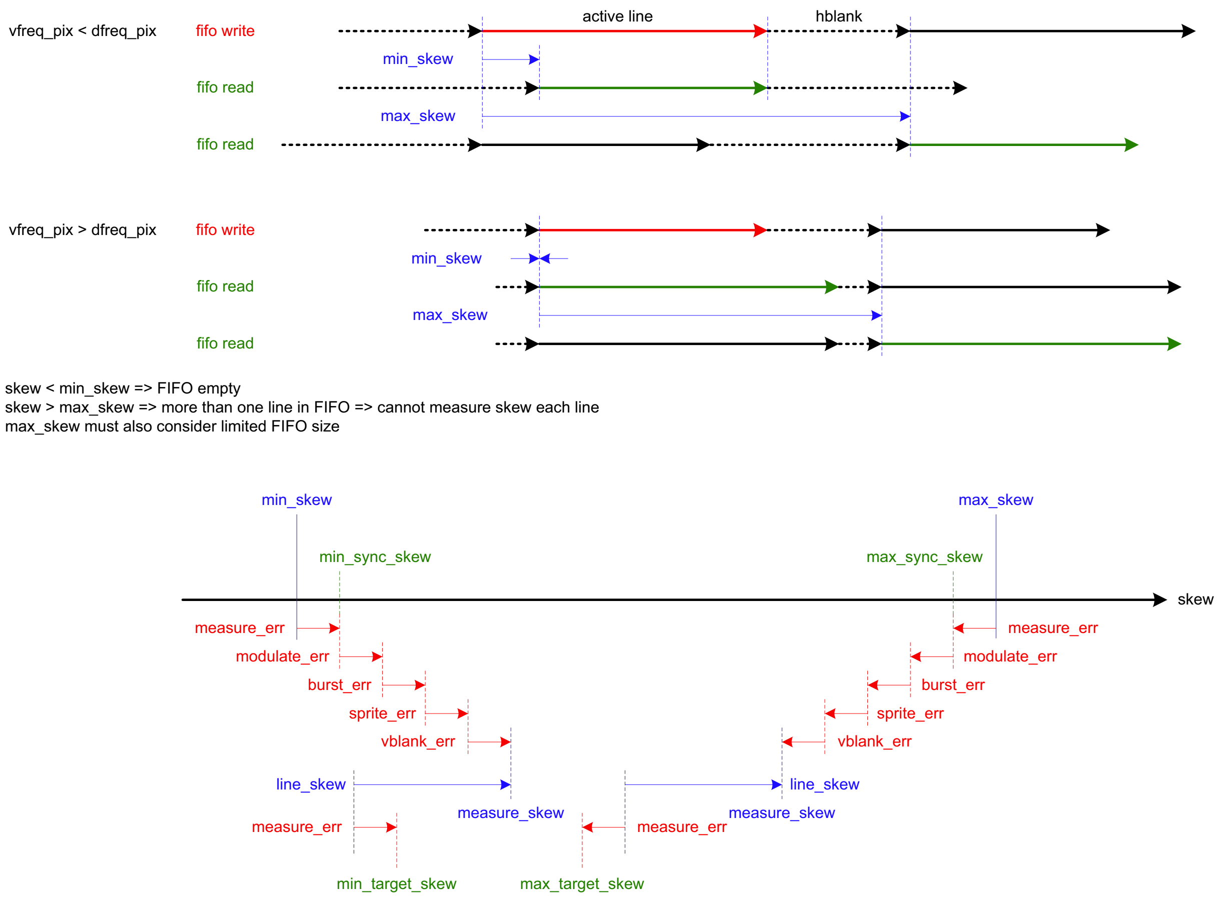
- SyncRangeLow to null and SyncRangeHigh to Htotal, but not larger than 1024.
  Only when skew measurements are inside this range for a certain period, which can be configured by LevSkewInRange (recommended value = 1), read-out and display of pixel data from the FIFO will synchronize to the correct display position. The appearance of display pixels, while display position is not yet synchronized, can be configured by SRDbgDisp.
- TargetSkew to (SyncRangeLow + SyncRangeHigh) / 2.
  This is the target skew value for both the horizontal and vertical regulation procedure.
- Optionally SRFastSync can be enabled in order to speed up the initial synchronization procedure to a new input signal. In that case the display timing will not start immediately when FgEn is enabled, but not before the first captured frame is received (the actual status can be determined with EnSts). So this can only be used when no display operation is required during synchronization procedure.
- When the Timing Controller is setup for dual channel operation, then the total length of a line must always be even. To ensure that SRAdj and SREven must be enabled. In addition, if the Timing Controller is setup to synchronize on hsync and vsync, the hsync offset HTOTAL – HSBP needs to be odd.

Note, that frame generation must be enabled (FgEn) before frame capturing (Cen) in any case.

For detection of stable operation see Timing Setup.

Most typically a setup cannot assume identical video modes, but must consider differences in refresh rate, pixel clock frequency and/or resolution. For this, and to ensure that a certain combination of modes is possible at all to setup, some more aspects must be considered:

- The actual skew must not get negative
- The actual skew must not get larger than what the FIFO capacity can handle
- The FIFO must never contain more than one complete line
- Skew measurement errors can occur from the following sources:
  - Clock domain crossings (*measure_err*)
  - Spread-spectrum modulated display clock (*modulate_err*)
  - Burst-like transmission of captured data (*burst_err*)
  - Line skew extrapolation during vertical blanking interval (*vblank_err*)
  - Pipeline stalls due to line fetch run-in/out delays (*sprite_err*)
  - Position of the secondary stream overlay is changed during operation (not considered in the following formulas).



skew < min_skew => FIFO empty
skew > max_skew => more than one line in FIFO => cannot measure skew each line
max_skew must also consider limited FIFO size

Figure 5-31: Target Skew Setup

A setup calculator is provided along with this document (see Setup Tools for Iris-MVL) to determine all related parameters from a given capture and display mode and from the operational environment. Resulting from that the setup must be adjusted as follows:

- SyncRangeLow to min_sync_skew and SyncRangeHigh to max_sync_skew.
- TargetSkew to any value between min_target_skew and max_target_skew.
- HTotalMin must not be larger than dpix_htot_min and HTotalMax not smaller than dpix_htot_max.

Also a setup must normally support a small range of possible frequencies in order to allow dynamic (or static) variations of both during operation. For that the computation above must be done for the following two corner cases:

1. vfreq_pix_{min} and dfreq_pix_{max}
2. vfreq_pix_{max} and dfreq_pix_{min}

This results in two sets of ranges for target skew and sync range. The intersection range of both must be considered for the setup of *TargetSkew* und *SyncRangeLow/High*. If there is no intersection, the range for pixel clocks violates the system limitations.

In general it is recommended to make sure that the display line frequency is always higher than the capture line frequency. This allows more flexibility because speeding up the display rate has a significant limit by the minimum front porch sizes.

For debugging purposes two timing parameters that are measured on the capture timing can be evaluated by SW:

- FrTot – Total length of a captured frame in display pixels. So the refresh rate of the captured video mode is dfreq_pix / FrTot.
- EpVal – Difference of the length in time of a total capture and a total display line in display pixels (line skew).
- SkewMon – Current skew between capture and display timing in display pixels. This should be approximately the configured target skew, corrected by the regular line skew.

All these values are updated once per frame only and may include jitter due to the different sources for measurement error as described above.

5.7.5.3 Video Layer

In FrameCap#0 set Width and Height to width and height of active capture input area. This must be done, because there is no mode detection. If values do not match the actual mode received, system will operate stable, but with tearing artifacts on all displayed frames.

In ExtSrc#0:

- ClipWindowEnable to DISABLE (clipping is not allowed).
- Setup the pixel format that is received on the external capture input interface: olorComponentBits and ColorComponentShift.
- For components with null bits (not present in the capture input) set ConstantColorRed/Green/Blue/Alpha.

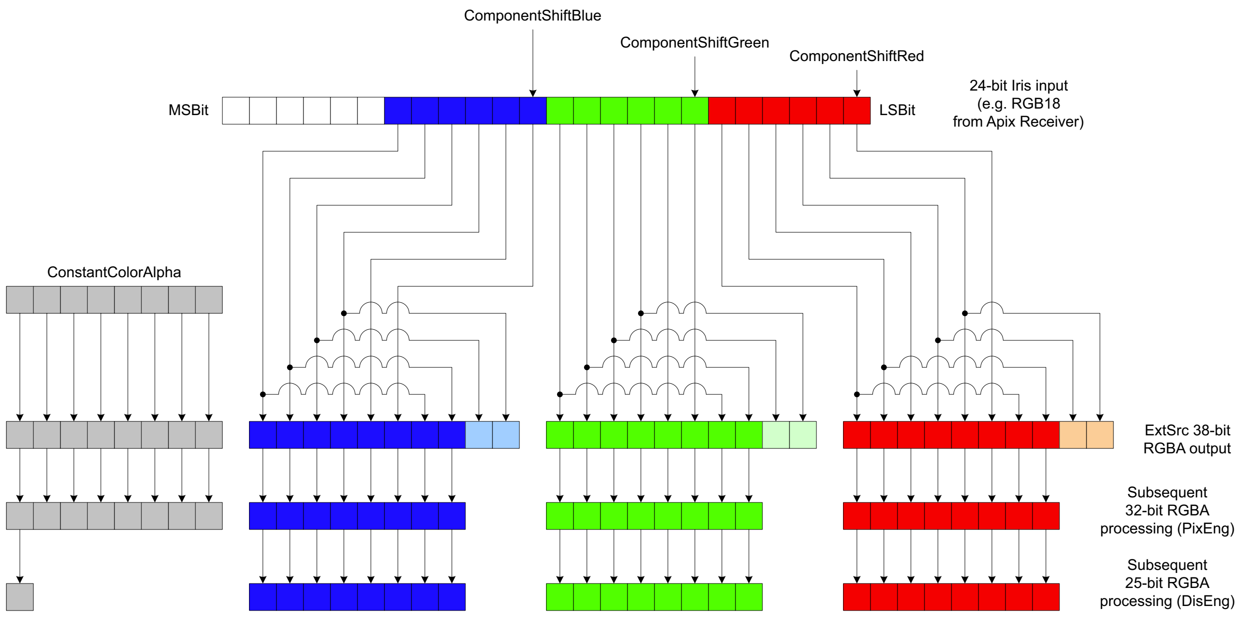

ComponentShiftBlue
ComponentShiftGreen
ComponentShiftRed
MSBit
LSBit
24-bit Iris input (e.g. RGB18 from Apix Receiver)
ConstantColorAlpha
ExtSrc 38-bit RGBA output
Subsequent 32-bit RGBA processing (PixEng)
Subsequent 25-bit RGBA processing (DisEng)

Figure 5-32: Pixel Format Capture Input

Components smaller than 8-bits in the capture input are up-scaled for subsequent processing. The following table gives some typical examples:

Table 5-15: Capture Pixel Format Examples

	Apix RGB24	Apix RGB18	Apix RGB12	Apix RGB10
ComponentShiftRed	0	0	0	0
ComponentShiftGreen	8	6	4	3
ComponentShiftBlue	16	12	8	7
ComponentShiftAlpha	x	x	x	x
ComponentBitsRed	8	6	4	3
ComponentBitsGreen	8	6	4	3
ComponentBitsBlue	8	6	4	3
ComponentBitsAlpha	0	0	0	0

5.7.5.4 Foreground Layer

Setup a Display Buffer for Fetch#0 (at least buffer and pixel format and clip window).

Fetch#0 and CLuT#0 can be used to setup a Color Palette.

The foreground (= secondary) layer is blended onto the video (= primary) layer by Blend Operation with LayerBlend#0.

5.7.5.5 Sprite Layer

Same as for the Memory Stream Sprite Layer, but that it is blended onto the video instead of background layer.

5.7.6 Display Buffer

5.7.6.1 AXI Setup

Related topic: “5.4.5.1.1 AXI Settings” Function

In general smaller burst lengths increase the Iris peak performance. Larger burst length, however, typically increase the system performance for most use cases (due to DDR memory access efficiency and issuing capabilities of the AXI interconnect).

The following settings are required to guarantee tearing-free display operation for all specified use cases (see Display Controller Limitations):

- FetchSprite (SetBurstLength / SetNumBuffers)
  - 2 / 8 (always horizontal scan direction)
- FetchRLD (SetBurstLength / SetNumBuffers)
  - 4 / 4 for horizontal scan direction.
  - 1 / 16 for vertical scan direction (swap).

5.7.6.2 Buffer Format

Related topic: “5.4.5.1.2 Source Buffer Formats” Function

Fetch unit:

- Source buffer size and position: BaseAddress, Stride, LineCount, LineWidth, BitsPerPixel.

Set-up must consider the following restrictions:

The following settings for FetchRLD unit in the Display Controller must not be used in combination (does not affect any relevant use case):

1. BitsPerPixel = 24 (source buffer with 24 bpp).
2. DeltaX < 0.0 (inverted scan in horizontal direction from right to left).
3. DeltaX > -1.0 (pixel replication in horizontal direction).

5.7.6.3 Pixel Format

Related topic: “5.4.5.1.3 Pixel Formats” Function

Fetch unit:

- Pixel format: ComponentBitsRed/Green/Blue/Alpha and ComponentShiftRed/Green/Blue/ Alpha.
- Constant color ConstantColorRed/Green/Blue/Alpha for null sized components.

5.7.6.4 Clip and Skip Window

Related topic:“5.4.5.1.4 Clip and Skip Window” Function

Fetch unit:

- Clip area of the buffer to be displayed: FrameWidth, FrameHeight, FrameXOffset, FrameYOffset.
  This defines the frame size for processing in subsequent processing units.
- Skip area: SkipWindowHeigh, SkipWindowWidth, SkipWindowXOffset, SkipWindowYOffset. Optionally skip window inversion: SkipInvert.
- Select color for outside clip and inside skip pixels: TileMode and DummySkipSelect.

5.7.6.5 Global Alpha

Related topic: “5.4.5.1.5 Global Alpha” Function

Fetch unit:

- Set AlphaMultiply to ENABLE.
- Set global alpha value to ConstantColorAlpha.

5.7.6.6 Constant Color

To generate a constant color frame without access to any memory resource:

- Set ConstantColor register.
- Set TileMode to TILE_FILL_CONSTANT.
- Set FrameXOffset / FrameYOffset to -1 and DeltaX / DeltaY to 0.
- Set frame dimension to FrameHeight / FrameWidth.

5.7.6.7 Multiply Modes

Related topic: “5.4.5.1.7 Multiply Modes” Function

Pre-multiply R, G and B with per-pixel alpha (Fetch unit):

- Set ColorMultiplySelect to ALPHA.

Multiply R, G, B and Alpha with constant factors:

- Set ColorMultiplySelect to CONSTANTCOLOR.
- Set factor for each component to ConstantColorRed/Green/Blue/Alpha (0 - > 0.0, 255 -> 1.0).

NOTE Both modes are applied after Global Alpha.

5.7.6.8 Transparent Color

Related topic: “5.4.5.1.6 Transparent Color” Function

Fetch unit:

- Set TransparentColorEnable to ENABLE.
- Set transparent color toTransparentColorRed/Green/Blue.

Replacement of alpha value for transparent colored pixels occurs before Multiply Modes are applied.

5.7.6.9 Run-Length Decoding

Related topic: “5.4.5.2.2 Run-Length Decoder” Function

Fetch unit:

- Set RLDEnable to ENABLE.
- Set start address of the encoded data stream toBaseAddress and its length to RLEWords.
- Set LineCount, LineWidth and BitsPerPixel according to the encode frame (note that Stride setting has no effect).

When the compression format differs from the TGA specification, the following modifications can be configured:

- EndianSwap to change the byte endianess in the encoded data stream.
- RLDCmdComp to change the header word format.

Using a clip window for RL encoded display buffers is not allowed! It must be setup to

- FrameWidth/Height to dimension of the encoded frame.
- X/YOffset to 0/0.

5.7.6.10 Scan Directions

Related topic: “5.4.5.2.1 Scan Directions” Function

Horizontal flip (horizontal scan direction):

- FetchSprite: not allowed.
- Other Fetch units: Set XOffset to right-most pixel and DeltaX to -1.

Vertical flip (horizontal scan direction):

- FetchSprite: not allowed.
- Other Fetch units: Set YOffset to bottom-most line and DeltaY to -1.

Swap (vertical scan direction):

- FetchSprite: not allowed.
- Other Fetch units: EnableFrameSwapDirections.

Any combination of the above is allowed.

For RL-encoded buffers none of the above is allowed.

Note that for vertical scan direction the achievable pixel rate significantly drops, because one read burst per pixel is required in any case. Particularly the 24 bpp pixel format should be avoided for that case because it may result in even two read bursts per pixel for certain columns.

Tearing-free display operation according to Display Controller Limitations is guaranteed for horizontal scan direction only.

Set-up must consider the following restrictions:

The following settings for FetchRLD unit in the Display Controller must not be used in combination (does not affect any relevant use case):

1. BitsPerPixel = 24 (source buffer with 24 bpp).
2. DeltaX < 0.0 (inverted scan in horizontal direction from right to left).
3. DeltaX > -1.0 (pixel replication in horizontal direction).

5.7.6.11 Simple Scaling

Related topic: “ Simple Scaling” Function

Pixel replication (up-scale by factor 2 or 4):

- Set DeltaX (horizontal scale) and/or DeltaY (vertical) to 0.5 or 0.25.
- Increase output frame dimension (FrameWidth and FrameHeight) by factor 2 or 4.

Pixel dropping (down-scale by factor 2 or 4):

- Set DeltaX (horizontal scale) and/or DeltaY (vertical) to 2.0 or 4.0.
- Decrease output frame dimension (FrameWidth and FrameHeight) by factor 2 or 4.

Horizontal and vertical scale can be setup independently.

Optionally the start phase can be adjusted for both modes with a granularity of 0.25 pixels:

- Set FrameXOffset / FrameXOffsetTwoDecimalPlaces and/or FrameYOffset / FrameYOffsetTwoDecimalPlaces
- When this may result in pixels being sampled outside the source buffer at image borders, setup a TileMode.

Tearing-free display operation according to Display Controller Limitations is guaranteed for up-scale modes only.

Set-up must consider the following restrictions.

The following settings for FetchRLD unit in the Display Controller must not be used in combination (does not affect any relevant use case):

1. BitsPerPixel = 24 (source buffer with 24 bpp).
2. DeltaX < 0.0 (inverted scan in horizontal direction from right to left).
3. DeltaX > -1.0 (pixel replication in horizontal direction).

5.7.6.12 Sprites

Related topic: “5.4.5.2.3 Sprites” Function

Fetch#1 setup:

- Set FrameWidth and FrameHeight to the dimension of the sprite layer that is generated and processed by subsequent processing units.
- Set ConstantColor. This color is used to generate a background for the sprite layer. Most typically the sprite layer is blended onto some other layer, so ConstantColorAlpha should be set to 0, making the area between sprites fully transparent. Constant RGB settings don’t have effect then. Alternatively this can be achieved by just setting SpriteBackgroundSelect to ZERO.
- Set pixel format, which is the same for all sprites: SpriteBitsPerPixel, SpriteComponentBits<col>, SpriteComponentShift<col>.
- Set SpriteIndexBits to use multiple color palettes for groups of sprites (see also Color Palette).

For each sprite to display:

- Set base address and dimension for sprite image data: Sprite<n>Address, Sprite<n>XWidth, Sprite<n>YWidth.
- Set sprite position relative to dimension of the complete sprite layer: Sprite<n>XOffset, Sprite<n>YOffset.

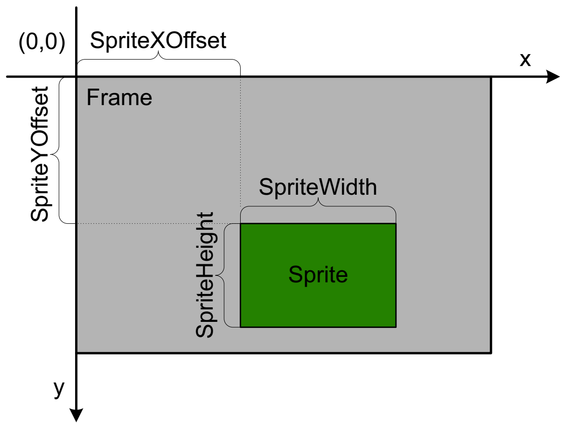


Figure 5-33: Sprite Layer

The stride for each sprite results implicitly form the width and bits-per-pixel setting:

- stride = int((width * bpp + 7) / 8)

Areas of sprites that exceed the background area, are cut off.

Areas of sprites that overlap other sprites are printed with a fixed priority: Lower index value on top.

5.7.7 Image Processing

5.7.7.1 Color Palette

Related topic: “5.4.5.9.2 Index Lookup” Function

To configure a color palette:

- Set (Sprite)BitsPerPixel and (Sprite)ComponentBitsRed of Fetch unit and IDX_BITS of CLuT to the size of the color index used (max 8 bits) and (Sprite)ComponentShiftRed of Fetch to 0 (red channel is used for the index value).
- Set CLuT MODE to INDEX_10BIT.
- Program palette values into CLuT#0: Set R_EN, G_EN and B_EN to ENABLE and write RGB values to LUT fields. 10 bits can be programmed per component.

When alpha information is needed, it can be stored in memory together with the RGB index value:

- Increase (Sprite)BitsPerPixel accordingly and set up (Sprite)ComponentBitsAlpha and (Sprite)ComponentShiftAlpha.

The CLuT will by-pass that alpha unchanged. Alternatively the alpha can be store into the look-up table together with indexed RGB at cost of alpha being limited to 6 bits:

- Set MODE of CluT#0 to *INDEX_RGBA*. In that case the mapping of programmed RGB values to RGBA output is as follows:

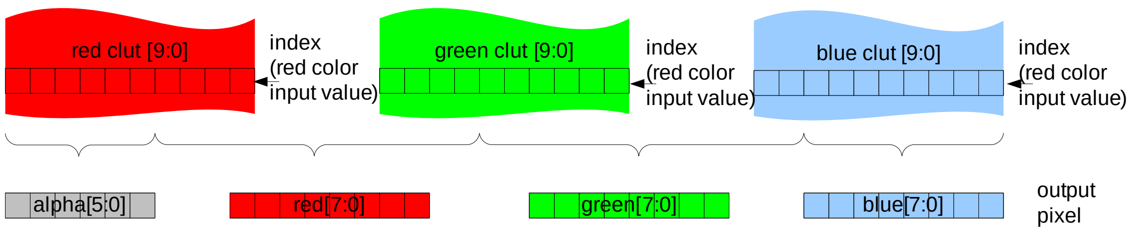


Figure 5-34: Color Mapping RGBA Index Mode

For sprite layers it is possible to store multiple palettes with less colors each for different sets of sprites into the look-up table:

- Set SpriteIndexBits of Fetch. This is the number of upper bits of the sprite index that is used for upper bits of the palette index. The lower palette index bits are set to the index value read from memory. Example:

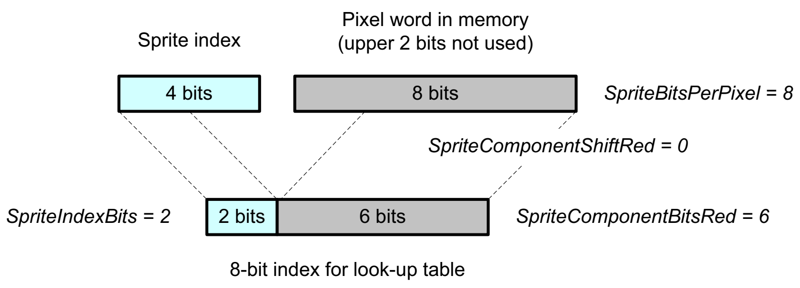


Figure 5-35: Sprite Index Palette Mapping

In this example, four palettes with 64 colors each (6-bit index) can be programmed for sprites 0..3, 4..7, 8..11 and 12..15.

5.7.7.2 Linear Transformation

Related topic: “5.4.5.8 Color Matrix” Function

Matrix setup:

- Set MODE field to MATRIX.
- Set matrix elements A11..A33 and offset values C1..C3.

5.7.7.3 Non-linear Transformation

Related topic: Color Lookup Function

CLuT setup:

- Set MODE field to LUT.
- Enable color channels for write access (R_EN, G_EN, B_EN).
- For each color channel (RGB) and each input 8-bit code program the corresponding output 10-bit code into the lookup table (LUT fields).

Note that the lookup table is not shadowed. When re-programming it during active area of output timing, single output pixels may be bypassed and unchanged by the CLuT. To prevent this along with tearing artifacts, the table must be programmed during vertical blank only (see Dynamic Control Flow).

5.7.7.4 Blend Operation

Related topic: "5.4.5.5 Layer Blend Unit" Function

LayerBlend setup:

- Set MODE to BLEND.
- Setup blending function: SEC/PRIM_A_BLD_FUNC and SEC/PRIM_C_BLD_FUNC.
- Setup a constant alpha value to ALPHA when used by the function.
- Setup the position of the secondary layer in relation to the primary one: XPOS/YPOS. Its size results implicitly from the corresponding input layer.

The secondary layer must be completely located inside the area of the primary layer! To achieve a setup where the secondary layer partly disappears at borders of the primary one, SW must use clip window functionality for the secondary layer (display buffer setup). Otherwise tearing artifacts may appear on the display.

5.7.7.5 Alpha Masking

Certain features in the processing path can be enabled individually for each pixel by evaluating its alpha value, which is used as a bit mask then.

In the Pixel Engine, where the alpha channel is 8-bit, the mask is computed to

- alpha < 128 => mask = 0, alpha ≥ 128 => mask = 1

In the Display Engine the alpha channel is 1 bit only anyway. At the crossing from the Pixel Engine the reduction to 1-bit is computed by the same formula as above.

Using this mask value the following operations can be controlled:

- Linear color transformation: AlphaMask and AlphaInvert of Matrix.
- Non-Linear color transformation: AlphaMask and AlphaInvert of CLuT.
- Dithering: alpha_mode of Dither unit.
- Signature computation: AlphaMask and AlphaInv of Sig#0/1/2/3.

The following sources are possible for the required bit mask:

- Store 1-bit alpha per pixel of a display layer in memory and set blend mode in LayerBlend to pass it unchanged.

- Use LayerBlend unit to generate a bit mask according to area of blended foreground layer: Set AlphaMaskEnable to ENABLE and AlphaMaskMode:

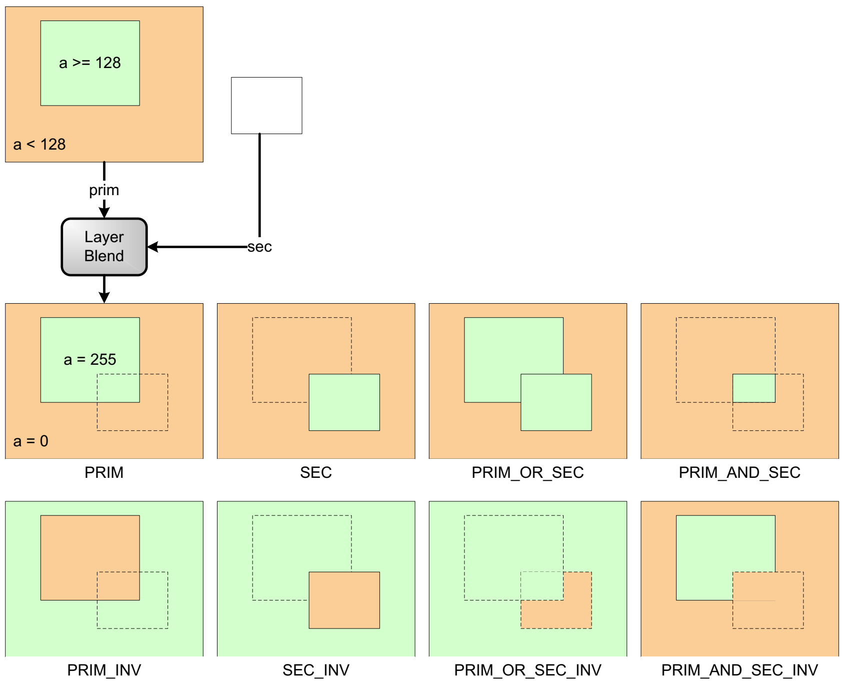


Figure 5-36: Alpha Mask Generation with Display Layers

The logical options with the *AlphaMaskMode* in LayerBlend allows to form different sort of combinations of multiple layers.

5.7.8 Safety Features

Related topic: “5.4.4.4 Safety Features” Function

5.7.8.1 Signature

Related topic: “5.4.5.12 Signature Unit” Function

This Iris derivative implements four Signature units that can be setup and operate completely independent from each other. The following describes the setup for any of them.

First a decision must be made where to set the tap for monitoring frame data:

- Set src_select field of Display Engine accordingly.

When setting to *DITHER*, the Dither unit must not be setup for temporal dithering! When the Matrix and CLuT are used to implement user controlled features in the monitored area (standard color controls, gamma, etc), it’s recommended to set it to *FRAMEGEN*, otherwise different reference values for any sort of user setup must be provided.

Second the area to monitor must be defined:

- Set EnEvalWin to ENABLE and registers EvalUpperLeft/EvalLowerRight. The evaluation window is relative to the total frame layout, including blanking intervals. It is a mandatory setup.
- Optionally the skip window can be configured: Set EnSkipWin to ENABLE and register SkipUpperLeft/SkipLowerRight.
- Optionally the per-pixel alpha values of the monitored frames can be used to mask pixels individually for signature computation (see Alpha Masking).

The color of a pixel is considered for signature computation when all of the three conditions above (when enabled) are full-filled.

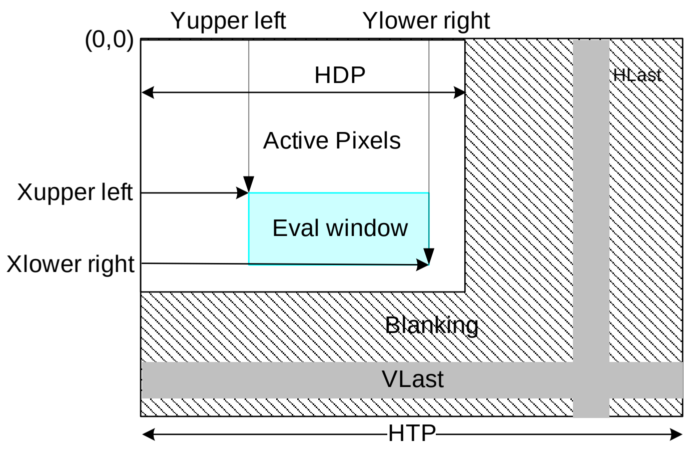


Figure 5-37: Signature Evaluation Window Setup

Next the kind of computation must be defined:

- Set EnSUM and/or EnCRC to ENABLE.

The first step during application development is then to determine the reference signature for a certain image and setup:

- Set SigMode to ONE_TIME.
- Set SigEN to ENABLE.
- When image is displayed and considered to be correct: Write ‘1’ to Kick. This will start measurement for the next frame.

- Either poll Valid field until status is VALID or wait for measurement complete interrupt of the Sig unit.

Now, the computed signature values can be read from SigCRCRed/Green/Blue and/or SigSumRed/Green/Blue fields until next measurement is started by SW.

The measured value can be used as reference value for operation of the final application. To check it automatically against each frame:

- Set SigMode to CYCLIC.
- Set SigEN to ENABLE.
- Set reference value to SigCRCRefR/G/B and/or SigSumRefR/G/B.
- When summation is used, define a tolerance for the check against reference: ThrSumR/G/B.
- Setup a hysteresis to control how many consecutive frames with failing reference check will set

SigError status to VIOLATION (ErrThres) and how many consecutive frame with passing reference check will reset it to NO_VIOLATION (ErrThresReset).

- Start cyclic measurement: Write ‘1’ to Kick.

SW can use the signature error interrupt to react to a signature violation or it can poll SigError status.

To stop measurement and checking:

- Set SigEN to DISABLE.

Typically image content to monitor is not static, so new reference values must be loaded regularly during operation. There are two different approaches to implement this.

Synchronized (can be used when image content changes synchronously to some other register change in the pipeline, e.g. a display buffer address or some overlay offset):

- Enable ShdEn and set ShdLdSel to Command (shadow load token of the monitored frames will load shadow registers of the Sig unit).
- Program new reference values together with other dynamic settings in the display path.
- Generate a shadow load token for the display path (see also Shadow Registers Function and Dynamic Control Flow Setup).

Unsynchronized:

- Enable ShdEn and set ShdLdSel to SW_Token. (Sig unit must generate its own shadow load token).
- Program new reference values at any time.
- Write ‘1’ to ShdTokGen close as possible to the time where image content will change.
- Wait for shadow load interrupt of Sig unit.

With that procedure there is no defined correlation between the actual frame where image content and where the signature reference changes, so ErrThres must be setup with some tolerance to avoid false signature violation signal.

5.7.8.2 Panic Mode

Two different kind of panic modes can be configured.

The global panic mode can switch the display mode of the Frame Generator during operation:

- Set FgDmPanic to a mode that shall be active during global panic

One of the following conditions will activate global panic:

- The external panic status gets active (see chapter “2.7.1 Panic Switch”)
- Global panic is signaled by any Signature unit in response to a signature error. Sig setup:
  - Set EnPanic to ENABLE.

- Set ObjectPanic to DISPLAY.

In first case, the normal display mode (FgDm) is restored as soon as the external panic status is released.

In second case (can be detected by evaluating PanicFlag status of the Sig unit), it must explicitly be reset by SW:

- Write ‘1’ to PanicFlag.

The local panic mode can switch the evaluation window only of a specific signature unit to constant color:

- Set SigPanicColor of Display Engine to overlay color for local panic.
- Set EnPanic to ENABLE and ObjectPanic to OBJECT in Sig unit.

Local panic can be signaled by Signature units only. Resetting local panic works same as for global panic.

Note, that when signature computation is restricted to a subset of the evaluation window pixels by setting up a skip window or using the alpha mask feature, this has no impact on what pixels are switched to constant color during local panic.

Both global and local panic modes can operate in parallel. Each Signature unit, however, can activate only one of them.

While local panic is more robust, it is less flexible than global panic. For example, a global panic mode setup can switch from capture to memory stream display, showing some static failure screen from memory. However, display output is still undefined in case there is a malfunction also in the memory stream.

5.7.9 Tweaking

5.7.9.1 Single Buffer Foreground Layer

The standard flow for single display buffer setups allows to modify buffer content during the vertical blanking interval of the output timing (see Dynamic Control Flow).

In case of foreground layers that have less height than the background layer, this can be optimized in order to gain time for rendering operations:

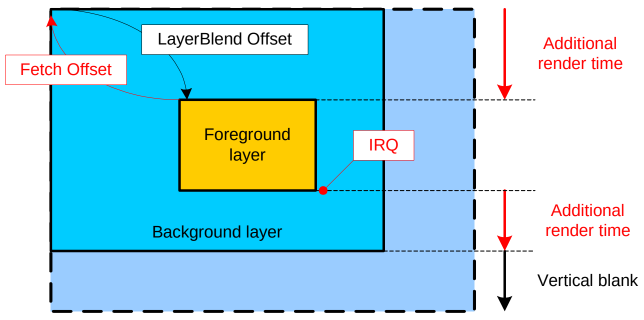


Figure 5-38: Single Buffer Foreground Layer

Because the first pixels of a foreground layer are fetched already from memory at start of active output area already, a special kind of setup is required to prevent this:

- Setup a Programmable Interrupt to the position of the last foreground layer pixel. This is the start trigger for render operation on the corresponding buffer.
- Set FrameWidth/FrameHeight of the foreground layer to the same values as for the background.

- Independent from the overlay position, always set XPOS/YPOS of LayerBlend to 0/0.
- Instead set the negative of the overlay position to FrameXOffset/FrameYOffset of the Fetch unit.
- Set TileMode to TILE_FILL_ZERO (= fully transparent).

By this the fore- and background layer are actually same in size and position, however, area outside the overlay are sampled in the tile region using constant pixel color with alpha=0, blocking any memory access until low before the overlay display data really starts.

If the display buffer is larger than the area that should be overlaid (clipping), then the skip instead of the clip window together with SkipInvert = Inverted and DummySkipSelect = ZERO must be used (see also Clip and Skip Window).

5.7.10 Iris-MVL Register Overview

5.7.10.1 Iris-MVL - Global Control

Table 5-16: Registers Overview

Base Address(es)	Instance no 0: BASEADDR="00030000"	
Absolute Address	**Register Name**	**Register Description**
BASEADDR + 0x0000	LockUnlock	Register to lock or unlock write access to registers of this unit with lock property.
BASEADDR + 0x0004	LockStatus	Lock status for write access to registers of this unit with lock property.
BASEADDR + 0x0008	IPIdentifier 🔒	IP Identifier for this IRIS derivate, needs to be unlocked.
BASEADDR + 0x0010	InterruptEnable	Interrupt Enable register
BASEADDR + 0x0014	InterruptPreset	Interrupt Preset register
BASEADDR + 0x0018	InterruptClear	Interrupt Clear register
BASEADDR + 0x001C	InterruptStatus	Interrupt Status register

5.7.10.2 Iris-MVL - Pixelbus

Table 5-17: Registers Overview

Base Address(es)	Instance no 0: BASEADDR="00030800"	
Absolute Address	**Register Name**	**Register Description**
BASEADDR + 0x0000	fetch0_cfg	pixelbus configuration for unit fetch0
BASEADDR + 0x0004	fetch1_cfg	pixelbus configuration for unit fetch1
BASEADDR + 0x0008	extsrc0_cfg	pixelbus configuration for unit extsrc0
BASEADDR + 0x000C	extdst0_cfg	pixelbus configuration for unit extdst0
BASEADDR + 0x0010	extdst1_cfg	pixelbus configuration for unit extdst1
BASEADDR + 0x0014	clut0_cfg	pixelbus configuration for unit clut0
BASEADDR + 0x0018	layerblend0_cfg	pixelbus configuration for unit layerblend0
BASEADDR + 0x001C	layerblend1_cfg	pixelbus configuration for unit layerblend1
BASEADDR + 0x0020	Request_Sequence_Complete	Pixel Engine request sequence complete register
BASEADDR + 0x0024	Synchronization_Mode	Pixel Engine synchronizer mode register
BASEADDR + 0x0028	Synchronization_Status	Pixel Engine synchronizer status register
BASEADDR + 0x002C	Synchronization_Trigger	Pixel Engine synchronizer trigger register
BASEADDR + 0x0030	extdst0_clk	extdst0 clock throttling, this value is used if the _clken of a module is configured for automatic mode
BASEADDR + 0x0034	extdst1_clk	extdst1 clock throttling, this value is used if the _clken of a module is configured for automatic mode

5.7.10.3 Iris-MVL - Display Configuration

Table 5-18: Registers Overview

Base Address(es)	Instance no 0: BASEADDR="00033000"	
Absolute Address	**Register Name**	**Register Description**
BASEADDR + 0x0000	PolarityCtrl	Modification of pixel output and its synchronization signals
BASEADDR + 0x0004	SigSrcSelect	Select to observe data stream of submodules of display engine
BASEADDR + 0x0008	SigPanicColor	Pixel component, that will be displayed in case of signature violation and sig0_control.sig0_ObjectPanic = 0x0
BASEADDR + 0x000C	ClockCtrl	Controls generation of display clock signals.

5.7.10.4 Iris-MVL - FetchRLD

Table 5-19: Registers Overview

Base Address(es)	Instance no 0: BASEADDR="00030C00"	
Absolute Address	**Register Name**	**Register Description**
BASEADDR + 0x0000	StaticControl	Fetch unit static control register
BASEADDR + 0x0004	BurstBufferManagement	AXI interface buffer management register
BASEADDR + 0x0008	BaseAddress	Source buffer base address
BASEADDR + 0x000C	SourceBufferStride	Source buffer stride
BASEADDR + 0x0010	SourceBufferAttributes	Source buffer attributes
BASEADDR + 0x0014	SourceBufferLength	Source Buffer Length for Run Length Decoding
BASEADDR + 0x0018	FrameXOffset	Frame X offset
BASEADDR + 0x001C	FrameYOffset	Frame Y offset
BASEADDR + 0x0020	FrameDimensions	Defines frame rectangle
BASEADDR + 0x0024	DeltaXX	DeltaXX stepsize
BASEADDR + 0x0028	DeltaYY	DeltaYY stepsize
BASEADDR + 0x002C	SkipWindowOffset	Skip window offset
BASEADDR + 0x0030	SkipWindowDimensions	Defines skip window rectangle, set to (0,0) when SkipInvert is set to Normal to not skip any pixels at all (disable of skip window) or set to (0,0) when SkipInvert is set to Inverted to skip all pixels inside the sourcebuffer (useful to create a constant color background).
BASEADDR + 0x0034	ColorComponentBits	Color component size of source buffer
BASEADDR + 0x0038	ColorComponentShift	Color component offset of source buffer
BASEADDR + 0x003C	ConstantColor	Constant color settings. These constant color values are required for tiling mode TILE_FILL_CONSTANT, skip mode CONSTANTCOLOR, fetchsprite background and if a color component bit width is set to 0.
BASEADDR + 0x0040	TransparentColor	Transparent color settings. These transparent color values are required for the transparent color feature. When the TransparentColorEnable is set to ENABLE then every pixel matching these transparent color components will get an alpha value of 0 and 255 otherwise. Please give each color component right aligned. Only the ColorComponentBits LSBs are evaluated, all others are ignored.
BASEADDR + 0x0044	Control	Fetch unit main control register
BASEADDR + 0x0048	ControlTrigger	Fetch unit trigger register
BASEADDR + 0x004C	Start	Fetch unit start register
BASEADDR + 0x0050	FetchType	Fetch unit type register
BASEADDR + 0x0054	BurstBufferProperties	Burst Buffer Property register
BASEADDR + 0x0058	Reserved	Do not modify

5.7.10.5 Iris-MVL - FetchSprite

Table 5-20: Registers Overview

Base Address(es)	Instance no 0: BASEADDR="00031000"	
Absolute Address	**Register Name**	**Register Description**
BASEADDR + 0x0000	StaticControl	Fetch unit static control register
BASEADDR + 0x0004	BitsPerPixel	Sets bits per pixel for all sprite source buffers.
BASEADDR + 0x0008	ColorComponentBits	Color component size of all source buffers of sprites
BASEADDR + 0x000C	ColorComponentShift	Color component offset of all source buffers of sprites
BASEADDR + 0x0010	Sprite00Address	BaseAddress of Sprite0
BASEADDR + 0x0014	Sprite00Dimension	Dimensions of Sprite0
BASEADDR + 0x0018	Sprite01Address	BaseAddress of Sprite1
BASEADDR + 0x001C	Sprite01Dimension	Dimensions of Sprite1
BASEADDR + 0x0020	Sprite02Address	BaseAddress of Sprite2
BASEADDR + 0x0024	Sprite02Dimension	Dimensions of Sprite2
BASEADDR + 0x0028	Sprite03Address	BaseAddress of Sprite3
BASEADDR + 0x002C	Sprite03Dimension	Dimensions of Sprite3
BASEADDR + 0x0030	Sprite04Address	BaseAddress of Sprite4
BASEADDR + 0x0034	Sprite04Dimension	Dimensions of Sprite4
BASEADDR + 0x0038	Sprite05Address	BaseAddress of Sprite5
BASEADDR + 0x003C	Sprite05Dimension	Dimensions of Sprite5
BASEADDR + 0x0040	Sprite06Address	BaseAddress of Sprite6
BASEADDR + 0x0044	Sprite06Dimension	Dimensions of Sprite6
BASEADDR + 0x0048	Sprite07Address	BaseAddress of Sprite7
BASEADDR + 0x004C	Sprite07Dimension	Dimensions of Sprite7
BASEADDR + 0x0050	Sprite08Address	BaseAddress of Sprite8
BASEADDR + 0x0054	Sprite08Dimension	Dimensions of Sprite8
BASEADDR + 0x0058	Sprite09Address	BaseAddress of Sprite9
BASEADDR + 0x005C	Sprite09Dimension	Dimensions of Sprite9
BASEADDR + 0x0060	Sprite10Address	BaseAddress of Sprite10
BASEADDR + 0x0064	Sprite10Dimension	Dimensions of Sprite10
BASEADDR + 0x0068	Sprite11Address	BaseAddress of Sprite11
BASEADDR + 0x006C	Sprite11Dimension	Dimensions of Sprite11
BASEADDR + 0x0070	Sprite12Address	BaseAddress of Sprite12
BASEADDR + 0x0074	Sprite12Dimension	Dimensions of Sprite12
BASEADDR + 0x0078	Sprite13Address	BaseAddress of Sprite13
BASEADDR + 0x007C	Sprite13Dimension	Dimensions of Sprite13
BASEADDR + 0x0080	Sprite14Address	BaseAddress of Sprite14
BASEADDR + 0x0084	Sprite14Dimension	Dimensions of Sprite14
BASEADDR + 0x0088	Sprite15Address	BaseAddress of Sprite15
BASEADDR + 0x008C	Sprite15Dimension	Dimensions of Sprite15
BASEADDR + 0x0090	BurstBufferManagement	AXI interface buffer management register
BASEADDR + 0x0094	SpriteEnable	Enables for each sprite
BASEADDR + 0x0098	Sprite00Offset	Offset of Sprite0 relative to output frame origin
BASEADDR + 0x009C	Sprite01Offset	Offset of Sprite1 relative to output frame origin
BASEADDR + 0x00A0	Sprite02Offset	Offset of Sprite2 relative to output frame origin
BASEADDR + 0x00A4	Sprite03Offset	Offset of Sprite3 relative to output frame origin
BASEADDR + 0x00A8	Sprite04Offset	Offset of Sprite4 relative to output frame origin
BASEADDR + 0x00AC	Sprite05Offset	Offset of Sprite5 relative to output frame origin
BASEADDR + 0x00B0	Sprite06Offset	Offset of Sprite6 relative to output frame origin
BASEADDR + 0x00B4	Sprite07Offset	Offset of Sprite7 relative to output frame origin
BASEADDR + 0x00B8	Sprite08Offset	Offset of Sprite8 relative to output frame origin
BASEADDR + 0x00BC	Sprite09Offset	Offset of Sprite9 relative to output frame origin
BASEADDR + 0x00C0	Sprite10Offset	Offset of Sprite10 relative to output frame origin

Table 5-20: Registers Overview (Continued)

Base Address(es)	Instance no 0: BASEADDR="00031000"	
Absolute Address	**Register Name**	**Register Description**
BASEADDR + 0x00C4	Sprite11Offset	Offset of Sprite11 relative to output frame origin
BASEADDR + 0x00C8	Sprite12Offset	Offset of Sprite12 relative to output frame origin
BASEADDR + 0x00CC	Sprite13Offset	Offset of Sprite13 relative to output frame origin
BASEADDR + 0x00D0	Sprite14Offset	Offset of Sprite14 relative to output frame origin
BASEADDR + 0x00D4	Sprite15Offset	Offset of Sprite15 relative to output frame origin
BASEADDR + 0x00D8	FrameDimensions	Defines frame rectangle
BASEADDR + 0x00DC	ConstantColor	Constant color settings. These constant color values are required for tiling mode TILE_FILL_CONSTANT, skip mode CONSTANTCOLOR, fetchsprite background and if a color component bit width is set to 0.
BASEADDR + 0x00E0	TransparentColor	Transparent color settings. These transparent color values are required for the transparent color feature. When the TransparentColorEnable is set to ENABLE then every pixel matching these transparent color components will get an alpha value of 0 and 255 otherwise. Please give each color component right aligned. Only the ColorComponentBits LSBs are evaluated, all others are ignored.
BASEADDR + 0x00E4	Control	Fetch unit main control register
BASEADDR + 0x00E8	ControlTrigger	Fetch unit trigger register
BASEADDR + 0x00EC	Start	Fetch unit start register
BASEADDR + 0x00F0	FetchType	Fetch unit type register
BASEADDR + 0x00F4	BurstBufferProperties	Burst Buffer Property register
BASEADDR + 0x00F8	Reserved	Do not modify

5.7.10.6 Iris-MVL - ExtSrc

Table 5-21: Registers Overview

Base Address(es)	**Instance no 0: BASEADDR="00031400"**	
Absolute Address	**Register Name**	**Register Description**
BASEADDR + 0x0000	StaticControl	ExtSrc static control register
BASEADDR + 0x0004	ClipWindowOffset	Clip window offset, to generate a clipping of the frame. It has to be within the input frame.
BASEADDR + 0x0008	ClipWindowDimension	Define the clip window dimension. If the clip window feature is enabled this dimension is used for the new frame dimension. Note that the clip window has to be smaller or equal to the original frame dimensions. The new frame has to be within the active area of the original frame.
BASEADDR + 0x000C	ColorComponentBits	Color component size of raw input data. Please note that the width must be equal or lower than the output width.
BASEADDR + 0x0010	ColorComponentShift	Color component offset of raw input data.
BASEADDR + 0x0014	ConstantColorRedGreen	Constant color settings for Red and Green channel. These constant color values are required if a color component bit width is set to 0.
BASEADDR + 0x0018	ConstantColorBlueAlpha	Constant color settings for Blue and Alpha channel. These constant color values are required if a color component bit width is set to 0.
BASEADDR + 0x001C	TransparentColor	Transparent color settings. These transparent color values are required for the transparent color feature. When the TransparentColorEnable is set to ENABLE then every pixel matching these transparent color components will get an alpha value of 0 and 255 otherwise. Please give each color component right aligned. Only the ColorComponentBits LSBs are evaluated, all others are ignored.
BASEADDR + 0x0020	Control	ExtSrc unit main control register
BASEADDR + 0x0024	ControlTrigger	ExtSrc unit trigger token generation
BASEADDR + 0x0028	Start	ExtSrc unit start register
BASEADDR + 0x002C	Reserved	Do not modify
BASEADDR + 0x0030	Reserved	Do not modify

5.7.10.7 Iris-MVL - CLuT

Table 5-22: Registers Overview

Base Address(es)	**Instance no 0: BASEADDR0="00032000" Instance no 1: BASEADDR1="00033C00"**	
Absolute Address	**Register Name**	**Register Description**
BASEADDRx + 0x0000	StaticControl	CLUT static control register
BASEADDRx + 0x0004	UnshadowedControl	CLUT unshadowed control register
BASEADDRx + 0x0008	Control	CLUT control register
BASEADDRx + 0x000C	Status	CLUT status register
BASEADDRx + 0x0010	LastControlWord	Value of last received control word, for debugging
BASEADDRx + 0x0400	LUT	Look Up Table

5.7.10.8 Iris-MVL - Matrix

Table 5-23: Registers Overview

Base Address(es)	Instance no 0: BASEADDR="00033800"	
Absolute Address	**Register Name**	**Register Description**
BASEADDR + 0x0000	StaticControl	Color Matrix static control register
BASEADDR + 0x0004	Control	Color Matrix control register
BASEADDR + 0x0008	Red0	Matrix values for calculation of the red output value
BASEADDR + 0x000C	Red1	Matrix values for calculation of the red output value
BASEADDR + 0x0010	Green0	Matrix values for calculation of the green output value
BASEADDR + 0x0014	Green1	Matrix values for calculation of the green output value
BASEADDR + 0x0018	Blue0	Matrix values for calculation of the blue output value
BASEADDR + 0x001C	Blue1	Matrix values for calculation of the blue output value
BASEADDR + 0x0020	LastControlWord	Value of last received control word, for debugging

5.7.10.9 Iris-MVL - LayerBlend

Table 5-24: Registers Overview

Base Address(es)	Instance no 0: BASEADDR0="00032800" Instance no 1: BASEADDR1="00032C00"	
Absolute Address	**Register Name**	**Register Description**
BASEADDRx + 0x0000	StaticControl	Layer Blend static control register
BASEADDRx + 0x0004	Control	Layer Blend control register
BASEADDRx + 0x0008	Position	Position of secondary (overlay) input frame
BASEADDRx + 0x000C	PrimControlWord	Value of last received primary (background) control word, for debugging
BASEADDRx + 0x0010	SecControlWord	Value of last received secondary (overlay) control word, for debugging

5.7.10.10 Iris-MVL - ExtDst

Table 5-25: Registers Overview

Base Address(es)	Instance no 0: BASEADDR0="00031800" Instance no 1: BASEADDR1="00031C00"	
Absolute Address	**Register Name**	**Register Description**
BASEADDRx + 0x0000	StaticControl	External Destination output control
BASEADDRx + 0x0004	SoftwareKick	External Destination software kick
BASEADDRx + 0x0008	Status	External Destination Unit current status
BASEADDRx + 0x000C	ControlWord	Value of last received control word
BASEADDRx + 0x0010	CurPixelCnt	pixel count of currently running frame
BASEADDRx + 0x0014	LastPixelCnt	pixel count between last two control words
BASEADDRx + 0x0018	PerfCounter	Performance counter result

5.7.10.11 FrameCap

Table 5-26: Registers Overview

Base Address(es)	Instance no 0: BASEADDR="00036800"	
Absolute Address	**Register Name**	**Register Description**
BASEADDR + 0x0000	Ctr	FrameCap Control Register
BASEADDR + 0x0004	Spr	FrameCap Sync Polarity Register
BASEADDR + 0x0008	Fdr	FrameCap Frame Dimension Register
BASEADDR + 0x000C	Kcr	FrameCap Kick Config Register
BASEADDR + 0x0010	Scr	FrameCap Sync Config Register
BASEADDR + 0x0014	Sts	FrameCap Status Register. Shows current status of the FrameCap module.
BASEADDR + 0x0018	StsClr	FrameCap Status Clear Register. Clears the locked status bits in Sts register.
BASEADDR + 0x001C	FRCnt	FrameCap Frame Rate Count Register.

5.7.10.12 Iris-MVL - FrameGen_PS

Table 5-27: Registers Overview

Base Address(es)	Instance no 0: BASEADDR="00033400"	
Absolute Address	**Register Name**	**Register Description**
BASEADDR + 0x0000	LockUnlock	Register to lock or unlock write access to registers of this unit with lock property.
BASEADDR + 0x0004	LockStatus	Lock status for write access to registers of this unit with lock property.
BASEADDR + 0x0008	FgStCtrl	FrameGen Static Control Register
BASEADDR + 0x000C	HtCfg1	FrameGen Horizontal Timing Config Register 1
BASEADDR + 0x0010	HtCfg2	FrameGen Horizontal Timing Config Register 2
BASEADDR + 0x0014	VtCfg1	FrameGen Vertical Timing Config Register 1
BASEADDR + 0x0018	VtCfg2	FrameGen Vertical Timing Config Register 2
BASEADDR + 0x001C	Int0Config	Coordinates of the trigger point for generation of the Int0 interrupt signal
BASEADDR + 0x0020	Int1Config	Coordinates of the trigger point for generation of the Int1 interrupt signal
BASEADDR + 0x0024	Int2Config	Coordinates of the trigger point for generation of the Int2 interrupt signal
BASEADDR + 0x0028	Int3Config	Coordinates of the trigger point for generation of the Int3 interrupt signal
BASEADDR + 0x002C	PKickConfig	Coordinates of the trigger point for generation of the primary kick signal
BASEADDR + 0x0030	SKickConfig	Coordinates of the trigger point for generation of the secondary kick signal
BASEADDR + 0x0034	SecStatConfig	Configuration register for controlling the behaviour of the SecSyncStat field in the FgSecChStat register.
BASEADDR + 0x0038	FgSRCR1	FrameGen Skew Regulation Control Register 1.
BASEADDR + 0x003C	FgSRCR2	FrameGen Skew Regulation Control Register 2
BASEADDR + 0x0040	FgSRCR3	FrameGen Skew Regulation Control Register 3
BASEADDR + 0x0044	FgSRCR4	FrameGen Skew Regulation Control Register 4
BASEADDR + 0x0048	FgSRCR5	FrameGen Skew Regulation Control Register 5
BASEADDR + 0x004C	FgSRCR6	FrameGen Skew Regulation Control Register 6
BASEADDR + 0x0050	FgKSDR	FrameGen Kick System Debug Register
BASEADDR + 0x0054	PaCfg	FrameGen Primary Area Config Register 1 (shadowed)
BASEADDR + 0x0058	SaCfg	FrameGen Secondary Area Config Register 1 (shadowed)

Table 5-27: Registers Overview (Continued)

Base Address(es)	**Instance no 0: BASEADDR="00033400"**	
Absolute Address	**Register Name**	**Register Description**
BASEADDR + 0x005C	FgInCtrl	FrameGen Input Control Register (shadowed)
BASEADDR + 0x0060	FgInCtrlPanic Ï	FrameGen Input Control Panic Register (shadowed)
BASEADDR + 0x0064	FgCCR	FrameGen Constant Color Register (shadowed)
BASEADDR + 0x0068	FgEnable	FrameGen Enable Register
BASEADDR + 0x006C	FgSlr	FrameGen Shadow Load Register
BASEADDR + 0x0070	FgEnSts	FrameGen Enable Status Register
BASEADDR + 0x0074	FgChStat	FrameGen Channel Status Register
BASEADDR + 0x0078	FgChStatClr	FrameGen Channel Status Clear Register
BASEADDR + 0x007C	FgSkewMon	FrameGen Skew Monitor Register for Secondary Channel Skew Control
BASEADDR + 0x0080	FgSFifoMin	FrameGen Secondary FIFO Min Fill Register
BASEADDR + 0x0084	FgSFifoMax	FrameGen Secondary FIFO Max Fill Register
BASEADDR + 0x0088	FgSFifoFillClr	FrameGen Secondary FIFO Fill Clear Register
BASEADDR + 0x008C	FgSrEpD	FrameGen Skew Regulation ExtraPolation Debug Register
BASEADDR + 0x0090	FgSrFtD	FrameGen Skew Regulation Frame Total Debug Register

5.7.10.13 Iris-MVL - Dither

Table 5-28: Registers Overview

Base Address(es)	Instance no 0: BASEADDR="00034400"	
Absolute Address	**Register Name**	**Register Description**
BASEADDR + 0x0000	Control	Dither Unit common control.
BASEADDR + 0x0004	DitherControl	Dither Unit processing control.
BASEADDR + 0x0008	Release	Dither Unit release.

5.7.10.14 Iris-MVL - TCon

Table 5-29: Registers Overview

Base Address(es)	Instance no 0: BASEADDR="00034800"	
Absolute Address	**Register Name**	**Register Description**
BASEADDR + 0x0000	SSqCnts	The 64 Sequencer Position Definitions registers define the X/Y scan positions of the sequencers, hold their output value and assign the sequencer to an odd/even field
BASEADDR + 0x0400	SSqCycle	This bitfield sets the sequencer cycle length. The value set here -1 is the number of sequencer cycles
BASEADDR + 0x0404	SWreset	TCON Software Reset - Reset all tcon registers except configuration registers. Detailed description in specification document
BASEADDR + 0x0408	TCON_CTRL	TCON Control register
BASEADDR + 0x040C	RSDSInvCtrl	Controls inversion of output polarity when connected IO cells operate in RSDS mode
BASEADDR + 0x0410	MapBit3_0	Mapping of 24 bit RGB or Timing Generator TSig[5:0] to bit 0 .. 3
BASEADDR + 0x0414	MapBit7_4	Mapping of 24 bit RGB or Timing Generator TSig[5:0] to bit 4 .. 7
BASEADDR + 0x0418	MapBit11_8	Mapping of 24 bit RGB or Timing Generator TSig[5:0] to bit 8 .. 11
BASEADDR + 0x041C	MapBit15_12	Mapping of 24 bit RGB or Timing Generator TSig[5:0] to bit 12 .. 15
BASEADDR + 0x0420	MapBit19_16	Mapping of 24 bit RGB or Timing Generator TSig[5:0] to bit 16 .. 19
BASEADDR + 0x0424	MapBit23_20	Mapping of 24 bit RGB or Timing Generator TSig[5:0] to bit 20 .. 23
BASEADDR + 0x0428	MapBit27_24	Mapping of 24 bit RGB or Timing Generator TSig[5:0] to bit 24 .. 27
BASEADDR + 0x042C	MapBit3_0_Dual	Same as MapBit3_0 for 2nd channel
BASEADDR + 0x0430	MapBit7_4_Dual	Same as MapBit7_4 for 2nd channel
BASEADDR + 0x0434	MapBit11_8_Dual	Same as MapBit11_8 for 2nd channel
BASEADDR + 0x0438	MapBit15_12_Dual	Same as MapBit15_12 for 2nd channel
BASEADDR + 0x043C	MapBit19_16_Dual	Same as MapBit19_16 for 2nd channel
BASEADDR + 0x0440	MapBit23_20_Dual	Same as MapBit23_20 for 2nd channel
BASEADDR + 0x0444	MapBit27_24_Dual	Same as MapBit27_24 for 2nd channel
BASEADDR + 0x0448	SPG0PosOn	Sync pulse generator 0, 'Switch on' position
BASEADDR + 0x044C	SPG0MaskOn	The Sequencer Pulse Generator 0 Mask Enable register is used to mask the enable of SPG 0
BASEADDR + 0x0450	SPG0PosOff	Sync pulse generator 0, 'Switch off' position
BASEADDR + 0x0454	SPG0MaskOff	The Sequencer Pulse Generator 0 Mask Enable register is used to mask the disable of SPG 0
BASEADDR + 0x0458	SPG1PosOn	Sync pulse generator 1, 'Switch on' position

Table 5-29: Registers Overview (Continued)

Base Address(es)	**Instance no 0: BASEADDR="00034800"**	
Absolute Address	**Register Name**	**Register Description**
BASEADDR + 0x045C	SPG1MaskOn	The Sequencer Pulse Generator 1 Mask Enable register is used to mask the enable of SPG 1
BASEADDR + 0x0460	SPG1PosOff	Sync pulse generator 1, 'Switch off' position
BASEADDR + 0x0464	SPG1MaskOff	The Sequencer Pulse Generator 1 Mask Enable register is used to mask the disable of SPG 1
BASEADDR + 0x0468	SPG2PosOn	Sync pulse generator 2, 'Switch on' position
BASEADDR + 0x046C	SPG2MaskOn	The Sequencer Pulse Generator 2 Mask Enable register is used to mask the enable of SPG 2
BASEADDR + 0x0470	SPG2PosOff	Sync pulse generator 2, 'Switch off' position
BASEADDR + 0x0474	SPG2MaskOff	The Sequencer Pulse Generator 2 Mask Enable register is used to mask the disable of SPG 2
BASEADDR + 0x0478	SPG3PosOn	Sync pulse generator 3, 'Switch on' position
BASEADDR + 0x047C	SPG3MaskOn	The Sequencer Pulse Generator 3 Mask Enable register is used to mask the enable of SPG 3
BASEADDR + 0x0480	SPG3PosOff	Sync pulse generator 3, 'Switch off' position
BASEADDR + 0x0484	SPG3MaskOff	The Sequencer Pulse Generator 3 Mask Enable register is used to mask the disable of SPG 3
BASEADDR + 0x0488	SPG4PosOn	Sync pulse generator 4, 'Switch on' position
BASEADDR + 0x048C	SPG4MaskOn	The Sequencer Pulse Generator 4 Mask Enable register is used to mask the enable of SPG 4
BASEADDR + 0x0490	SPG4PosOff	Sync pulse generator 4, 'Switch off' position
BASEADDR + 0x0494	SPG4MaskOff	The Sequencer Pulse Generator 4 Mask Enable register is used to mask the disable of SPG 4
BASEADDR + 0x0498	SPG5PosOn	Sync pulse generator 5, 'Switch on' position
BASEADDR + 0x049C	SPG5MaskOn	The Sequencer Pulse Generator 5 Mask Enable register is used to mask the enable of SPG 5
BASEADDR + 0x04A0	SPG5PosOff	Sync pulse generator 5, 'Switch off' position
BASEADDR + 0x04A4	SPG5MaskOff	The Sequencer Pulse Generator 5 Mask Enable register is used to mask the disable of SPG 5
BASEADDR + 0x04A8	SPG6PosOn	Sync pulse generator 6, 'Switch on' position
BASEADDR + 0x04AC	SPG6MaskOn	The Sequencer Pulse Generator 6 Mask Enable register is used to mask the enable of SPG 6
BASEADDR + 0x04B0	SPG6PosOff	Sync pulse generator 6, 'Switch off' position
BASEADDR + 0x04B4	SPG6MaskOff	The Sequencer Pulse Generator 6 Mask Enable register is used to mask the disable of SPG 6
BASEADDR + 0x04B8	SPG7PosOn	Sync pulse generator 7, 'Switch on' position
BASEADDR + 0x04BC	SPG7MaskOn	The Sequencer Pulse Generator 7 Mask Enable register is used to mask the enable of SPG 7
BASEADDR + 0x04C0	SPG7PosOff	Sync pulse generator 7, 'Switch off' position
BASEADDR + 0x04C4	SPG7MaskOff	The Sequencer Pulse Generator 7 Mask Enable register is used to mask the disable of SPG 7
BASEADDR + 0x04C8	SPG8PosOn	Sync pulse generator 8, 'Switch on' position
BASEADDR + 0x04CC	SPG8MaskOn	The Sequencer Pulse Generator 8 Mask Enable register is used to mask the enable of SPG 8
BASEADDR + 0x04D0	SPG8PosOff	Sync pulse generator 8, 'Switch off' position
BASEADDR + 0x04D4	SPG8MaskOff	The Sequencer Pulse Generator 8 Mask Enable register is used to mask the disable of SPG 8
BASEADDR + 0x04D8	SPG9PosOn	Sync pulse generator 9, 'Switch on' position
BASEADDR + 0x04DC	SPG9MaskOn	The Sequencer Pulse Generator 9 Mask Enable register is used to mask the enable of SPG 9
BASEADDR + 0x04E0	SPG9PosOff	Sync pulse generator 9, 'Switch off' position
BASEADDR + 0x04E4	SPG9MaskOff	The Sequencer Pulse Generator 9 Mask Enable register is used to mask the disable of SPG 9

Table 5-29: Registers Overview (Continued)

Base Address(es)	**Instance no 0: BASEADDR="00034800"**	
Absolute Address	**Register Name**	**Register Description**
BASEADDR + 0x04E8	SPG10PosOn	Sync pulse generator 10, 'Switch on' position
BASEADDR + 0x04EC	SPG10MaskOn	The Sequencer Pulse Generator 10 Mask Enable register is used to mask the enable of SPG 10
BASEADDR + 0x04F0	SPG10PosOff	Sync pulse generator 10, 'Switch off' position
BASEADDR + 0x04F4	SPG10MaskOff	The Sequencer Pulse Generator 10 Mask Enable register is used to mask the disable of SPG 10
BASEADDR + 0x04F8	SPG11PosOn	Sync pulse generator 11, 'Switch on' position
BASEADDR + 0x04FC	SPG11MaskOn	The Sequencer Pulse Generator 11 Mask Enable register is used to mask the enable of SPG 11
BASEADDR + 0x0500	SPG11PosOff	Sync pulse generator 11, 'Switch off' position
BASEADDR + 0x0504	SPG11MaskOff	The Sequencer Pulse Generator 11 Mask Enable register is used to mask the disable of SPG 11
BASEADDR + 0x0508	SMx0Sigs	Selection of input signals of sync mixer
BASEADDR + 0x050C	SMx0FctTable	The sync mixer output is the result of the function table a=s4*2**4+s3*2**3+s2*2**2+s1*2**1+s0*2**0 whereby a is bit number and s result of sync mixer input selection
BASEADDR + 0x0510	SMx1Sigs	Selection of input signals of sync mixer
BASEADDR + 0x0514	SMx1FctTable	The sync mixer output is the result of the function table a=s4*2**4+s3*2**3+s2*2**2+s1*2**1+s0*2**0 whereby a is bit number and s result of sync mixer input selection
BASEADDR + 0x0518	SMx2Sigs	Selection of input signals of sync mixer
BASEADDR + 0x051C	SMx2FctTable	The sync mixer output is the result of the function table a=s4*2**4+s**3*2**3+s2*2**2+s1*2**1+s0*2**0 whereby a is bit number and s result of sync mixer input selection
BASEADDR + 0x0520	SMx3Sigs	Selection of input signals of sync mixer
BASEADDR + 0x0524	SMx3FctTable	The sync mixer output is the result of the function table a=s4*2**4+s3*2**3+s2*2**2+s1*2**1+s0*2**0 whereby a is bit number and s result of sync mixer input selection
BASEADDR + 0x0528	SMx4Sigs	Selection of input signals of sync mixer
BASEADDR + 0x052C	SMx4FctTable	The sync mixer output is the result of the function table a=s4*2**4+s3*2**3+s2*2**2+s1*2**1+s0*2**0 whereby a is bit number and s result of sync mixer input selection
BASEADDR + 0x0530	SMx5Sigs	Selection of input signals of sync mixer
BASEADDR + 0x0534	SMx5FctTable	The sync mixer output is the result of the function table a=s4*2**4+s3*2**3+s2*2**2+s1*2**1+s0*2**0 whereby a is bit number and s result of sync mixer input selection
BASEADDR + 0x0538	SMx6Sigs	Selection of input signals of sync mixer
BASEADDR + 0x053C	SMx6FctTable	The sync mixer output is the result of the function table a=s4*2**4+s3*2**3+s2*2**2+s1*2**1+s0*2**0 whereby a is bit number and s result of sync mixer input selection
BASEADDR + 0x0540	SMx7Sigs	Selection of input signals of sync mixer
BASEADDR + 0x0544	SMx7FctTable	The sync mixer output is the result of the function table a=s4*2**4+s3*2**3+s2*2**2+s1*2**1+s0*2**0 whereby a is bit number and s result of sync mixer input selection
BASEADDR + 0x0548	SMx8Sigs	Selection of input signals of sync mixer
BASEADDR + 0x054C	SMx8FctTable	The sync mixer output is the result of the function table a=s4*2**4+s3*2**3+s2*2**2+s1*2**1+s0*2**0 whereby a is bit number and s result of sync mixer input selection
BASEADDR + 0x0550	SMx9Sigs	Selection of input signals of sync mixer
BASEADDR + 0x0554	SMx9FctTable	The sync mixer output is the result of the function table a=s4*2**4+s3*2**3+s2*2**2+s1*2**1+s0*2**0 whereby a is bit number and s result of sync mixer input selection
BASEADDR + 0x0558	SMx10Sigs	Selection of input signals of sync mixer

Table 5-29: Registers Overview (Continued)

Base Address(es)	**Instance no 0: BASEADDR="00034800"**	
Absolute Address	**Register Name**	**Register Description**
BASEADDR + 0x055C	SMx10FctTable	The sync mixer output is the result of the function table a=s4*2**4+s3*2**3+s2*2**2+s1*2**1+s0*2**0 whereby a is bit number and s result of sync mixer input selection
BASEADDR + 0x0560	SMx11Sigs	Selection of input signals of sync mixer
BASEADDR + 0x0564	SMx11FctTable	The sync mixer output is the result of the function table a=s4*2**4+s3*2**3+s2*2**2+s1*2**1+s0*2**0 whereby a is bit number and s result of sync mixer input selection
BASEADDR + 0x0568	Reserved	Do not modify
BASEADDR + 0x056C	Reserved	Do not modify

5.7.10.15 Iris-MVL - Sig

Table 5-30: Registers Overview

Base Address(es)	**Instance no 0: BASEADDR0="00035000"** **Instance no 1: BASEADDR1="00035400"** **Instance no 2: BASEADDR2="00035800"** **Instance no 3: BASEADDR3="00035C00"**	
Absolute Address	**Register Name**	**Register Description**
BASEADDRx + 0x0000	SigLockUnlock	Register to lock or unlock write access to registers of this unit with lock property
BASEADDRx + 0x0004	SigLockStatus	Lock Status for write access to registers of this unit with lock property
BASEADDRx + 0x0008	SigEnable Ï	Turn on and stop the signature unit.
BASEADDRx + 0x000C	StaticControl Ï	Signature configuration and Static control register
BASEADDRx + 0x0010	ThrSumRed Ï	Threshold on Red channel for Summation Signature
BASEADDRx + 0x0014	ThrSumGreen Ï	Threshold on Green channel for Summation Signature
BASEADDRx + 0x0018	ThrSumBlue Ï	Threshold on Blue channel for Summation Signature
BASEADDRx + 0x001C	ErrorThreshold Ï	Number of tolerated signature violation before activating interrupt and setting Status
BASEADDRx + 0x0020	EvalUpperLeft Ï	UpperLeft coordinate of Evaluation Window
BASEADDRx + 0x0024	EvalLowerRight Ï	LowerRight coordinate of Evaluation Window
BASEADDRx + 0x0028	SkipUpperLeft Ï	UpperLeft coordinate of Skip Window
BASEADDRx + 0x002C	SkipLowerRight Ï	LowerRight coordinate of Skip Window
BASEADDRx + 0x0030	SigCRCRefRed Ï	Reference Signature of Type CRC on channel Red
BASEADDRx + 0x0034	SigCRCRefGreen Ï	Reference Signature of Type CRC on channel Green
BASEADDRx + 0x0038	SigCRCRefBlue Ï	Reference Signature of Type CRC on channel Blue
BASEADDRx + 0x003C	SigSumRefRed Ï	Reference Signature of Type Summation (color summation) on channel Red
BASEADDRx + 0x0040	SigSumRefGreen Ï	Reference Signature of Type Summation (color summation) on channel Green
BASEADDRx + 0x0044	SigSumRefBlue Ï	Reference Signature of Type Summation (color summation) on channel Blue
BASEADDRx + 0x0048	Load_Shadow Ï	trigger to load from shadowed registers
BASEADDRx + 0x004C	SoftwareKick Ï	Kick to start signature generation
BASEADDRx + 0x0050	PanicFlag Ï	Monitor the signature violation. Once this Flag is set, it remains 1'active until it's written with '1'. Pls note, the setting of PanicFlag depends on ErrorThreshold and StaticCon-trol.EnPanic = ENABLE
BASEADDRx + 0x0054	Status Ï	Signature Status (update after every signature generation)
BASEADDRx + 0x0058	SigErrCount Ï	Number of frames with signature errors

Table 5-30: Registers Overview (Continued)

Base Address(es)	**Instance no 0: BASEADDR0="00035000"** **Instance no 1: BASEADDR1="00035400"** **Instance no 2: BASEADDR2="00035800"** **Instance no 3: BASEADDR3="00035C00"**	
Absolute Address	**Register Name**	**Register Description**
BASEADDRx + 0x005C	SigCRCRed Ï	Result of CRC Signature for channel Red (update after every signature generation)
BASEADDRx + 0x0060	SigCRCGreen Ï	Result of CRC Signature for channel Green (update after every signature generation)
BASEADDRx + 0x0064	SigCRCBlue Ï	Result of CRC Signature for channel Blue (update after every signature generation)
BASEADDRx + 0x0068	SigSumRed Ï	Result of Summation Signature for channel Red (update after every signature generation)
BASEADDRx + 0x006C	SigSumGreen Ï	Result of Summation Signature for channel Green (update after every signature generation)
BASEADDRx + 0x0070	SigSumBlue Ï	Result of Summation Signature for channel Blue (update after every signature generation)

This page intentionally left blank

Chapter 6: Peripherals

6.1 General

Figure 6-1 shows an overview of all peripherals.

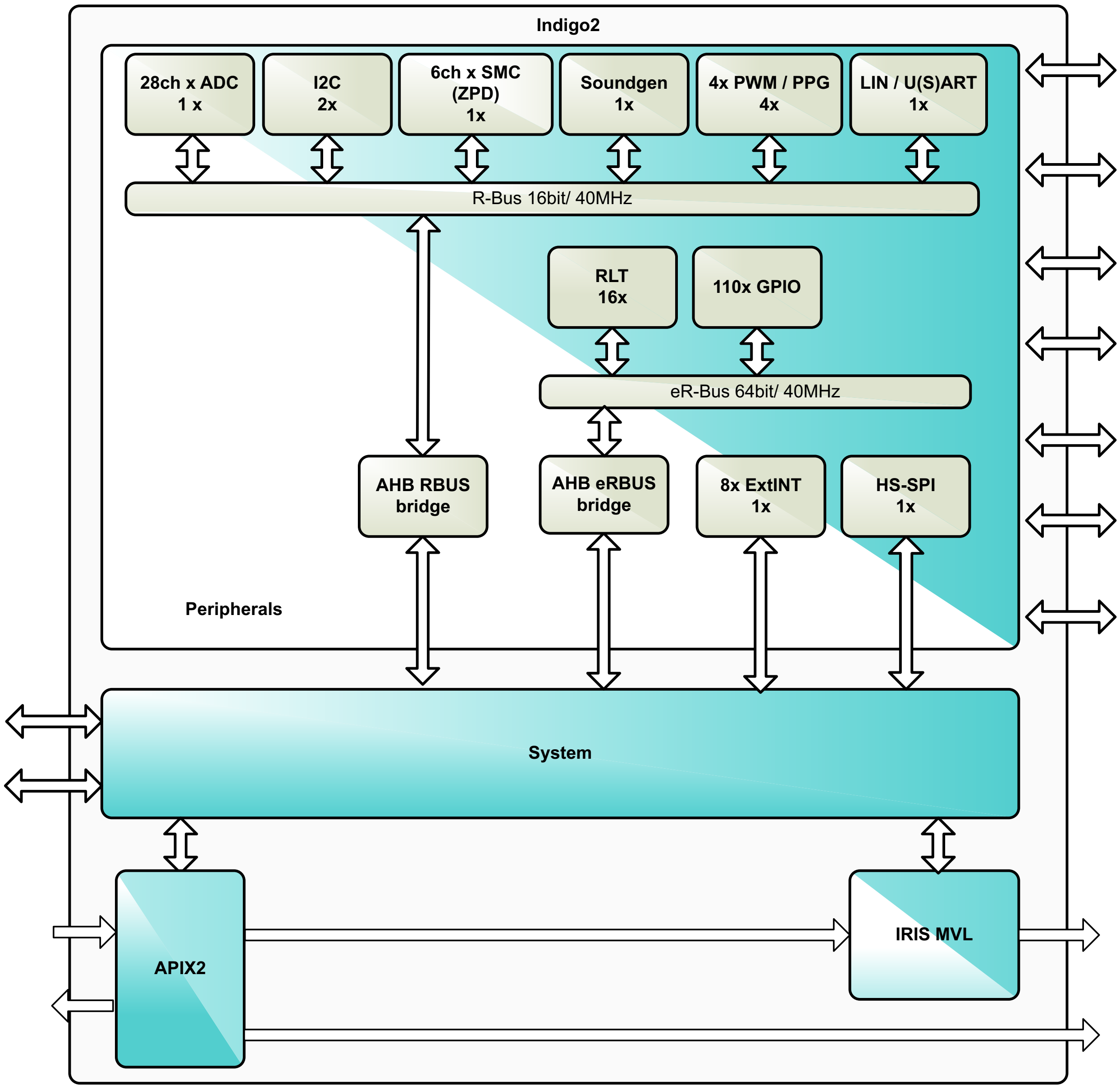


Figure 6-1: Peripherals block diagram

6.2 Stepper Motor Controller

The Stepper Motor Controller is comprised of PWM trigger generators, PWM pulse generators, motor drivers and selector logic circuits.

The four motor drivers have a high-output driving capability and the two motor coils can be connected directly to four pins. The motor rotation is designed to be controlled by a combination of PWM pulse generators and selector logic circuits. The synchronization mechanism enables the synchronous operation of the two PWM pulse generators in order to operate safely.

WARNING: The stepper motor controller can only be used when the HVDD voltage is set to 5V.

6.2.1 Features

- High-output driving capability
- Supports a special algorithm developed by Fujitsu Semiconductor for software based Zero Point Detection.
- Supported by ConfigFIFO
- Constant High-low out

6.2.2 Block Diagram of the Stepping Motor Controller

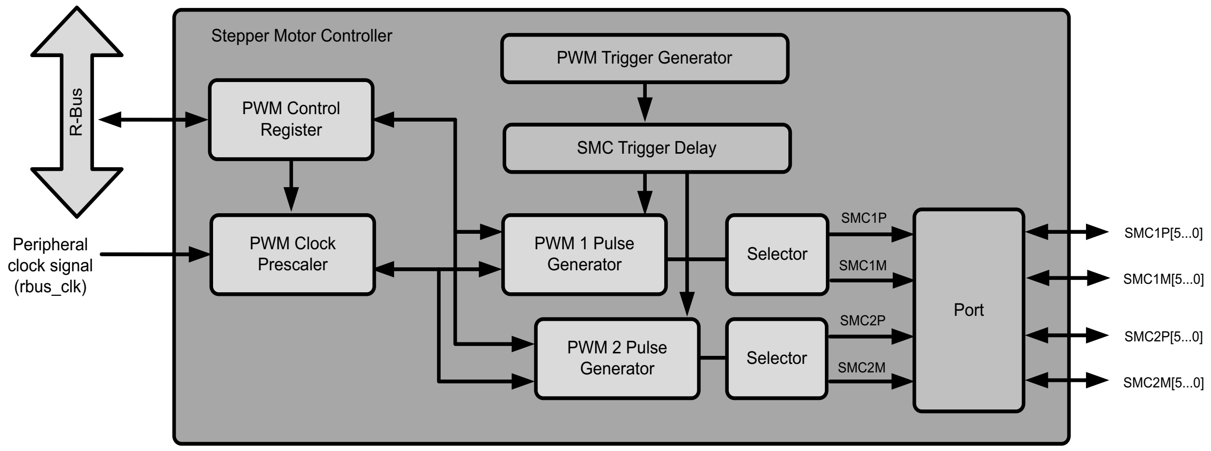


Figure 6-2: Block Diagram of Stepping Motor Controller

6.2.3 Operation of Stepping Motor Controller

The output PWM signal generation depends on the setting of Stepping Motor Controller.

- Setting Operation of Stepping Motor Controller

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SMCn_PWC									–	P2	P1	P0	CE	SC	–	–
									X	U	U	U	U	U	X	X
SMCn_PWCS									–	–	–	–	CES	–	–	–
									X	X	X	X	1	X	X	X
SMCn_PWCC									–	–	–	–	CEC	–	–	–
									X	X	X	X	0	X	X	X
SMCn_PWC1	–	–	–	–	–	–	PWM1 H width (compare value) is set									
	X	X	X	X	X	X										
SMCn_PWC2	–	–	–	–	–	–	PWM2 H width (compare value) is set									
	X	X	X	X	X	X										
SMCn_PWS	–	BS	P22	P21	P20	M22	M21	M20	–	–	P12	P11	P10	M12	M11	M10
	X	U	U	U	U	U	U	U	X	X	U	U	U	U	U	U
SMCn_PWSS	–	BSS	–	–	–	–	–	–	–	–	–	–	–	–	–	–
	X	1	X	X	X	X	X	X	X	X	X	X	X	X	X	X

U: Used bit, X: Not used bit, 1: 1is set, 0: 0 is set, n: Channel no.

Figure 6-3: Setting of Stepping Motor Controller (1)

SMCn_PTRGDL									D[7]	D[6]	D[5]	D[4]	D[3]	D[2]	D[1]	D[0]
									U	U	U	U	U	U	U	U
SMCTGg_PTRGS	–	–	S25	S24	S23	S22	S21	S20	–	–	S15	S14	S13	S12	S11	S10
	X	X	U	U	U	U	U	U	X	X	U	U	U	U	U	U
SMCTGg_PTRG	–	–	–	–	–	–	–	–	–	–	–	–	–	–	TR2	TR1
	X	X	X	X	X	X	X	X	X	X	X	X	X	X	U	U

U:Used bit, X:Not used bit, 1:1 is set, 0:0 is set, n:Channel no

Figure 6-4: Setting of Stepping Motor Controller (2)

6.2.4 Operation of PWM-pulse Generator

■ Selection of Motor Drive Signals

Motor drive signals, that are output to each pin related to the Stepping Motor Controller, can be selected from four types of signals for each pin by setting the PWM Selection Register. Table 6-1 lists the selection of the motor drive signals and the settings of PWM Selection Register.

Table 6-1: Motor drive signals selection and PWM Selection Register setting

P12, P11, P10 Bits P22, P21, P20 Bits	**SMC1P Output SMC2P Output**	**M12, M11, M10 Bits M22, M21, M20 Bits**	**SMC1M Output SMC2M Output**
000_B	L	000_B	L
001_B	H	001_B	H
$01X_B$	PWM Pulse	$01X_B$	PWM Pulse
$1XX_B$	High impedance	$1XX_B$	High impedance
X= don’t care			

■ PWM-pulse Generator

When the counter starts (SMCn_PWC: CE = 1), it increments from 00_H in step of 1 on the leading edge of the selected count clock rising. The PWM output pulse wave remains "H" until the value of the counter matches the value set in the PWM Compare Register. It then changes to "L" and remains "L" until the value of the counter overflows (for 8-bit operation:$FF_H \rightarrow 00_H$ and for 10-bit operation:$3FF_H \rightarrow 000_H$). Figure 6-5 shows the PWM waveforms generated by the PWM-pulse generator.

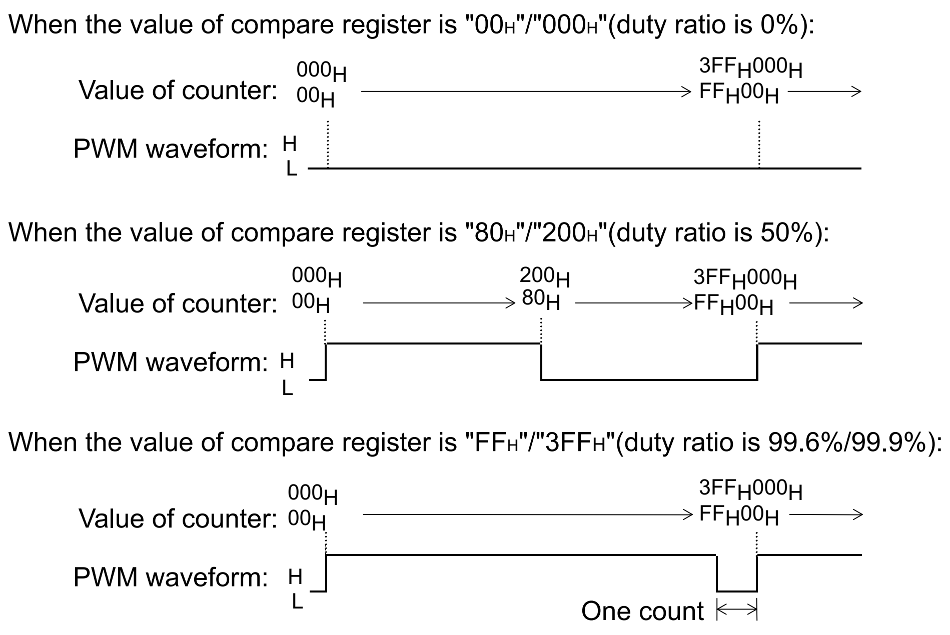

When the value of compare register is "00H"/"000H"(duty ratio is 0%):
Value of counter:
000H
00H
3FFH000H
FFH00H
PWM waveform:
H
L
When the value of compare register is "80H"/"200H"(duty ratio is 50%):
Value of counter:
000H
00H
200H
80H
3FFH000H
FFH00H
PWM waveform:
H
L
When the value of compare register is "FFH"/"3FFH"(duty ratio is 99.6%/99.9%):
Value of counter:
000H
00H
3FFH000H
FFH00H
PWM waveform:
H
L
One count

Figure 6-5: Examples of PWM1 and PWM2 output waveforms

■ Usage of BS-bit

When the PWM Selection Register (SMCn_PWS) is set and "1" is written to the SMCn_PWS:BS bit, the setting of the PWM Selection Register (SMCn_PWS) is enabled at the end of the current PWM cycle. The BS-bit is cleared automatically at the beginning of the next PWM cycle. When "1" is written to the BS-bit, and the BS-bit is cleared simultaneously at the beginning of the next PWM cycle, "1" is written to the BS-bit and the BS-bit clearing is cancelled.

Figure 6-6 shows the load timing of PWM.

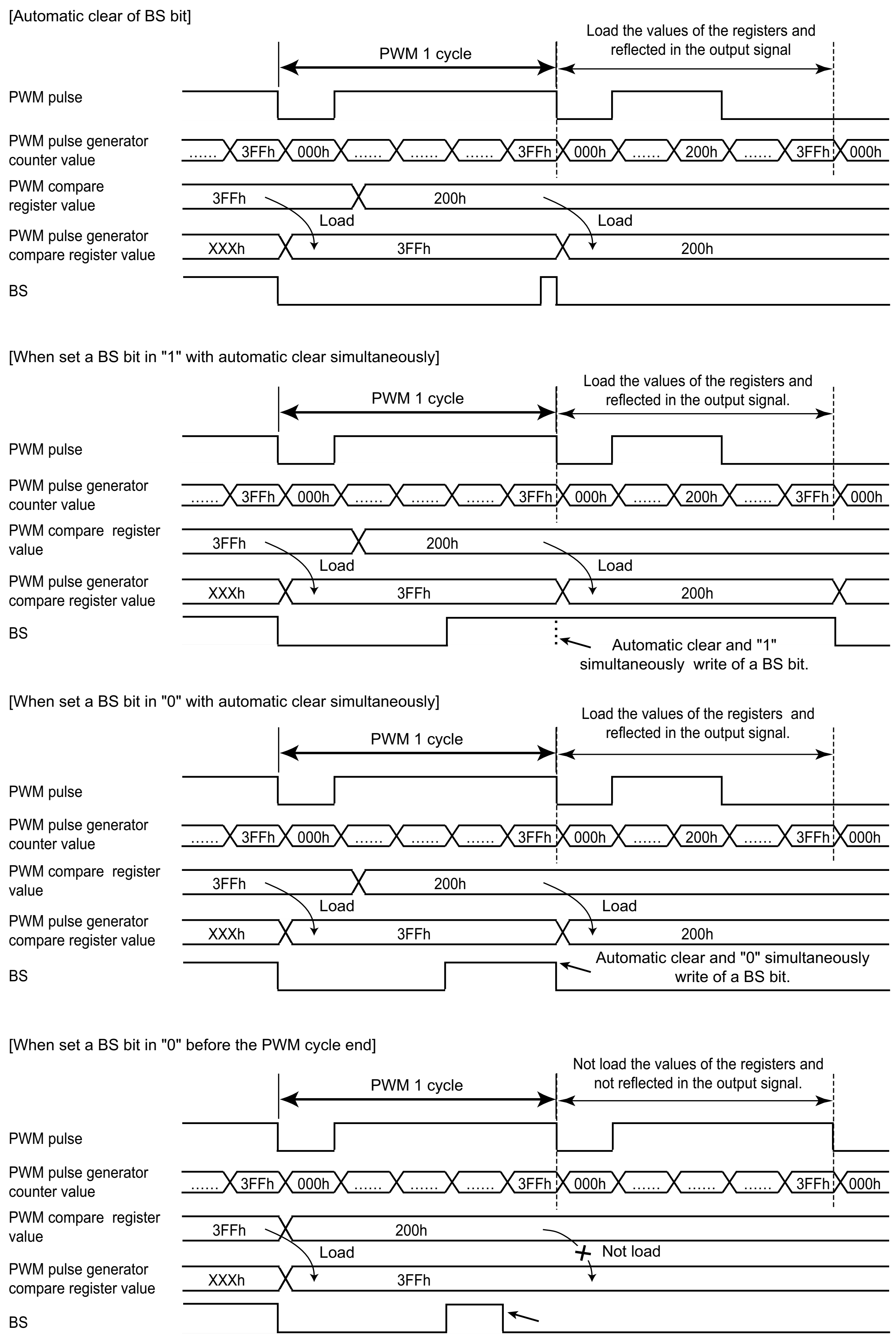

[Automatic clear of BS bit]
PWM 1 cycle
Load the values of the registers and reflected in the output signal
PWM pulse
PWM pulse generator counter value
...... 3FFh 000h .... 3FFh 000h ...... 200h 3FFh 000h
PWM compare register value
3FFh 200h
Load
Load
PWM pulse generator compare register value
XXXh 3FFh 200h
BS
[When set a BS bit in "1" with automatic clear simultaneously]
PWM 1 cycle
Load the values of the registers and reflected in the output signal.
PWM pulse
PWM pulse generator counter value
...... 3FFh 000h .... 3FFh 000h ...... 200h 3FFh 000h
PWM compare register value
3FFh 200h
Load
Load
PWM pulse generator compare register value
XXXh 3FFh 200h
BS
Automatic clear and "1" simultaneously write of a BS bit.
[When set a BS bit in "0" with automatic clear simultaneously]
PWM 1 cycle
Load the values of the registers and reflected in the output signal.
PWM pulse
PWM pulse generator counter value
...... 3FFh 000h .... 3FFh 000h ...... 200h 3FFh 000h
PWM compare register value
3FFh 200h
Load
Load
PWM pulse generator compare register value
XXXh 3FFh 200h
BS
Automatic clear and "0" simultaneously write of a BS bit.
[When set a BS bit in "0" before the PWM cycle end]
PWM 1 cycle
Not load the values of the registers and not reflected in the output signal.
PWM pulse
PWM pulse generator counter value
...... 3FFh 000h .... 3FFh 000h 3FFh 000h
PWM compare register value
3FFh 200h
Load
Not load
PWM pulse generator compare register value
XXXh 3FFh
BS

Figure 6-6: Load timing of PWM compare register value

6.2.5 Operation of PWM-Trigger Generator

- Trigger to start the PWM-pulse generation of one group of SMCs

The Stepping Motor Controller can generate trigger signals to start PWM-pulse generation for one group of SMC-channels, once SMC-Trigger Select Register (SMCTGg_PTRGS), SMC-Trigger Register (SMCTGg_PTRG), and SMC-Trigger Delay Register (SMCn_PTRGDL) are configured.

SMC-Trigger is used to start one group of SMC-channels, however PWM output of different SMCs of that group starts after its programmable delay configured in SMCn_PTRGDL. The trigger input to SMCs sets the SMCn_PWC:CE bit of the selected SMCs (same as the SW writes to "1" to PWCSn:CES) after the programmable delay configured in SMCn_PTRGDL. The SMCn_PWC:CE bit can be cleared by writing ’1’ to PWCCn:CEC.

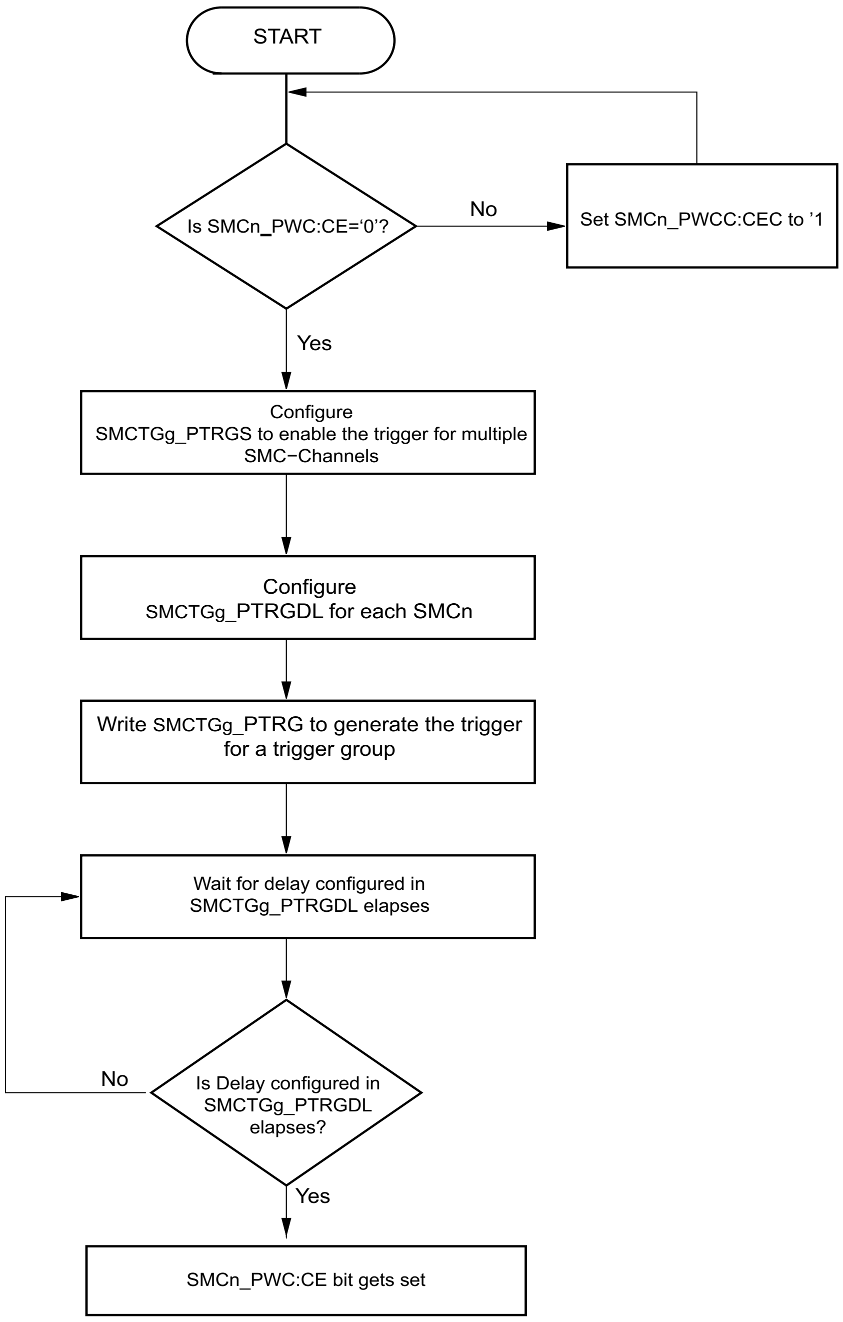

START
Is SMCn_PWC:CE=‘0’?
No
Set SMCn_PWCC:CEC to ’1
Yes
Configure
SMCTGg_PTRGS to enable the trigger for multiple SMC−Channels
Configure
SMCTGg_PTRGDL for each SMCn
Write SMCTGg_PTRG to generate the trigger for a trigger group
Wait for delay configured in SMCTGg_PTRGDL elapses
No
Is Delay configured in SMCTGg_PTRGDL elapses?
Yes
SMCn_PWC:CE bit gets set

Figure 6-7: Flowchart for triggering multiple SMC Channels

Figure 6-7 shows how an application reads SMCn_PWC:CE bit to determine whether PWM is running or not. If the SMC is generating PWM pulses, then a ‘1’ is written to the SMCn_PWCC:CEC bit of each intended SMC to stop the PWM pulse generation.

In Figure 6-8, the register configuration to trigger SMC0, SMC1 and not to trigger SMC2 1st group of SMC‘s is:

- PWM-Trigger Registers: SMCTGg_PTRGS=0x0003 and SMCTGg_PTRG=0x01 (to start triggering for 1st group of SMC‘s).
- SMC0 Registers: SMCn_PTRGDL0=0x00.
- SMC1 Registers: SMCn_PTRGDL1=0x00.
- SMC2 Registers: SMCn_PTRGDL2=0x00.

:

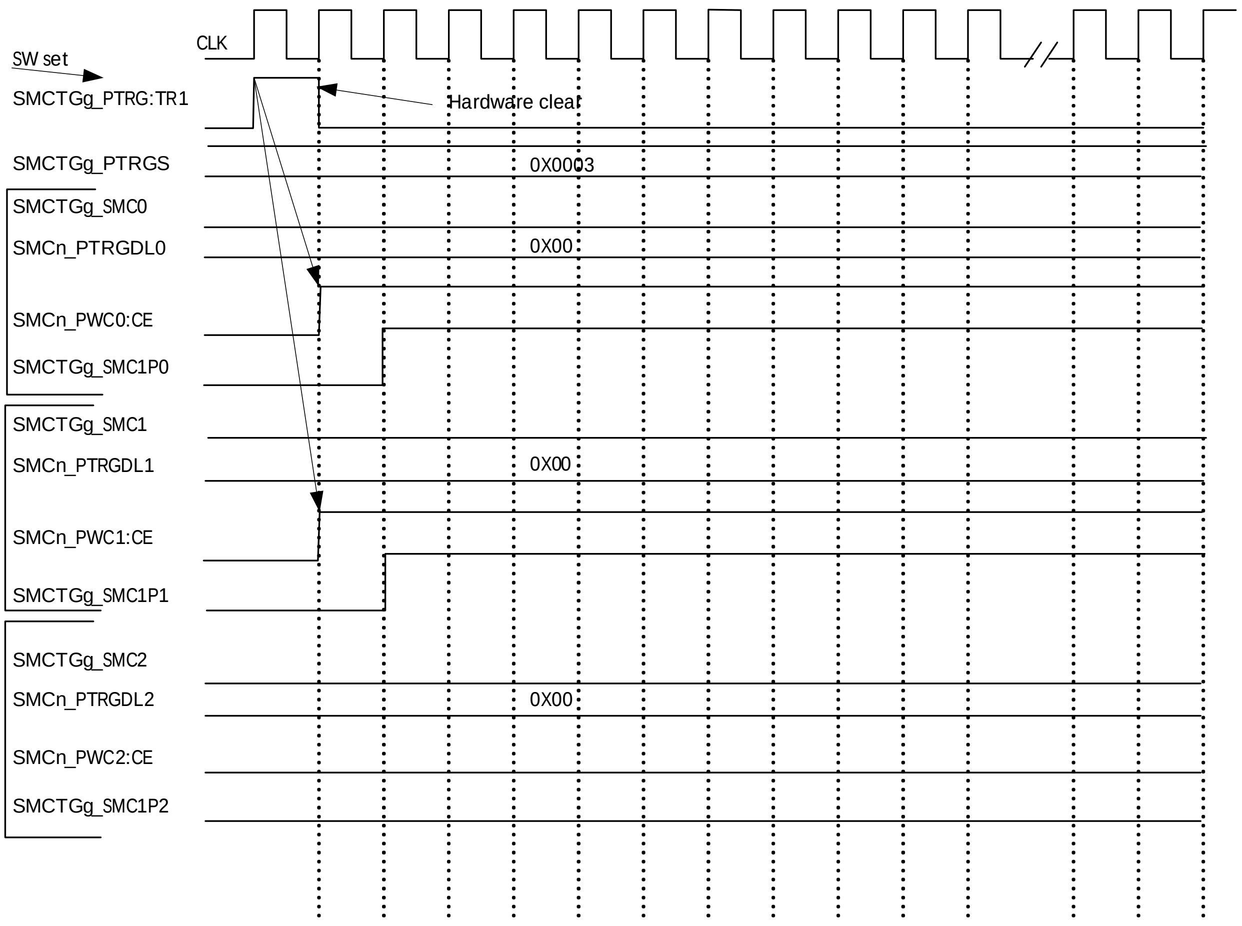


Figure 6-8: Timing diagram of simultaneous triggering of SMCs

Figure 6-8 shows that SMC1P0 and SMC1P1, 1st group of SMC‘s, starts simultaneously, while SMC1P2 is not triggered because the corresponding bit for SMC2 in SMCTGg_PTRGS register is set to ’0’. The SMCn_PWC:CE and SMCn_PWC1:CE bit is set by the trigger, however SMCn_PWC2:CE bit is not set.

6.2.6 Shadow Register Setup

There is an alternative method to access the PWC1_D, PWC2_D, PWS_M1, PWS_M2, PWS_P1, PWS_P2 and PWS_BS register fields of all six stepper motor controllers when using the MB88F33x ‘Indigo2(-x)’. This method sorts the register fields to make it possible to allow the ConfigFIFO to access the stepper motor controllers with little bus overhead cost. Two additional registers (SHD1_SMCx and SHD2_SMCx) exist for each stepper motor controller. The SHD1_SMCx registers contain PWC1_D, PWS_P1 and PWS_M1 bitfields and the SHD2_SMCx registers contain PWC2_D, PWS_P2 and PWS_M2 bitfields. When writing to the SHD2_SMCx registers, the PWS_BS bits are automatically set to 1. The additional registers for all of the stepper motor controllers are located in an ascending address space.

6.2.7 Notes on Using Stepping Motor Controller

Steps to pay attention when changing the PWM settings and writing to PWM-Trigger Registers are described below.

- Cautions when changing the PWM Setting

The PWM Compare Registers 1 and 2 (SMCn_PWC1, SMCn_PWC2) and the PWM Selection Register (SMCn_PWS) can always be accessed. However, to change the setting of "H" width of PWM or to change the PWM output, "1" must be written to the SMCn_PWS:BS bit after (or and at the same time) a setting is written to those registers (the PWM Compare Register 1 and 2 and the PWM Selection Register).

When "1" is set to the SMCn_PWS:BS bit, the new setting is enabled at the end of the current PWM cycle and the SMCn_PWS:BS bit is cleared automatically.

When "1" is written to the SMCn_PWS:BS bit and the SMCn_PWS:BS bit is reset at the end of the PWM cycle simultaneously, "1" is written to the SMCn_PWS:BS and resetting of the SMCn_PWS:BS bit is cancelled.

- Writing to PWM-Trigger Registers (SMCTGg_PTRG, SMCTGg_PTRGS)

To use the PWM Trigger function, set the SMCn_PWC:CE bit of the SMC to be triggered by the trigger function to ’0’. If SMCn_PWC:CE bit is already set to ’1’ when PWM start is triggered, the PWM output is not affected.

6.2.8 Zero Point Detection

Indigo2 supports a special algorithm developed by Fujitsu Semiconductor for software based Zero Point Detection. This Zero Point Detection can be done together with the ADC.

The zero point detection is only possible when the supplies HVDD, VDP5, and AVCC are set to 5V.

6.2.9 Stepper Motor Controller Additional Register Information

6.2.9.1 PWM Control Register (SMCn_PWC)

Table 6-2: Settings of P2, P1, P0 bits to generate clock for PWM pulse generator

P2	P1	P0	Clock input	PWM cycle (at rbus_clk = 40 MHz)	
				SC=0	SC=1
0	0	0	CLK	6.4 us	25.6 us
0	0	1	CLK/4	25.6 us	102.4 us
0	1	0	CLK/5	32 us	128 us
0	1	1	CLK/6	38.4 us	153.6 us
1	0	0	CLK/8	51.2us	204.8 us
1	0	1	CLK/10	64us	256 us
1	1	0	CLK/12	76.8us	307.2us
1	1	1	CLK/16	102.4 us	409.6 us

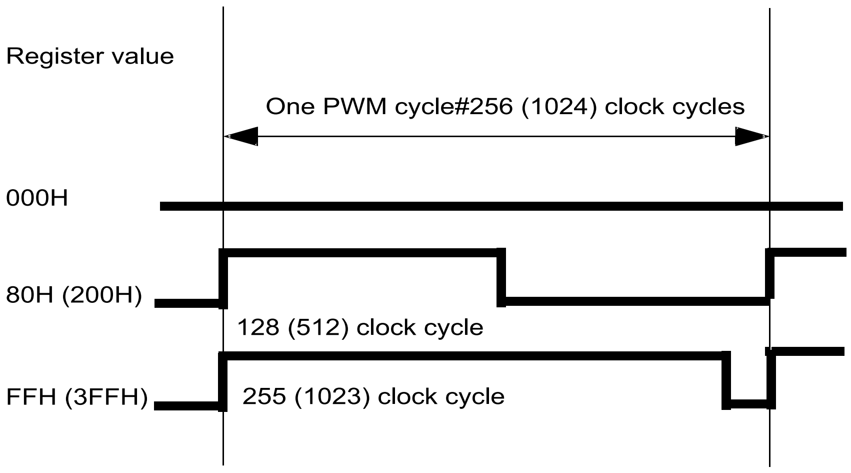


Figure 6-9: Relationship between the Compare register setting value and PWM pulse width

Table 6-3: Relationship between the output levels and the select bits

Pm2	Pm1	Pm0	SMCmPn	Mm2	Mm1	Mm0	SMCmMn
0	0	0	L	0	0	0	L
0	0	1	H	0	0	1	H
0	1	X	PWM Pulse	0	1	X	PWM Pulse
1	X	X	High impedance	1	X	X	High impedance

NOTEm = 1 to 2 (motor coils), n = 0 to 5 (stepper motor channels).
SMCn_PWS register can be read or written with 8-bit and 16-bit access.

The procedure to update SMCn_PWC1, SMCn_PWC2 and SMCn_PWS Registers:

1. Update PWM Compare Registers (SMCn_PWC1, SMCn_PWC2).
2. Configure PWM Selection Register with BS =’1’, or Set BS via SMCn_PWSS:BSS.

6.2.10 Stepper Motor Controller Core Register Overview

Table 6-4: Registers Overview

Base Address(es)	**Instance no 0: BASEADDR0="00080000"** **Instance no 1: BASEADDR1="00080400"** **Instance no 2: BASEADDR2="00080800"** **Instance no 3: BASEADDR3="00080C00"** **Instance no 4: BASEADDR4="00081000"** **Instance no 5: BASEADDR5="00081400"**	
Absolute Address	**Register Name**	**Register Description**
BASEADDRx + 0x0000	SMCn_PWC	PWM Control Register
BASEADDRx + 0x0001	Reserved	Do not modify
BASEADDRx + 0x0002	SMCn_PWCS	PWM Control Set Register
BASEADDRx + 0x0003	Reserved	Do not modify
BASEADDRx + 0x0004	SMCn_PWCC	PWM Control Clear Register
BASEADDRx + 0x0005	Reserved	Do not modify
BASEADDRx + 0x0006	SMCn_PWC1	PWM1 Compare Registers
BASEADDRx + 0x0008	SMCn_PWC2	PWM2 Compare Registers
BASEADDRx + 0x000A	SMCn_PWS	PWM Selection Register
BASEADDRx + 0x000C	SMCn_PWSS	PWM Selection Set Register
BASEADDRx + 0x000E	SMCn_PTRGDL	SMC-Trigger Delay Register
BASEADDRx + 0x000F	Reserved	Do not modify
BASEADDRx + 0x0010	Reserved	Do not modify
BASEADDRx + 0x0011	Reserved	Do not modify

6.2.11 Stepper Motor Controller Trigger Register Overview

Table 6-5: Registers Overview

Base Address(es)	Instance no 0: BASEADDR="00081C00"	
Absolute Address	**Register Name**	**Register Description**
BASEADDR + 0x0000	SMCTGg_PTRGS	SMC-Trigger Select Register
BASEADDR + 0x0002	SMCTGg_PTRG	SMC-Trigger Register
BASEADDR + 0x0003	Reserved	Do not modify
BASEADDR + 0x0004	SMCTGg_SHD1_SMC0	Shadow register access for Coil1 of SMC0
BASEADDR + 0x0006	SMCTGg_SHD2_SMC0	Shadow register access for Coil2 of SMC0
BASEADDR + 0x0008	SMCTGg_SHD1_SMC1	Shadow register access for Coil1 of SMC1
BASEADDR + 0x000A	SMCTGg_SHD2_SMC1	Shadow register access for Coil2 of SMC1
BASEADDR + 0x000C	SMCTGg_SHD1_SMC2	Shadow register access for Coil1 of SMC2
BASEADDR + 0x000E	SMCTGg_SHD2_SMC2	Shadow register access for Coil2 of SMC2
BASEADDR + 0x0010	SMCTGg_SHD1_SMC3	Shadow register access for Coil1 of SMC3
BASEADDR + 0x0012	SMCTGg_SHD2_SMC3	Shadow register access for Coil2 of SMC3
BASEADDR + 0x0014	SMCTGg_SHD1_SMC4	Shadow register access for Coil1 of SMC4
BASEADDR + 0x0016	SMCTGg_SHD2_SMC4	Shadow register access for Coil2 of SMC4
BASEADDR + 0x0018	SMCTGg_SHD1_SMC5	Shadow register access for Coil1 of SMC5
BASEADDR + 0x001A	SMCTGg_SHD2_SMC5	Shadow register access for Coil2 of SMC5

6.3 Analog Digital Converter (ADC)

The A/D Converter converts analog input voltages into digital values. The A/D Converter features 28 separate result data registers, four range comparators and three interrupt flags.

WARNING: The ADC inputs which are on pins in the HVDD voltage domain (input number 28 - 16) can only be used when the supplies HVDD, VDP5 and AVCC are equal. Using them when this condition is not fulfilled can destroy the Indigo2.

6.3.1 Features of the A/D Converter

- Conversion time: Minimum 1µs per channel.
- RC type successive approximation conversion with sample & hold circuit.
- 10-bit or 8-bit resolution.
- Programmable analog input selection from 28 channels (16 dedicated channels, 12 channels used for SMC zero point detection).
- One common result data register and 28 dedicated channel result data registers.
- Scan conversion mode, continuous conversion of multiple channels, programmable for up to 28 channels.
  - Continuous mode, repeatedly converts the specified channels.
  - Single mode, converts the specified channel(s) only once.
  - Stop mode, converts one channel, then temporarily halts until the next activation (enables synchronization of the conversion start timing).
- Interrupt request generation for:
  - End of conversion interrupt (single channel).
  - End of scan interrupt (all enabled channels), results are stored in dedicated result registers.
- A/D conversion can generate a DMA transfer request to transfer the results of A/D conversion to memory.
- Four range comparator channels, comparing the upper 8 bit of the conversion result.
- Programmable upper and lower thresholds, individually for each comparator.
- Any of the available ADC channel can be assigned to one of the four range comparators.
- The comparison results will set flags per ADC channel, depending on the configuration. Possible configurations:
  - ‘Outside range’: The flags are set if the A/D result is below the lower OR above the upper threshold.
  - ‘Inside range’: The flags are set if the A/D result is above the lower AND below the upper threshold.
- The configuration can be set individually per ADC channel.
- Range comparison triggers an A/D range comparator interrupt request.
- Result of range comparator can be filtered to ignore multiple spikes.

6.3.2 Block Diagram

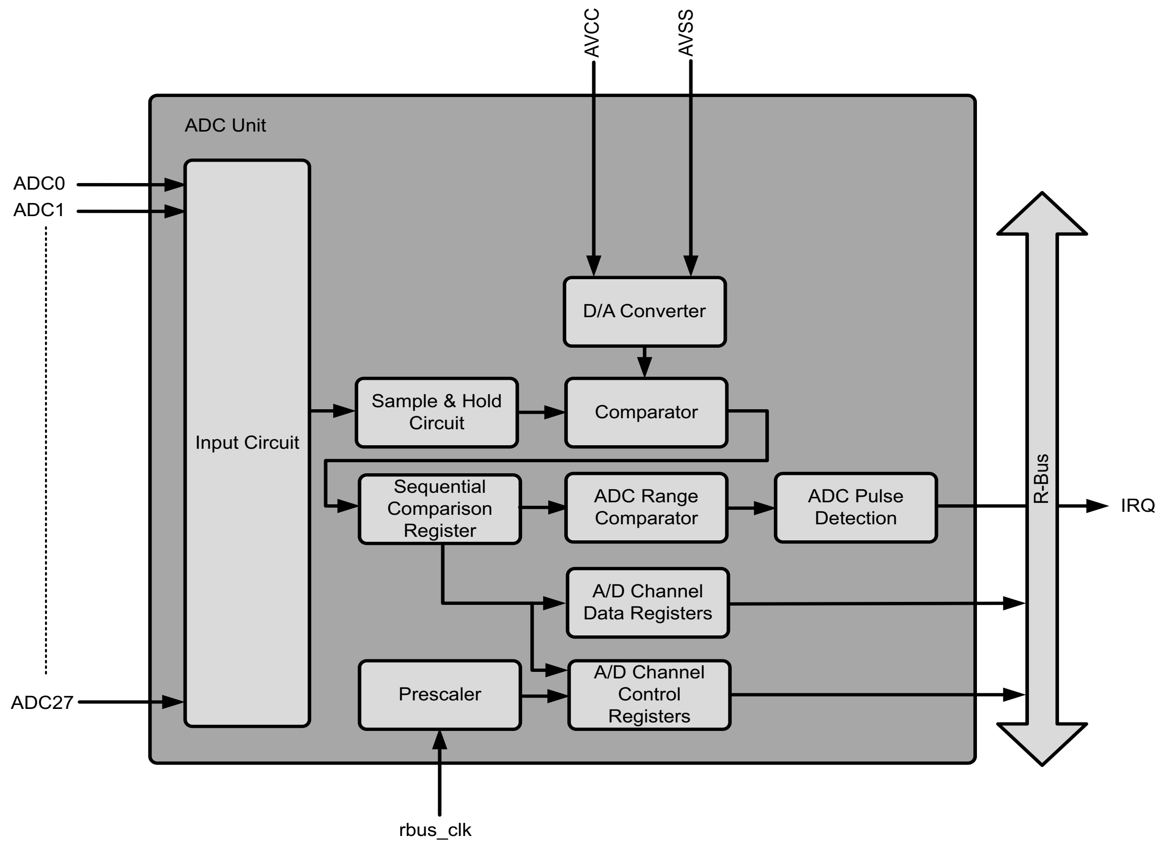


Figure 6-10: Block diagram of A/D converter

6.3.3 Operation of the A/D Converter

The A/D Converter operates using a successive approximation method with a resolution of either 10-bits or 8-bits. There is one 16-bit register provided to store conversion results (ADCn_CR), which is updated each time conversion completes. Additionally, there is one ADC Channel Data Register per channel (ADCn_CD27~0), which is updated each time the assigned channel is converted.

The MB88F334 ’Indigo-2’ supports 28 ADC. Channels 0 to 15 are used for ADC input and channels 16 to 27 are used for the Stepper Motor Zero Point Detection. Channels 16 - 27 need to be enabled with register IFC_CTRL.zpd_adc_en. The Channel Data registers especially improve the continuous conversion mode.

The range comparator compares the converted values with the configured values in the threshold registers. It then accordingly generates an interrupt for 'inside range' or 'outside range' depending on the configuration of ADCn_RCOIRS32, ADCn_RCOIRS10 registers. There are four range comparators present, where any of the range comparators can be configured for any of the available ADC analog input channels. The ADC pulse detection function detects events of desired length and also filters parasitic inverted events. Each ADC channel has a dedicated pulse detection module.

The following section describes the operation of the A/D Converter.

6.3.3.1 Single Mode

In single conversion mode, the analog input signals which are enabled via ADCn_ER32 and ADCn_ER10 are selected and converted in order. Upon assertion of the start signal selected by the ADCn_CS1:STS[1:0] bits, the conversion starts from the start channel which is configured by ADCn_SCH:ANS bits and stops when the conversion of the end channel configured by ADCn_ECH:ANE is over. The analog input signals which are disabled via ADCn_ER32 and ER10 are not converted in the range set between ADCn_SCH:ANS and ADCn_ECH:ANE. If the start channel and end channel are the same (ADCn_SCH:ANS = ADCn_ECH:ANE), only a single channel conversion is performed. For the ADC to start conversion, the channel selected by ADCn_ANS[4:0] bits (start channel) must be enabled by the corresponding ADCn_ER32 nad ADCn_ER10:ADE bit.

Examples:

ADCn_ER32:ADE[0:3] = '1111'

ADCn_SCH:ANS = '00000', ADCn_ECH:ANE = '00011'
Start → AN0 → AN1 → AN2 → AN3 → end

ADCn_SCH:ANS = '00010', ADCn_ECH:ANE = '00010'
Start → AN2 → end

ADCn_ER32:ADE[0:3] = '1011'

ADCn_SCH:ANS = '00000', ADCn_ECH:ANE = '00011'
Start → AN0 → AN2 → AN3 → end

ADCn_ER32:ADE[2] = '1'

ADCn_SCH:ANS = '00010', ADCn_ECH:ANE = '00010'
Start → AN2 → end

NOTE In single mode, for the conversion of one scan sequence, a single trigger is required which is determined by the ADCn_CS1:STS[1:0] bits. This mode enables synchronization of the conversion start signal.

6.3.3.2 Continuous Mode

In continuous mode, the analog input signals which are enabled via ADCn_ER32 and ADCn_ER10 are selected and converted in order. Upon assertion of the start signal selected by the ADCn_CS1:STS[1:0] bits, the conversion starts from the start channel which is configured by ADCn_SCH:ANS and stops when the conversion of the end channel configured by ADCn_ECH:ANE is finished. After this the converter returns to the ADCn_SCH:ANS channel and repeats the process continuously. When the start and end channels are the same (ADCn_SCH:ANS = ADCn_ECH:ANE), conversion is performed continuously for that channel. For the ADC to start conversion, the channel selected by ADCn_SCH:ANS[4:0] bits (start channel) must be enabled by the corresponding ADCn_ER32 and ADCn_ER10:ADE bit.

Examples:

ADCn_ER32:ADE[0:3] = '1111'

ADCn_SCH:ANS = '00000', ADCn_ECH:ANE = '00011'
Start → AN0 → AN1 → AN2 → AN3 → AN0... → repeat

ADCn_SCH:ANS = '00010', ADCn_ECH:ANE = '00010'
Start → AN2 → AN2 → AN2... →repeat

ADCn_ER32:ADE[0:3] = '1011'

ADCn_SCH:ANS = ' 00000', ADCn_ECH:ANE = '00011'
Start → AN0 → AN2 → AN3 → AN0... → repeat

ADCn_ER32:ADE[2] = '1'

ADCn_SCH:ANS = '00010', ADCn_ECH:ANE = '00010'
Start → AN2 → AN2 → AN2... → repeat

In continuous mode the conversion is repeated until '1' is written to the ADCn_CSC1:BUSYC bit which clears ADCn_CS1:BUSY (clearing the BUSY bit forcibly stops the conversion operation).

NOTE

1. Forcibly terminating operation halts the current conversion during mid-conversion (if operation is forcibly terminated, the value in the conversion register is the result of the most recently completed conversion).
2. In continuous mode, for conversion of one scan sequence a single trigger is required which is determined by the ADCn_CS1:STS[1:0] bits. This mode enables synchronization of the conversion start signal.

6.3.3.3 Stop Mode

In stop mode, upon assertion of the start signal selected by the ADCn_CS1:STS[1:0] bits, the analog input channels selected by the ADCn_SCH:ANS bits and ADCn_ECH:ANE bits are converted in order, but conversion operation pauses after each channel. The pause is released by applying another start signal.

After the conversion on the end channel determined by the ADCn_ECH:ANE bits, the converter returns to the ADCn_SCH:ANS channel and repeats the conversion process continuously. When the start and end channels are the same (ADCn_SCH:ANS = ADCn_ECH:ANE), only a single channel conversion is performed.

In stop mode, each conversion is triggered by the trigger source determined by the ADCn_CS1:STS[1:0] bits. This mode enables synchronization of the conversion start signal.

Examples:

ADCn_SCH:ANS = '00000', ADCn_ECH:ANE = '00011'
Start → AN0 → stop → start → AN1 → stop → start → AN2 → stop → start → AN3 → stop → start → AN0... → repeat

ADCn_SCH:ANS = '00010', ADCn_ECH:ANE = '00010'
Start → AN2 → stop → start → AN2 → stop → start → AN2... → repeat

The channels that are enabled by ADCn_ER32, ADCn_ER10:ADE bits are only converted in stop mode. The conversion is never started if the start channel (set by ADCn_SCH:ANS) is set to '0'.

6.3.3.4 Single-shot Conversion

The following figure shows the operation of A/D converter in Single-shot conversion mode:

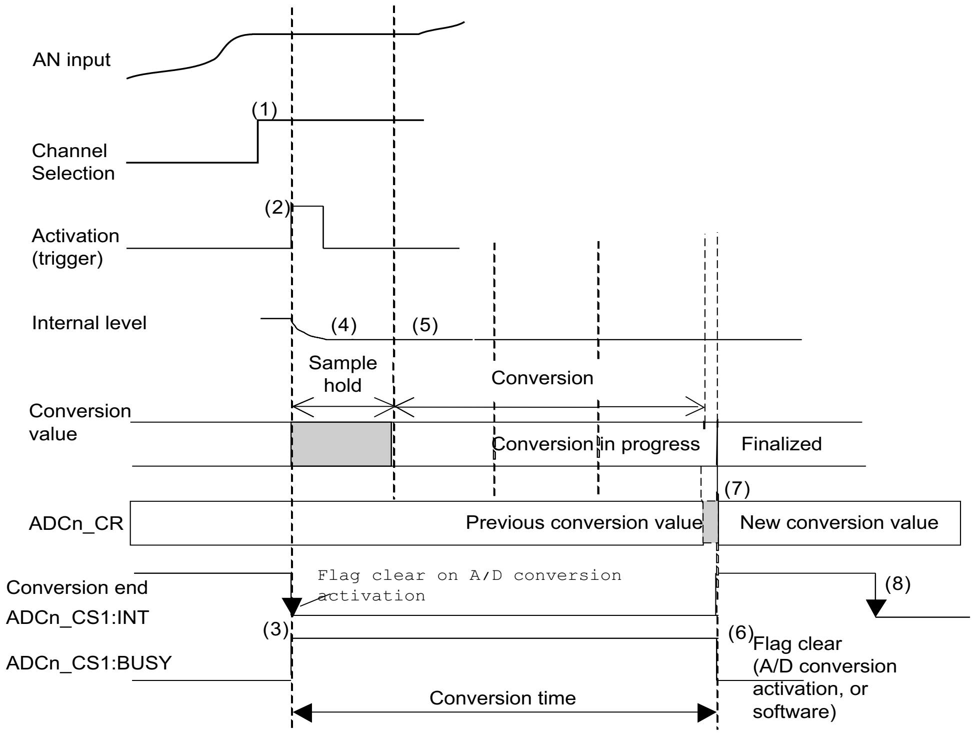


Figure 6-11: Operation of ADC in single-shot conversion mode

(1) Channel selection
(2) A/D conversion activation (Trigger input: Software trigger/Reload timer/External trigger)
(3) ADCn_CS1:INT flag clear, ADCn_CS1:BUSY flag set
(4) Sample hold
(5) Conversion
(6) Conversion end, ADCn_CS1:INT flag set, ADCn_CS1:BUSY flag clear
(7) Store result in ADC Common Data Register (ADCn_CR)
(8) Software-based ADCn_CS1:INT flag clear

NOTE In Figure 6-11 the ADCn_CS1:INT interrupt flag at the beginning (point 3) is shown to be cleared. This clearing is done by the host CPU or Command Sequencer after reading the previous converted value from the common data register ADCn_CR by writing ’1’ to ADCn_CSC1:INTC. Or the interrupt is automatically cleared by reading the ADCn_CR register. This has to be enabled by ADCn_CS2:INTAC

6.3.3.5 Scan Conversion

The following figure shows the operation of A/D converter in Scan conversion mode:

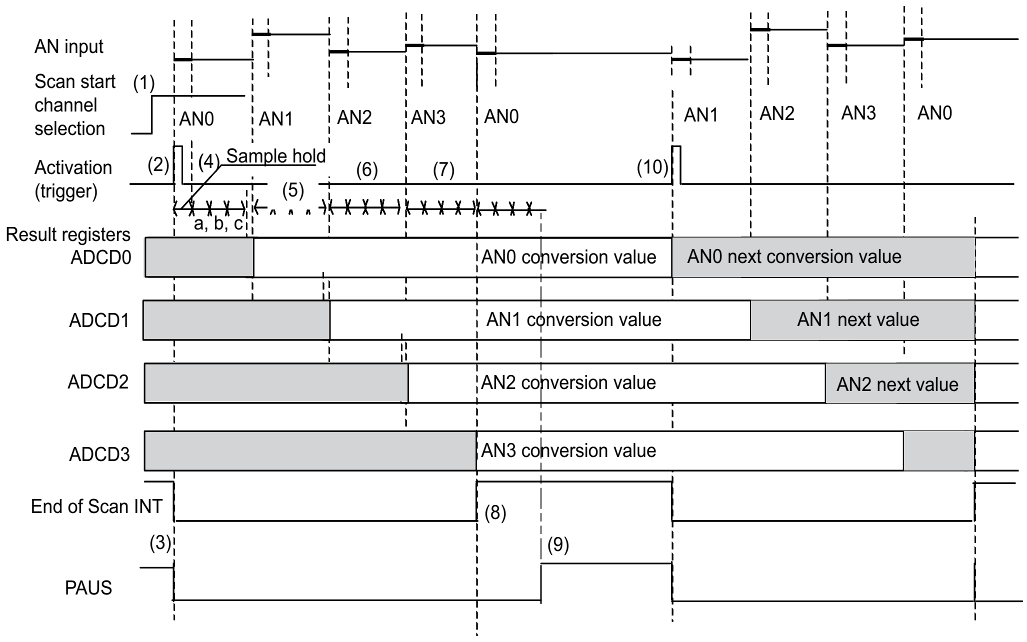


Figure 6-12: Operation of the ADC in scan conversion mode

(1) Activation channel selection
(2) A/D activation (Trigger: Software trigger/Reload timer/External trigger)
(3) ADCn_CS1:INT flag clear, ADCn_CS1:PAUS flag clear.
(4) AN0 conversion
 (a) Sample hold, conversion
 (b) Conversion end
 (c) Buffers the conversion value
(5) AN1 conversion
(6) AN2 conversion
(7) AN3 conversion
(8) ADCn_CS3:INT2 (End of Scan) flag is set, AN0 conversion starts
(9) It is assumed that the data protection feature is enabled (ADCn_CS2:DPDIS = '0'). Since ADCn_CS3:INT2 has not been cleared yet, ADC protects the value in ADCn_CD0 (data register of AN0) against overwriting, pauses the conversion operation and sets ADCn_CS1:PAUS bit.
(10) ADCn_CS3:INT2 flag is cleared by software, the ADC stores the result of AN0 in ADCn_CD0 and continues sampling AN1. The ADCn_CS1:PAUS bit is made low by the software to indicate a new occurrence of pause condition to the user when it occurs.

6.3.3.6 Protection of the ADC Channel Data Registers

There are 28 ADC result data registers, one register per channel. The registers are written by hardware at the end of conversion of the attached channel. ADCn_CD0 is attached to channel 0, ADCn_CD27 is attached to channel 27.

The host CPU or Command Sequencer can read the data registers any time. If a conversion is finished and the data of the previous conversion has not been read out, previous data would be overwritten. To avoid this problem, the next conversion data is not stored in the data registers before the appropriate interrupt flag (ADCn_CS1:INT or ADCn_CS3:INT2) is cleared. A/D conversion halts during this time and the ADCn_CS1:PAUS flag is set.

The register protection function depends on the conversion mode and the setting of ADCn_CS3:INTE2, ADCn_CS1:INTE, ADCn_MAR:DRQEN, and also on ADCn_CS2:DPDIS bits.
If ADCn_CS2:DPDIS = '1' then conversion is continued and former conversion data may be overwritten. Else, if ADCn_CS2:DPDIS = '0' the former conversion data is not overwritten before the appropriate interrupt flag (ADCn_CS1:INT or ADCn_CS3:INT2) is cleared. The following table gives the protection scheme.

Table 6-6: Protection of the ADC Channel Data register

Mode	ADCn_ CS3: INTE2	ADCn_ CS1: INTE	ADCn_ MAR: DRQEN	Function
Continuous	‘0’	‘0’	‘0’	ADCn_CS1:PAUS is not set because no interrupt request is generated (although the flags are set) and no DMA transfer is requested.
	‘0’	‘1’	‘0’	ADCn_CS1:PAUS is set after each channel conversion when the interrupt request ADCn_CS1:INT of the preceding conversion has not been cleared. To resume conversion, the ADCn_CS1:INT flag must be cleared by writing '1' to ADCn_CSC1:INTC. Or by reading the ADCn_CR register when ADCn_CS2:INTAC is enabled.
	‘0’	X	‘1’	ADCn_CS1:PAUS is set after each channel conversion when the interrupt request ADCn_CS1:INT of the preceding conversion has not been cleared. To resume conversion, the ADCn_CS1:INT flag must be cleared by writing '1' to ADCn_CSC1:INTC or by reading the ADCn_CR register.
	‘1’	‘0’	‘0’	ADCn_CS1:PAUS is set after the conversion of the start channel (defined by ADCn_SCH register) when the interrupt request ADCn_CS3:INT2 of the preceding scan has not been cleared. To resume conversion, the ADCn_CS3:INT2 flag must be cleared by writing '1' to ADCn_CSC3:INT2C.
	‘1’	‘1’	‘0’	ADCn_CS1:PAUS is set after each channel conversion when the interrupt request ADCn_CS1:INT of the preceding conversion or the interrupt request ADCn_CS3:INT2 of the preceding scan has not been cleared. To resume conversion, the ADCn_CS1:INT and ADCn_CS3:INT2 flags must be cleared by writing '1' to ADCn_CSC1:INTC and ADCn_CSC3:INT2C respectively. ADCn_CS1:INT can also be cleared by reading the ADCn_CR register when ADCn_CS2:INTAC is enabled.
	‘1’	X	‘1’	ADCn_CS1:PAUS is set after each channel conversion when the interrupt request ADCn_CS1:INT of the preceding conversion or the interrupt request ADCn_CS3:INT2 of the preceding scan has not been cleared. To resume conversion, the ADCn_CS1:INT and ADCn_CS3:INT2 flags must be cleared. ADCn_CS1:INT is cleared by writing '1' to ADCn_CSC1:INTC or by reading the ADCn_CR register. ADCn_CS3:INT2 flag is cleared by writing '1' to ADCn_CSC3:INT2C.

Table 6-6: Protection of the ADC Channel Data register

Mode	ADCn_ CS3: INTE2	ADCn_ CS1: INTE	ADCn_ MAR: DRQEN	Function
Others	X	‘0’	‘0’	ADCn_CS1:PAUS is not set because no interrupt request is generated (although the flags are set) and no DMA transfer is requested.
	X	‘1’	‘0’	ADCn_CS1:PAUS is set after each channel conversion when the interrupt request ADCn_CS1:INT of the preceding conversion has not been cleared. To resume conversion, the ADCn_CS1:INT flag must be cleared by writing '1' to ADCn_CSC1:INTC. Or by reading the ADCn_CR register when ADCn_CS2:INTAC is enabled.
	X	X	‘1’	ADCn_CS1:PAUS is set after each channel conversion when the interrupt request ADCn_CS1:INT of the preceding conversion has not been cleared. To resume conversion, the ADCn_CS1:INT flag must be cleared by writing '1' to ADCn_CSC1:INTC or by reading the ADCn_CR register.
Note: X signifies a ‘don’t care’ condition.				

6.3.3.7 Protection of ADCn_CD27~0

In continuous mode with ADCn_CS3:INTE2=1, ADCn_CS1:PAUS is set when data of the start channel (set by ADCn_SCH) is ready for writing to the registers, but ADCn_CS3:INT2 (End of Scan interrupt) is already active.

Example:

Start channel=4, end channel=7, continuous mode, ADCn_CS1:INTE=0, ADCn_CS3:INTE2=1

Start by CPU → convert channel 4 + safe data to ADCn_CD4,
convert channel 5 + save data to ADCn_CD5,
convert channel 6 + save data to ADCn_CD6,
convert channel 7 + save data to ADCn_CD7 → End of Scan interrupt (INT2),
convert channel 4 + set CS1:PAUS (protect ADCn_CD4...7).

After the CPU has read the data registers and cleared ADCn_CS3:INT2, the scan conversion continues.

This function can be disabled by setting ADCn_CS2:DPDIS = 1. Then conversion is continued and former conversion data are overwritten.

6.3.3.8 Protection of ADCn_CR

If ADCn_CS1:INTE = '1' or ADCn_MAR:DRQEN = '1', ADCn_CS1:PAUS is set when data of any channel is ready for writing to the common data register (and the dedicated data register) but ADCn_CS1:INT (end of conversion) is active. In this mode the protection function is active after each single conversion, the ADCn_CR register is protected.

6.3.3.9 DMA Transfer

DMA transfer can be triggered by end of conversion interrupt or by end of scan interrupt provided the DMA transfer is enabled by ADCn_MAR:DRQEN and ADCn_MAR:DRQEN2 respectively.

6.3.3.9.1 Data Protection During DMA Transfer

During DMA mode of data transfer (ADCn_MAR:DRQEN = '1') the data of the ADCn_CR register is protected against overwriting if ADCn_CS2:DPDIS = '0' (data protection enabled) until the ADCn_CR register value is read (except during debug master (DAP) access). After the DMA read is over the ADCn_CS1:INT bit is internally cleared when any master reads the register.

There is no data protection function available for the end of scan DMA request (ADCn_MAR:DRQEN2 = '1').

6.3.3.10 ADC Pulse Detection Function

6.3.3.10.1 Positive Events/negative Events

The output of the range comparator signifies either a positive event or a negative event depending on the configuration of ADCn_RCOIRS32, ADCn_RCOIRS10 registers and the converted digital value of the ADC.

If the range comparator is configured for 'inside range' comparison, any ADC value within the range will be treated as a positive event whereas ADC values above the upper or below the lower thresholds are treated as negative events. If the range comparator is configured for 'outside range' comparison, any ADC value within the range will be treated as negative event while ADC values above the upper or below the lower thresholds are treated as positive events. Table 6-7 shows the range comparator settings and appropriate event definition.

Table 6-7: Range comparator settings

ADCn_RCOIRS32:RCOIRS, ADCn_RCOIRS10:RCOIRS value	Range comparator output	Events
'0' (configured for 'outside range')	Inside range	Negative event
	Outside range	Positive event
'1' (configured for 'inside range')	Outside range	Negative event
	Inside range	Positive event

Whenever a positive event occurs, the positive counter ADCn_PCTPCT31~0 is decremented. Similarly, whenever a corresponding negative event occurs, the corresponding negative counter ADCn_PCTNCT31~0 is decremented.

6.3.3.10.2 Working Principle of ADC Pulse Detection Function

Each ADC channel has its own filter with

– positive counter (ADCn_PCTPCT27~0),
– negative counter (ADCn_PCTNCT27~0),
– along with their reload registers (ADCn_PCTPRL27~0/ ADCn_PCTNRL27~0),
– zero flag (ADCn_PCZF32, ADCn_PCZF10:CTPZF),
– and zero flag clear bit (ADCn_PCZFC32, ADCn_PCZFC10:CTPZFC).

It also has

– the Interrupt Enable Register (ADCn_PCIE32, ADCn_PCIE10:CTPIE),
– the Interrupt Enable Set Register (ADCn_PCIES32, ADCn_PCIES10:CTPIES),
– and the Interrupt Enable Clear Register (ADCn_PCIEC32, ADCn_PCIEC10:CTPIEC).

The purpose of the positive counter is to detect consecutive ADC range comparator events of desired length. The negative counter can be used to force a restart of the positive counter if a negative signal of a certain length is detected due to spikes, noise etc.

The following steps describe the working principle.

■ The positive counter ADCn_PCTPCT27~0 decrements with each positive event of the corresponding ADC channel

- The zero flag ADCn_PCZF32, ADCn_PCZF10:CTPZF is set as the positive counter reaches zero. This flag remains set until it is cleared through ADCn_PCZFC32, ADCn_PCZFC10:CTPZFC. The positive counter and the negative counter are stopped as long as zero flag of the corresponding channel is '1'
- If the zero flag (ADCn_PCZFC32, ADCn_PCZF10:CTPZF) is set and the corresponding pulse detection interrupt is enabled (ADCn_PCIE32, ADCn_PCIE10:CTPIE = '1') then an interrupt is generated
- The negative counter ADCn_PCTNCT27~0 decrements with each negative event of the corresponding ADC channel except while the corresponding zero flag is set
- The following conditions will cause a reload of the positive and/or negative counter with the values set in their reload registers ADCn_PCTPRL27~0 and ADCn_PCTNRL27~0:
  - **Positive counter is reloaded when**

    1. Negative counter reaches zero.
    2. '1' is written to ADCn_PCZFC32, ADCn_PCZFC10:CTPZFC (independent of actual value of the corresponding ADCn_PCZF32, ADCn_PCZF10:CTPZF).
  - **Negative counter is reloaded when**

    1. Any positive event occurs.
    2. '1' is written to ADCn_PCZFC32, ADCn_PCZFC10:CTPZFC (independent of actual value of the corresponding ADCn_PCZF32, ADCn_PCZF10:CTPZF).
    3. The positive counter reaches zero and the zero flag is set. The negative counter will hold the reload value as long as the zero flag is not cleared.

Figure 6-13 shows the operation of the ADC pulse detection function for channel '0' with ADCn_RCOIRS32:RCOIRS[0] = '0' configured for outside range, positive reload register ADCn_PCTPRL0 = '100' and negative reload register ADCn_PCTNRL0 = '010'.

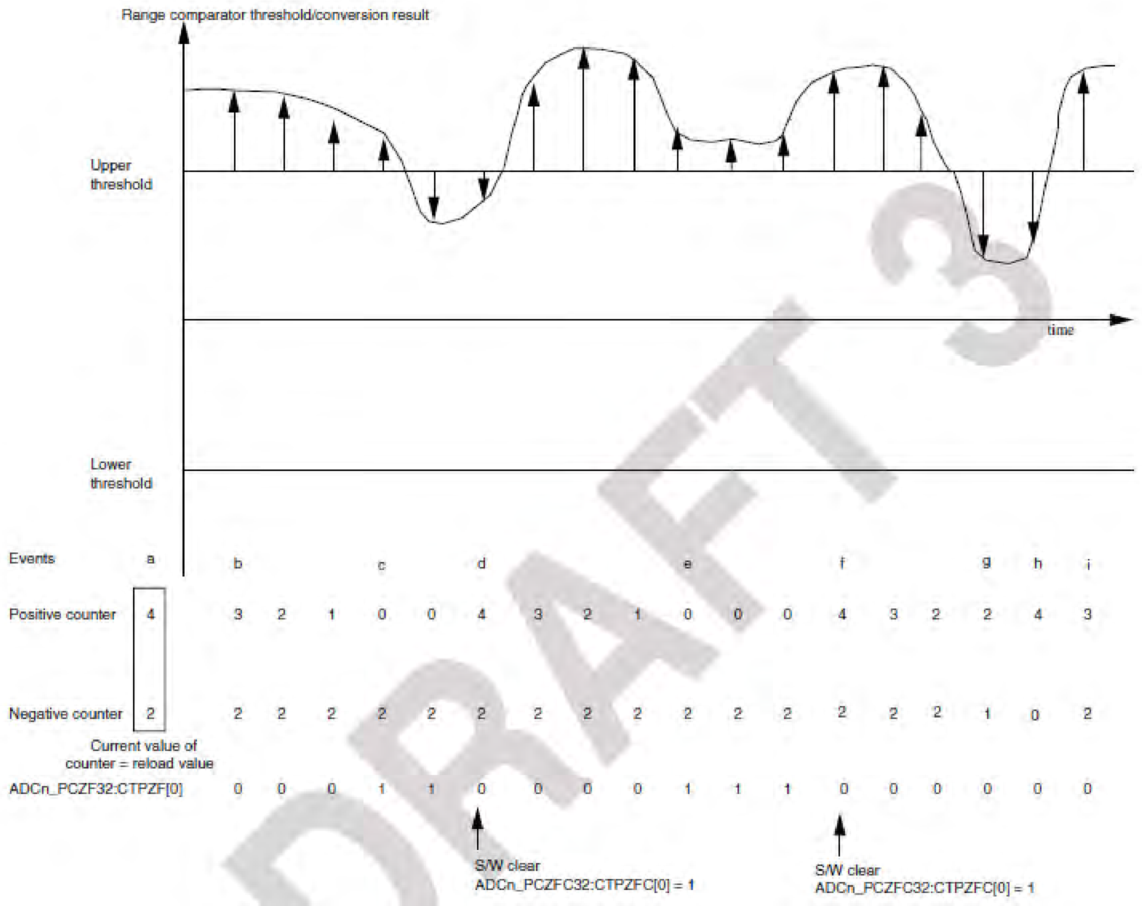

Range comparator threshold/conversion result
Upper threshold
time
Lower threshold
Events a b c d e f g h i
Positive counter 4 3 2 1 0 0 4 3 2 1 0 0 0 4 3 2 2 4 3
Negative counter 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 0 2
Current value of counter = reload value
ADCn_PCZF32:CTPZF[0] 0 0 0 1 1 0 0 0 0 1 1 1 0 0 0 0 0 0
S/W clear ADCn_PCZFC32:CTPZFC[0] = 1
S/W clear ADCn_PCZFC32:CTPZFC[0] = 1

Figure 6-13: Example of ADC pulse detection function

a) Reload counters with appropriate reload value by writing ‘1’ to ADCn_PCZFC32,ADCn_PCZFC10:CTPZFC[0].

b) Positive counter decrements with positive events.

c) Positive counter expires, zero flag (ADCn_PCZF32, ADCn_PCZF10:CTPZF[0]) is set.

d) A series of negative events does not decrement the negative counter as the ADCn_PCZF32, ADCn_PCZF10:CTPZF[0] (zero flag) is set. The zero flag ADCn_PCZF32, ADCn_PCZF10:CTPZF[0] is cleared and the positive as well as the negative counter are reloaded.

e) Positive counter expires, zero flag ADCn_PCZF32, ADCn_PCZF10:CTPZF[0] = '1'.

f) Software clear of ADCn_PCZF32, ADCn_PCZF10:CTPZF[0] reloads positive and negative counter.

g) Negative event decrements negative counter.

h) Negative counter expires and reloads positive counter.

i) Positive counter decrements with positive event.

6.3.4 ADC Software Interface

The base address for the ADC register space is **0x00098000.** The hardware uses channels 0 to 15 for ADC input and channels 16 to 27 for the Stepper Motor Zero Point Detection.

NOTE Channels 16 - 27 need to be enabled with register IFC_CTRL.zpd_adc_en.

6.3.5 Analog-Digital Converter Register Overview

Table 6-8: Registers Overview

Base Address(es)	Instance no 0: BASEADDR="00098000"	
Absolute Address	**Register Name**	**Register Description**
BASEADDR + 0x0000	ADCn_ER32	ADC Input Enable Register 32
BASEADDR + 0x0002	ADCn_ER10	ADC Input Enable Register 10
BASEADDR + 0x0004	ADCn_CS0	ADC Control Status Register 0
BASEADDR + 0x0005	ADCn_CS1	ADC Control Status Register 1
BASEADDR + 0x0006	ADCn_CS2	ADC Control Status Register 2
BASEADDR + 0x0007	ADCn_CS3	ADC Control Status Register 3
BASEADDR + 0x0008	Reserved	Do not modify
BASEADDR + 0x0009	ADCn_CSS1	ADC Control Status Register 1 Set Register
BASEADDR + 0x000A	Reserved	Do not modify
BASEADDR + 0x000B	ADCn_CSC1	ADC Control Status Register 1 Clear Register
BASEADDR + 0x000C	Reserved	Do not modify
BASEADDR + 0x000D	ADCn_CSS3	ADC Control Status Register 3 Set Register
BASEADDR + 0x000E	Reserved	Do not modify
BASEADDR + 0x000F	ADCn_CSC3	ADC Control Status Register 3 Clear Register
BASEADDR + 0x0010	ADCn_CR	Common Data Register
BASEADDR + 0x0012	ADCn_MIR_CS0	Read only Mirror of Status Register 0
BASEADDR + 0x0013	ADCn_MIR_CS3	Read only Mirror of Status Register 3
BASEADDR + 0x0014	Reserved	Do not modify
BASEADDR + 0x0016	Reserved	Do not modify
BASEADDR + 0x0018	ADCn_CD0	Dedicated ADC Channel 0 Data Register
BASEADDR + 0x001A	ADCn_CD1	Dedicated ADC Channel 1 Data Register
BASEADDR + 0x001C	ADCn_CD2	Dedicated ADC Channel 2 Data Register
BASEADDR + 0x001E	ADCn_CD3	Dedicated ADC Channel 3 Data Register
BASEADDR + 0x0020	ADCn_CD4	Dedicated ADC Channel 4 Data Register
BASEADDR + 0x0022	ADCn_CD5	Dedicated ADC Channel 5 Data Register
BASEADDR + 0x0024	ADCn_CD6	Dedicated ADC Channel 6 Data Register
BASEADDR + 0x0026	ADCn_CD7	Dedicated ADC Channel 7 Data Register
BASEADDR + 0x0028	ADCn_CD8	Dedicated ADC Channel 8 Data Register
BASEADDR + 0x002A	ADCn_CD9	Dedicated ADC Channel 9 Data Register
BASEADDR + 0x002C	ADCn_CD10	Dedicated ADC Channel 10 Data Register
BASEADDR + 0x002E	ADCn_CD11	Dedicated ADC Channel 11 Data Register
BASEADDR + 0x0030	ADCn_CD12	Dedicated ADC Channel 12 Data Register
BASEADDR + 0x0032	ADCn_CD13	Dedicated ADC Channel 13 Data Register
BASEADDR + 0x0034	ADCn_CD14	Dedicated ADC Channel 14 Data Register
BASEADDR + 0x0036	ADCn_CD15	Dedicated ADC Channel 15 Data Register
BASEADDR + 0x0038	ADCn_CD16	Dedicated ADC Channel 16 Data Register
BASEADDR + 0x003A	ADCn_CD17	Dedicated ADC Channel 17 Data Register
BASEADDR + 0x003C	ADCn_CD18	Dedicated ADC Channel 18 Data Register
BASEADDR + 0x003E	ADCn_CD19	Dedicated ADC Channel 19 Data Register
BASEADDR + 0x0040	ADCn_CD20	Dedicated ADC Channel 20 Data Register
BASEADDR + 0x0042	ADCn_CD21	Dedicated ADC Channel 21 Data Register
BASEADDR + 0x0044	ADCn_CD22	Dedicated ADC Channel 22 Data Register
BASEADDR + 0x0046	ADCn_CD23	Dedicated ADC Channel 23 Data Register
BASEADDR + 0x0048	ADCn_CD24	Dedicated ADC Channel 24 Data Register
BASEADDR + 0x004A	ADCn_CD25	Dedicated ADC Channel 25 Data Register
BASEADDR + 0x004C	ADCn_CD26	Dedicated ADC Channel 26 Data Register
BASEADDR + 0x004E	ADCn_CD27	Dedicated ADC Channel 27 Data Register

Table 6-8: Registers Overview (Continued)

Base Address(es)	Instance no 0: BASEADDR="00098000"	
Absolute Address	**Register Name**	**Register Description**
BASEADDR + 0x0050	Reserved	Do not modify
BASEADDR + 0x0052	Reserved	Do not modify
BASEADDR + 0x0054	Reserved	Do not modify
BASEADDR + 0x0056	Reserved	Do not modify
BASEADDR + 0x0058	Reserved	Do not modify
BASEADDR + 0x005A	Reserved	Do not modify
BASEADDR + 0x005C	Reserved	Do not modify
BASEADDR + 0x005E	ADCn_CT	ADC Conversion Time Setting Register
BASEADDR + 0x0060	ADCn_SCH	ADC Start Channel Setting Register
BASEADDR + 0x0061	ADCn_ECH	ADC End Channel Setting Register
BASEADDR + 0x0062	ADCn_MAR	ADC DMA Configuration Register
BASEADDR + 0x0063	Reserved	Do not modify
BASEADDR + 0x0064	ADCn_MACR	ADC DMA Configuration Register Clear Register
BASEADDR + 0x0065	Reserved	Do not modify
BASEADDR + 0x0066	ADCn_MASR	ADC DMA Configuration Register Set Register
BASEADDR + 0x0067	Reserved	Do not modify
BASEADDR + 0x0068	ADCn_RCOL0	Range Comparator Lower Threshold Register 0
BASEADDR + 0x0069	ADCn_RCOH0	Range Comparator Upper Threshold Register 0
BASEADDR + 0x006A	ADCn_RCOL1	Range Comparator Lower Threshold Register 1
BASEADDR + 0x006B	ADCn_RCOH1	Range Comparator Upper Threshold Register 1
BASEADDR + 0x006C	ADCn_RCOL2	Range Comparator Lower Threshold Register 2
BASEADDR + 0x006D	ADCn_RCOH2	Range Comparator Upper Threshold Register 2
BASEADDR + 0x006E	ADCn_RCOL3	Range Comparator Lower Threshold Register 0
BASEADDR + 0x006F	ADCn_RCOH3	Range Comparator Upper Threshold Register 3
BASEADDR + 0x0070	ADCn_CC0	ADC Converter Channel Control Register 0
BASEADDR + 0x0071	ADCn_CC1	ADC Converter Channel Control Register 1
BASEADDR + 0x0072	ADCn_CC2	ADC Converter Channel Control Register 2
BASEADDR + 0x0073	ADCn_CC3	ADC Converter Channel Control Register 3
BASEADDR + 0x0074	ADCn_CC4	ADC Converter Channel Control Register 4
BASEADDR + 0x0075	ADCn_CC5	ADC Converter Channel Control Register 5
BASEADDR + 0x0076	ADCn_CC6	ADC Converter Channel Control Register 6
BASEADDR + 0x0077	ADCn_CC7	ADC Converter Channel Control Register 7
BASEADDR + 0x0078	ADCn_CC8	ADC Converter Channel Control Register 8
BASEADDR + 0x0079	ADCn_CC9	ADC Converter Channel Control Register 9
BASEADDR + 0x007A	ADCn_CC10	ADC Converter Channel Control Register 10
BASEADDR + 0x007B	ADCn_CC11	ADC Converter Channel Control Register 11
BASEADDR + 0x007C	ADCn_CC12	ADC Converter Channel Control Register 12
BASEADDR + 0x007D	ADCn_CC13	ADC Converter Channel Control Register 13
BASEADDR + 0x007E	Reserved	Do not modify
BASEADDR + 0x007F	Reserved	Do not modify
BASEADDR + 0x0080	ADCn_RCOIRS32	Inverted Range Selection Register 32
BASEADDR + 0x0082	ADCn_RCOIRS10	Inverted Range Selection Register 32
BASEADDR + 0x0084	ADCn_RCOOF32	Range Comparator over threshold flag Register 32
BASEADDR + 0x0086	ADCn_RCOOF10	Range Comparator over threshold flag Register 10
BASEADDR + 0x0088	ADCn_RCOINT32	Range Comparator Interrupt flag Register 32
BASEADDR + 0x008A	ADCn_RCOINT10	Range Comparator Interrupt flag Register 32
BASEADDR + 0x008C	ADCn_RCOINTC32	Range Comparator Interrupt Clear Register 32
BASEADDR + 0x008E	ADCn_RCOINTC10	Range Comparator Interrupt Clear Register 10
BASEADDR + 0x0090	ADCn_PCTPRL0	ADC Pulse Counter Positive Reload Register 0
BASEADDR + 0x0091	ADCn_PCTNRL0	ADC Pulse Counter Negative Reload Register 0

Table 6-8: Registers Overview (Continued)

Base Address(es)	**Instance no 0: BASEADDR="00098000"**	
Absolute Address	**Register Name**	**Register Description**
BASEADDR + 0x0092	ADCn_PCTPCT0	ADC Pulse Counter Positive Count Register 0
BASEADDR + 0x0093	ADCn_PCTNCT0	ADC Pulse Counter Negative Count Register 0
BASEADDR + 0x0094	ADCn_PCTPRL1	ADC Pulse Counter Positive Reload Register 1
BASEADDR + 0x0095	ADCn_PCTNRL1	ADC Pulse Counter Negative Reload Register 1
BASEADDR + 0x0096	ADCn_PCTPCT1	ADC Pulse Counter Positive Count Register 1
BASEADDR + 0x0097	ADCn_PCTNCT1	ADC Pulse Counter Negative Count Register 1
BASEADDR + 0x0098	ADCn_PCTPRL2	ADC Pulse Counter Positive Reload Register 2
BASEADDR + 0x0099	ADCn_PCTNRL2	ADC Pulse Counter Negative Reload Register 2
BASEADDR + 0x009A	ADCn_PCTPCT2	ADC Pulse Counter Positive Count Register 2
BASEADDR + 0x009B	ADCn_PCTNCT2	ADC Pulse Counter Negative Count Register 2
BASEADDR + 0x009C	ADCn_PCTPRL3	ADC Pulse Counter Positive Reload Register 3
BASEADDR + 0x009D	ADCn_PCTNRL3	ADC Pulse Counter Negative Reload Register 3
BASEADDR + 0x009E	ADCn_PCTPCT3	ADC Pulse Counter Positive Count Register 3
BASEADDR + 0x009F	ADCn_PCTNCT3	ADC Pulse Counter Negative Count Register 3
BASEADDR + 0x00A0	ADCn_PCTPRL4	ADC Pulse Counter Positive Reload Register 4
BASEADDR + 0x00A1	ADCn_PCTNRL4	ADC Pulse Counter Negative Reload Register 4
BASEADDR + 0x00A2	ADCn_PCTPCT4	ADC Pulse Counter Positive Count Register 4
BASEADDR + 0x00A3	ADCn_PCTNCT4	ADC Pulse Counter Negative Count Register 4
BASEADDR + 0x00A4	ADCn_PCTPRL5	ADC Pulse Counter Positive Reload Register 5
BASEADDR + 0x00A5	ADCn_PCTNRL5	ADC Pulse Counter Negative Reload Register 5
BASEADDR + 0x00A6	ADCn_PCTPCT5	ADC Pulse Counter Positive Count Register 5
BASEADDR + 0x00A7	ADCn_PCTNCT5	ADC Pulse Counter Negative Count Register 5
BASEADDR + 0x00A8	ADCn_PCTPRL6	ADC Pulse Counter Positive Reload Register 6
BASEADDR + 0x00A9	ADCn_PCTNRL6	ADC Pulse Counter Negative Reload Register 6
BASEADDR + 0x00AA	ADCn_PCTPCT6	ADC Pulse Counter Positive Count Register 6
BASEADDR + 0x00AB	ADCn_PCTNCT6	ADC Pulse Counter Negative Count Register 6
BASEADDR + 0x00AC	ADCn_PCTPRL7	ADC Pulse Counter Positive Reload Register 7
BASEADDR + 0x00AD	ADCn_PCTNRL7	ADC Pulse Counter Negative Reload Register 7
BASEADDR + 0x00AE	ADCn_PCTPCT7	ADC Pulse Counter Positive Count Register 7
BASEADDR + 0x00AF	ADCn_PCTNCT7	ADC Pulse Counter Negative Count Register 7
BASEADDR + 0x00B0	ADCn_PCTPRL8	ADC Pulse Counter Positive Reload Register 8
BASEADDR + 0x00B1	ADCn_PCTNRL8	ADC Pulse Counter Negative Reload Register 8
BASEADDR + 0x00B2	ADCn_PCTPCT8	ADC Pulse Counter Positive Count Register 8
BASEADDR + 0x00B3	ADCn_PCTNCT8	ADC Pulse Counter Negative Count Register 8
BASEADDR + 0x00B4	ADCn_PCTPRL9	ADC Pulse Counter Positive Reload Register 9
BASEADDR + 0x00B5	ADCn_PCTNRL9	ADC Pulse Counter Negative Reload Register 9
BASEADDR + 0x00B6	ADCn_PCTPCT9	ADC Pulse Counter Positive Count Register 9
BASEADDR + 0x00B7	ADCn_PCTNCT9	ADC Pulse Counter Negative Count Register 9
BASEADDR + 0x00B8	ADCn_PCTPRL10	ADC Pulse Counter Positive Reload Register 10
BASEADDR + 0x00B9	ADCn_PCTNRL10	ADC Pulse Counter Negative Reload Register 10
BASEADDR + 0x00BA	ADCn_PCTPCT10	ADC Pulse Counter Positive Count Register 10
BASEADDR + 0x00BB	ADCn_PCTNCT10	ADC Pulse Counter Negative Count Register 10
BASEADDR + 0x00BC	ADCn_PCTPRL11	ADC Pulse Counter Positive Reload Register 11
BASEADDR + 0x00BD	ADCn_PCTNRL11	ADC Pulse Counter Negative Reload Register 11
BASEADDR + 0x00BE	ADCn_PCTPCT11	ADC Pulse Counter Positive Count Register 11
BASEADDR + 0x00BF	ADCn_PCTNCT11	ADC Pulse Counter Negative Count Register 11
BASEADDR + 0x00C0	ADCn_PCTPRL12	ADC Pulse Counter Positive Reload Register 12
BASEADDR + 0x00C1	ADCn_PCTNRL12	ADC Pulse Counter Negative Reload Register 12
BASEADDR + 0x00C2	ADCn_PCTPCT12	ADC Pulse Counter Positive Count Register 12
BASEADDR + 0x00C3	ADCn_PCTNCT12	ADC Pulse Counter Negative Count Register 12

Table 6-8: Registers Overview (Continued)

Base Address(es)	**Instance no 0: BASEADDR="00098000"**	
Absolute Address	**Register Name**	**Register Description**
BASEADDR + 0x00C4	ADCn_PCTPRL13	ADC Pulse Counter Positive Reload Register 13
BASEADDR + 0x00C5	ADCn_PCTNRL13	ADC Pulse Counter Negative Reload Register 13
BASEADDR + 0x00C6	ADCn_PCTPCT13	ADC Pulse Counter Positive Count Register 13
BASEADDR + 0x00C7	ADCn_PCTNCT13	ADC Pulse Counter Negative Count Register 13
BASEADDR + 0x00C8	ADCn_PCTPRL14	ADC Pulse Counter Positive Reload Register 14
BASEADDR + 0x00C9	ADCn_PCTNRL14	ADC Pulse Counter Negative Reload Register 14
BASEADDR + 0x00CA	ADCn_PCTPCT14	ADC Pulse Counter Negative Count Register 14
BASEADDR + 0x00CB	ADCn_PCTNCT14	ADC Pulse Counter Positive Count Register 14
BASEADDR + 0x00CC	ADCn_PCTPRL15	ADC Pulse Counter Positive Reload Register 15
BASEADDR + 0x00CD	ADCn_PCTNRL15	ADC Pulse Counter Negative Reload Register 15
BASEADDR + 0x00CE	ADCn_PCTPCT15	ADC Pulse Counter Positive Count Register 15
BASEADDR + 0x00CF	ADCn_PCTNCT15	ADC Pulse Counter Negative Count Register 15
BASEADDR + 0x00D0	ADCn_PCTPRL16	ADC Pulse Counter Positive Reload Register 16
BASEADDR + 0x00D1	ADCn_PCTNRL16	ADC Pulse Counter Negative Reload Register 16
BASEADDR + 0x00D2	ADCn_PCTPCT16	ADC Pulse Counter Positive Count Register 16
BASEADDR + 0x00D3	ADCn_PCTNCT16	ADC Pulse Counter Negative Count Register 16
BASEADDR + 0x00D4	ADCn_PCTPRL17	ADC Pulse Counter Positive Reload Register 17
BASEADDR + 0x00D5	ADCn_PCTNRL17	ADC Pulse Counter Negative Reload Register 17
BASEADDR + 0x00D6	ADCn_PCTPCT17	ADC Pulse Counter Positive Count Register 17
BASEADDR + 0x00D7	ADCn_PCTNCT17	ADC Pulse Counter Negative Count Register 17
BASEADDR + 0x00D8	ADCn_PCTPRL18	ADC Pulse Counter Positive Reload Register 18
BASEADDR + 0x00D9	ADCn_PCTNRL18	ADC Pulse Counter Negative Reload Register 18
BASEADDR + 0x00DA	ADCn_PCTPCT18	ADC Pulse Counter Positive Count Register 18
BASEADDR + 0x00DB	ADCn_PCTNCT18	ADC Pulse Counter Negative Count Register 18
BASEADDR + 0x00DC	ADCn_PCTPRL19	ADC Pulse Counter Positive Reload Register 19
BASEADDR + 0x00DD	ADCn_PCTNRL19	ADC Pulse Counter Negative Reload Register 19
BASEADDR + 0x00DE	ADCn_PCTPCT19	ADC Pulse Counter Positive Count Register 19
BASEADDR + 0x00DF	ADCn_PCTNCT19	ADC Pulse Counter Negative Count Register 19
BASEADDR + 0x00E0	ADCn_PCTPRL20	ADC Pulse Counter Positive Reload Register 20
BASEADDR + 0x00E1	ADCn_PCTNRL20	ADC Pulse Counter Negative Reload Register 20
BASEADDR + 0x00E2	ADCn_PCTPCT20	ADC Pulse Counter Positive Count Register 20
BASEADDR + 0x00E3	ADCn_PCTNCT20	ADC Pulse Counter Negative Count Register 20
BASEADDR + 0x00E4	ADCn_PCTPRL21	ADC Pulse Counter Positive Reload Register 21
BASEADDR + 0x00E5	ADCn_PCTNRL21	ADC Pulse Counter Negative Reload Register 21
BASEADDR + 0x00E6	ADCn_PCTPCT21	ADC Pulse Counter Positive Count Register 21
BASEADDR + 0x00E7	ADCn_PCTNCT21	ADC Pulse Counter Negative Count Register 21
BASEADDR + 0x00E8	ADCn_PCTPRL22	ADC Pulse Counter Positive Reload Register 22
BASEADDR + 0x00E9	ADCn_PCTNRL22	ADC Pulse Counter Negative Reload Register 22
BASEADDR + 0x00EA	ADCn_PCTPCT22	ADC Pulse Counter Positive Count Register 22
BASEADDR + 0x00EB	ADCn_PCTNCT22	ADC Pulse Counter Negative Count Register 22
BASEADDR + 0x00EC	ADCn_PCTPRL23	ADC Pulse Counter Positive Reload Register 23
BASEADDR + 0x00ED	ADCn_PCTNRL23	ADC Pulse Counter Negative Reload Register 23
BASEADDR + 0x00EE	ADCn_PCTPCT23	ADC Pulse Counter Positive Count Register 23
BASEADDR + 0x00EF	ADCn_PCTNCT23	ADC Pulse Counter Negative Count Register 23
BASEADDR + 0x00F0	ADCn_PCTPRL24	ADC Pulse Counter Positive Reload Register 24
BASEADDR + 0x00F1	ADCn_PCTNRL24	ADC Pulse Counter Negative Reload Register 24
BASEADDR + 0x00F2	ADCn_PCTPCT24	ADC Pulse Counter Positive Count Register 24
BASEADDR + 0x00F3	ADCn_PCTNCT24	ADC Pulse Counter Negative Count Register 24
BASEADDR + 0x00F4	ADCn_PCTPRL25	ADC Pulse Counter Positive Reload Register 25
BASEADDR + 0x00F5	ADCn_PCTNRL25	ADC Pulse Counter Negative Reload Register 25

Table 6-8: Registers Overview (Continued)

Base Address(es)	Instance no 0: BASEADDR="00098000"	
Absolute Address	**Register Name**	**Register Description**
BASEADDR + 0x00F6	ADCn_PCTPCT25	ADC Pulse Counter Positive Count Register 25
BASEADDR + 0x00F7	ADCn_PCTNCT25	ADC Pulse Counter Negative Count Register 25
BASEADDR + 0x00F8	ADCn_PCTPRL26	ADC Pulse Counter Positive Reload Register 26
BASEADDR + 0x00F9	ADCn_PCTNRL26	ADC Pulse Counter Negative Reload Register 26
BASEADDR + 0x00FA	ADCn_PCTPCT26	ADC Pulse Counter Positive Count Register 26
BASEADDR + 0x00FB	ADCn_PCTNCT26	ADC Pulse Counter Negative Count Register 26
BASEADDR + 0x00FC	ADCn_PCTPRL27	ADC Pulse Counter Positive Reload Register 27
BASEADDR + 0x00FD	ADCn_PCTNRL27	ADC Pulse Counter Negative Reload Register 27
BASEADDR + 0x00FE	ADCn_PCTPCT27	ADC Pulse Counter Positive Count Register 27
BASEADDR + 0x00FF	ADCn_PCTNCT27	ADC Pulse Counter Negative Count Register 28
BASEADDR + 0x0100	Reserved	Do not modify
BASEADDR + 0x0101	Reserved	Do not modify
BASEADDR + 0x0102	Reserved	Do not modify
BASEADDR + 0x0103	Reserved	Do not modify
BASEADDR + 0x0104	Reserved	Do not modify
BASEADDR + 0x0105	Reserved	Do not modify
BASEADDR + 0x0106	Reserved	Do not modify
BASEADDR + 0x0107	Reserved	Do not modify
BASEADDR + 0x0108	Reserved	Do not modify
BASEADDR + 0x0109	Reserved	Do not modify
BASEADDR + 0x010A	Reserved	Do not modify
BASEADDR + 0x010B	Reserved	Do not modify
BASEADDR + 0x010C	Reserved	Do not modify
BASEADDR + 0x010D	Reserved	Do not modify
BASEADDR + 0x010E	Reserved	Do not modify
BASEADDR + 0x010F	Reserved	Do not modify
BASEADDR + 0x0110	ADCn_PCZF10	ADC Pulse Counter Zero Flag Register 10
BASEADDR + 0x0112	ADCn_PCZF32	ADC Pulse Counter Zero Flag Register 32
BASEADDR + 0x0114	ADCn_PCZFC10	ADC Pulse Counter Zero Flag Clear Register 10
BASEADDR + 0x0116	ADCn_PCZFC32	ADC Pulse Counter Zero Flag Clear Register 32
BASEADDR + 0x0118	ADCn_PCIE10	ADC Pulse Counter Interrupt Enable Register 10
BASEADDR + 0x011A	ADCn_PCIE32	ADC Pulse Counter Interrupt Enable Register 32
BASEADDR + 0x011C	ADCn_PCIES10	ADC Pulse Counter Interrupt Enable Set Register 10
BASEADDR + 0x011E	ADCn_PCIES32	ADC Pulse Counter Interrupt Enable Set Register 32
BASEADDR + 0x0120	ADCn_PCIEC10	ADC Pulse Counter Interrupt Enable Clear Register 10
BASEADDR + 0x0122	ADCn_PCIEC32	ADC Pulse Counter Interrupt Enable Clear Register 32
BASEADDR + 0x0124	Reserved	Do not modify

6.3.6 Additional Information for Registers

6.3.6.1 A/D End Channel Setting Register (ADCn_SCH)

NOTE

Example: Channel Setting ANS = 26ch, ANE = 3ch, single conversion mode

Operation: Conversion channel 26ch → 27ch → 0ch → 1ch → 2ch → 3ch end

ANS[4] ANE[4]	ANS[3] ANE[3]	ANS[2] ANE[2]	ANS[1] ANE[1]	ANS[0] ANE[0]	Start / End Channel
0	0	0	0	0	AN0
0	0	0	0	1	AN1
0	0	0	1	0	AN2
0	0	0	1	1	AN3
...					...
1	1	0	1	1	AN27
1	1	1	0	0	reserved
1	1	1	0	1	reserved
1	1	1	1	0	reserved
1	1	1	1	1	reserved

Channels that are disabled in ADCn_ER32, ADCn_ER10:ADE[31:0] in the range of ANS and ANE will be skipped.

NOTE Mapping of the ADC channels to the registers ADCn_CC15-0 are done as follows:
ADCn_CC15~0[3:0] are mapped to **even** ADC input channels (0, 2, 4......14).
ADCn_CC15~0[7:4] are mapped to **odd** ADC input channels (1, 3, 5........15).

Examples:
Channel 5 will be mapped to ADCn_CC2[7:4] and
Channel 4 will be mapped to ADCn_CC2[3:0].

6.4 I^{2}C Interface

MB88F33x ‘Indigo2(-x)’ has two I^{2}C interfaces, which can be controlled by internal register accesses.

The I^{2}C bus is a multi-master bus. More than one device capable of controlling the bus can be connected to it. Data on the I^{2}C-bus can be transferred at a rate up to 100 kBit/s in ‘standard mode’, or up to 400 kBit/s in ‘fast mode’.

The number of interfaces connected to the bus is solely dependent on the bus capacitance limit of 400 pF. Only two bus lines are required; a serial data line (SDA) and a serial clock line (SCL). Each device connected to the bus is software addressable by a unique address and simple master/ slave relationships exist at all times; masters can operate as master-transmitters or as master-receivers. Its a true multi-master bus including collision detection and arbitration to prevent data corruption if two or more masters simultaneously initiate data transfer.

6.4.1 Features of the I^{2}C Interface

- Master transmitting and receiving functions
- Arbitration function
- Clock synchronization function
- General call addressing support
- Transfer direction detection function
- Repeated start condition generation and detection function
- Bus error detection function
- 7 bit and 10 bit addressing as master
- Up to 400 kBit/s transfer rate (I^{2}C fast mode)
- Possibility to use built-in noise filters for SDA and SCL
- Support for DMA interface.

6.4.2 Operation of the I2C Interface

The I^2C bus executes communication using two bidirectional bus lines, the serial data line (SDA) and serial clock line (SCL). The I^2C interface has two open-drain I/O pins (SDA/SCL) corresponding to these lines, enabling wired logic applications.

6.4.2.1 Start Conditions

When the bus is free (I2Cn_IBCSR:BB="0", I2Cn_IBCSR:MSS="0"), writing “1” to the I2Cn_IBCSR:MSS bit places the I^2C interface in master mode and generates a start condition.

If a “1” is written to the I2Cn_IBCSR:MSS bit while the bus is idle (I2Cn_IBCSR:MSS="0" and I2Cn_IBCSR:BB="0"), a start condition is generated and the contents of the I2Cn_IODAR:IODAR register (which should be address data) is sent.
Repeated start conditions can be generated by writing “1” to the I2Cn_IBCSR:SCC bit when in bus master mode and interrupt status (I2Cn_IBCSR:MSS="1" and I2Cn_IBCSR:INT="1").

If a “1” is written to the I2Cn_IBCSR:MSS bit while the bus is in use (I2Cn_IBCSR:BB="1" and I2Cn_IBCSR:TRX="0"; I2Cn_IBCSR:MSS="0"and I2Cn_IBCSR:INT="0"), the interface waits until the bus is free and then starts sending.

Writing "1" to the I2Cn_IBCSR:MSS bit or I2Cn_IBCSR:SCC bit in any other situation has no significance.

6.4.2.2 Stop Conditions

Writing “0” to the I2Cn_IBCSR:MSS bit generates a stop condition.

After clearing the I2Cn_IBCSR:MSS bit, the interface tries to generate a stop condition which might fail if a certain external condition causes signal transition caused from "1" to "0" at the SCL line before the generation of this stop condition. In this case, the I2Cn_IBCSR:AL bit is set to "1" and interrupt is signalled at the end of the next byte.

6.4.2.3 Addressing Slaves

In master mode, after a start condition is generated the I2Cn_IBCSR:BB and I2Cn_IBCSR:TRX bits are set to “1” and the contents of the I2Cn_IODAR:IODAR register is sent in MSB first order. After address data is sent and an acknowledge signal was received from the slave device, bit 0 of the sent data (bit 0 of the I2Cn_IODAR:IODAR register after sending) is inverted and stored in the I2Cn_IBCSR:TRX bit. Acknowledgement by the slave may be checked using the I2Cn_IBCSR:LRB bit in the I2Cn_IBCSR register. This procedure also applies to a repeated start condition.

In order to address a ten bit slave for write access, two bytes have to be sent. The first one is the ten bit address header which consists of the bit sequence "1 1 1 1 0 A9 A8 0", it is followed by the second byte containing the lower eight bits of the ten bit slave address (A7 to A0).

A ten bit slave is accessed for reading by sending the above byte sequence and generating a repeated start condition (I2Cn_IBCSR:SCC bit) followed by a ten bit address header with read access (1 1 1 1 0 A9 A8 1).

Summary of the address data bytes:

7 bit slave, write access: Start condition - A6 A5 A4 A3 A2 A1 A0 0.

7 bit slave, read access: Start condition - A6 A5 A4 A3 A2 A1 A0 1.

10 bit slave, write access: Start condition - 1 1 1 1 0 A9 A8 0 - A7 A6 A5 A4 A3 A2 A1 A0.

10 bit slave, read access: Start condition - 1 1 1 1 0 A9 A8 1 - A7 A6 A5 A4 A3 A2 A1 A0 - repeated start - 1 1 1 1 0 A9 A8 1.

6.4.2.4 Arbitration

During sending in master mode, if another master device is sending data at the same time, arbitration is performed. If a device is sending the data value "1" and the data on the SDA line has an "L" level value, the device is considered to have lost arbitration, and the I2Cn_IBCSR:AL bit is set to "1." Also, the I2Cn_IBCSR:AL bit is set to "1" if a start condition is detected at the first bit of a data byte but the interface did not want to generate one or the generation of a start or stop condition failed by some reason.

Arbitration loss detection clears both the I2Cn_IBCSR:MSS and I2Cn_IBCSR:TRX bit.

6.4.2.5 Acknowledgement

Acknowledge bits are sent from the receiver to the transmitter. The I2Cn_IBCSR:ACK bit can be used to select whether to send an acknowledgment when data bytes are received.

In master mode, acknowledgement by the slave can be checked by reading the I2Cn_IBCSR:LRB bit.

6.4.3 Programming Flow Charts

Each programming flow charts for the 400 kHz I^{2}C interface is shown below.

6.4.3.1 Programming Flow Charts

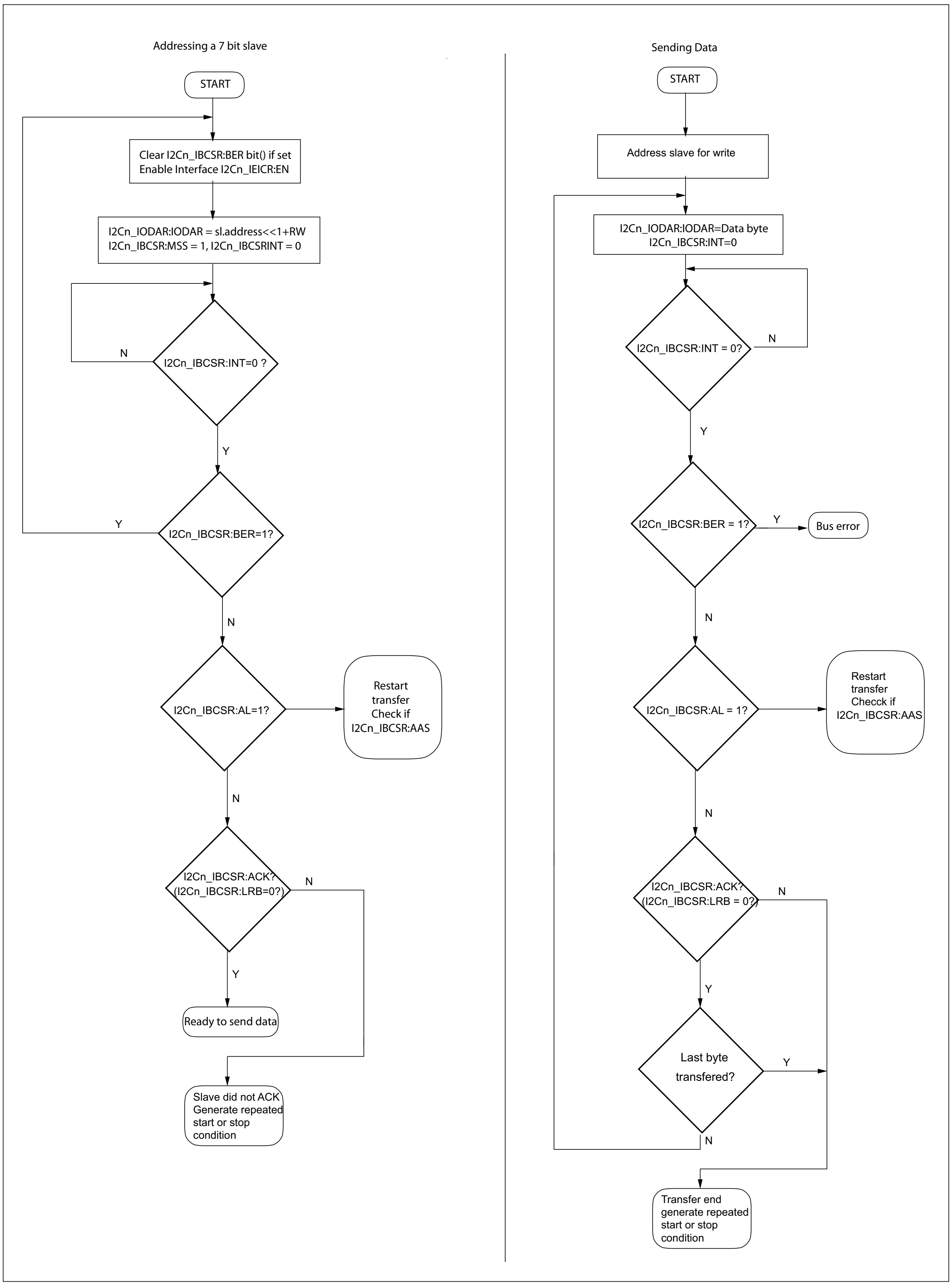

Addressing a 7 bit slave
START
Clear I2Cn_IBCSR:BER bit() if set
Enable Interface I2Cn_IEICR:EN
I2Cn_IODAR:IODAR = sl.address<<1+RW
I2Cn_IBCSR:MSS = 1, I2Cn_IBCSRINT = 0
N
I2Cn_IBCSR:INT=0 ?
Y
Y
I2Cn_IBCSR:BER=1?
N
I2Cn_IBCSR:AL=1?
Restart transfer Check if I2Cn_IBCSR:AAS
N
I2Cn_IBCSR:ACK? (I2Cn_IBCSR:LRB=0?)
N
Y
Ready to send data
Slave did not ACK Generate repeated start or stop condition
Sending Data
START
Address slave for write
I2Cn_IODAR:IODAR=Data byte
I2Cn_IBCSR:INT=0
I2Cn_IBCSR:INT = 0?
N
Y
I2Cn_IBCSR:BER = 1?
Y
Bus error
N
I2Cn_IBCSR:AL = 1?
Restart transfer Checck if I2Cn_IBCSR:AAS
N
I2Cn_IBCSR:ACK? (I2Cn_IBCSR:LRB = 0?)
N
Y
Last byte transfered?
Y
N
Transfer end generate repeated start or stop condition

Figure 6-14: Example of slave addressing and sending data

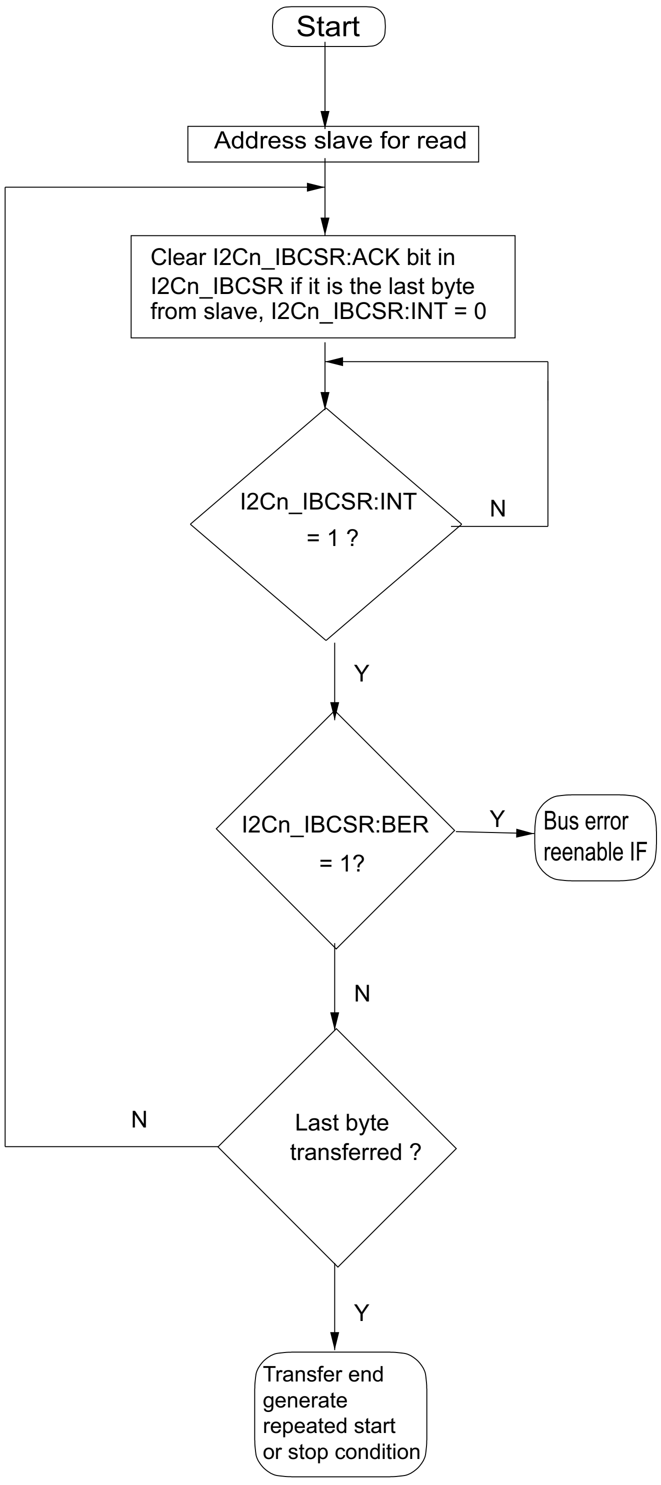

Start
Address slave for read
Clear I2Cn_IBCSR:ACK bit in I2Cn_IBCSR if it is the last byte from slave, I2Cn_IBCSR:INT = 0
I2Cn_IBCSR:INT = 1 ?
N
Y
I2Cn_IBCSR:BER = 1?
Y
Bus error reenable IF
N
N
Last byte transferred ?
Y
Transfer end generate repeated start or stop condition

Figure 6-15: Example of receiving data

6.4.4 I^{2}C Register Overview

Table 6-9: Registers Overview

Base Address(es)	Instance no 0: BASEADDR0="00094000" Instance no 1: BASEADDR1="00095000"	
Absolute Address	**Register Name**	**Register Description**
BASEADDRx + 0x0000	I2Cn_IBCSR	Bus Control and Status Register
BASEADDRx + 0x0002	Reserved	Do not modify
BASEADDRx + 0x0004	Reserved	Do not modify
BASEADDRx + 0x0006	Reserved	Do not modify
BASEADDRx + 0x0008	I2Cn_IODAR	Output Data Register
BASEADDRx + 0x0009	Reserved	Do not modify
BASEADDRx + 0x000A	I2Cn_ICCR	Clock Control Register
BASEADDRx + 0x000B	Reserved	Do not modify
BASEADDRx + 0x000C	I2Cn_ICDIDAR	CPU and DMA Input Data Register
BASEADDRx + 0x000E	I2Cn_IEICR	Interface Enable and Interrupt Clear Register
BASEADDRx + 0x0010	I2Cn_DDMACFG	DMA Configuration Register
BASEADDRx + 0x0012	I2Cn_IEIER	Error Interrupt Enable Register
BASEADDRx + 0x0013	Reserved	Do not modify

6.4.5 I²C Interface Additional Register Information

6.4.5.1 Bus Control and Status Register (I2Cn_IBCSR)

The bus control and status register (I2Cn_IBCSR) has the following functions:

- Bus busy detection
- Repeated start condition detection
- Arbitration loss detection
- Acknowledge detection
- Data transfer direction indication
- General call address detection
- Address/data detection
- Interrupt enabling flags
- Interrupt generation flag
- Bus error detection flag
- Repeated start condition generation
- General call acknowledge generation enabling
- Data byte acknowledge generation enabling

The lower byte of I2Cn_IBCSR register is read-only, all bits are controlled by the hardware. All bits (lower byte) are cleared if the interface is not enabled (I2Cn_IEICR:EN = "0").

Write access to I2Cn_IBCSR register should only occur while the I2Cn_IBCSR:INT="1" or, if a transfer is to be started. The user should not write to this register during an ongoing transfer since changes to the II2Cn_IBCSR:ACK or I2Cn_IBCSR:GCAA bits could result in bus errors. All bits in this register except the I2Cn_IBCSR:BER and the I2Cn_IBCSR:BEIE bit are cleared if the interface is not enabled (I2Cn_IEICR:EN="0").

6.4.5.1.1 SCC, MSS and INT Bit Competition

Simultaneously writing to the I2Cn_IBCSR:SCC, I2Cn_IBCSR:MSS and I2Cn_IBCSR:INT bits causes a competition to transfer the next byte, to generate a repeated start condition or to generate a stop condition. In these cases the order of priority is as follows:

- Next byte transfer and stop condition generation. When I2Cn_IBCSR:INT bit is cleared and "0" is written to the I2Cn_IBCSR:MSS bit, the I2Cn_IBCSR:MSS bit takes priority and a stop condition is generated.
- Repeated start condition generation and stop condition generation. When a “1” is written to the I2Cn_IBCSR:SCC bit and “0” to the I2Cn_IBCSR:MSS bit, the I2Cn_IBCSR:MSS bit clearing takes priority. A stop condition is generated. If specific conditions are met, the I2Cn_IBCSR:AL (arbitration lost) bit does not set the I2Cn_IBCSR:INT (interrupt) bit. Those conditions are presented in and .
  - **Case 1: When SCL and SDA signals are kept at "L"**
    The I2Cn_IBCSR:AL bit is set immediately after the I2Cn_IBCSR:MSS bit set to "1" while the I2Cn_IBCSR:BB bit is indicating "0" (no start condition is detected). However, the I2Cn_IBCSR:AL bit will not set the I2Cn_IBCSR:INT bit under this circumstance.

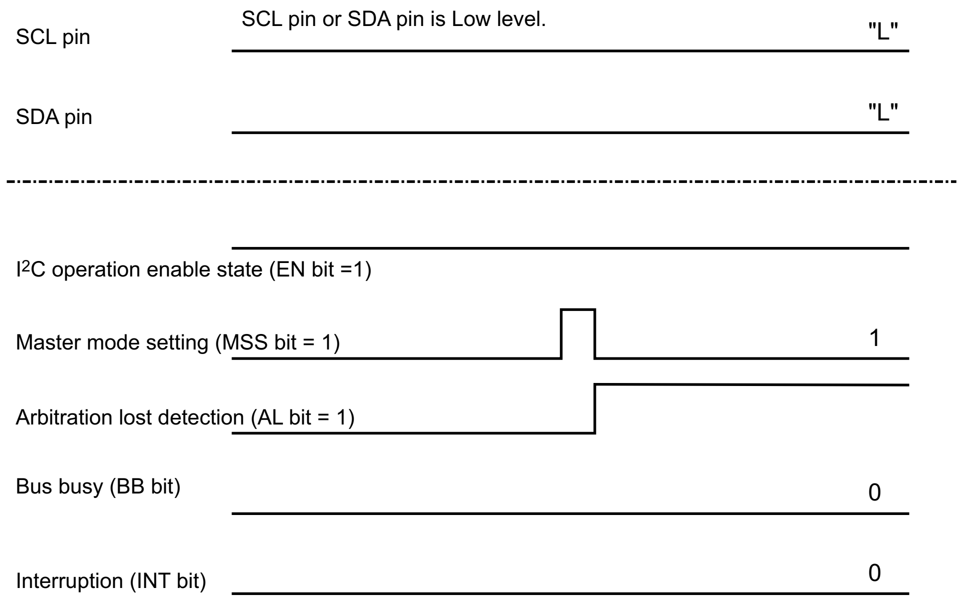

SCL pin
SCL pin or SDA pin is Low level.
"L"
SDA pin
"L"
I2C operation enable state (EN bit =1)
Master mode setting (MSS bit = 1)
1
Arbitration lost detection (AL bit = 1)
Bus busy (BB bit)
0
Interruption (INT bit)
0

Figure 6-16: Diagram of timing at which an interrupt upon detection of "AL bit = 1" does not occur

- **Case 2: When I^{2}C interface is enabled while there is ongoing communication with another bus master;**
  The interface participates in the I^{2}C bus while the bus is occupied with ongoing communication if the I2Cn_IEICR:EN bit is set from "0" to "1". In this case, the I2Cn_IBCSR:BB bit stays "0" (no start condition is detected) and setting the I2Cn_IBCSR:MSS bit to "1" results in the I2Cn_IBCSR:AL bit indicating "1". However, the I2Cn_IBCSR:AL bit will not set the I2Cn_IBCSR:INT bit under this circumstance.

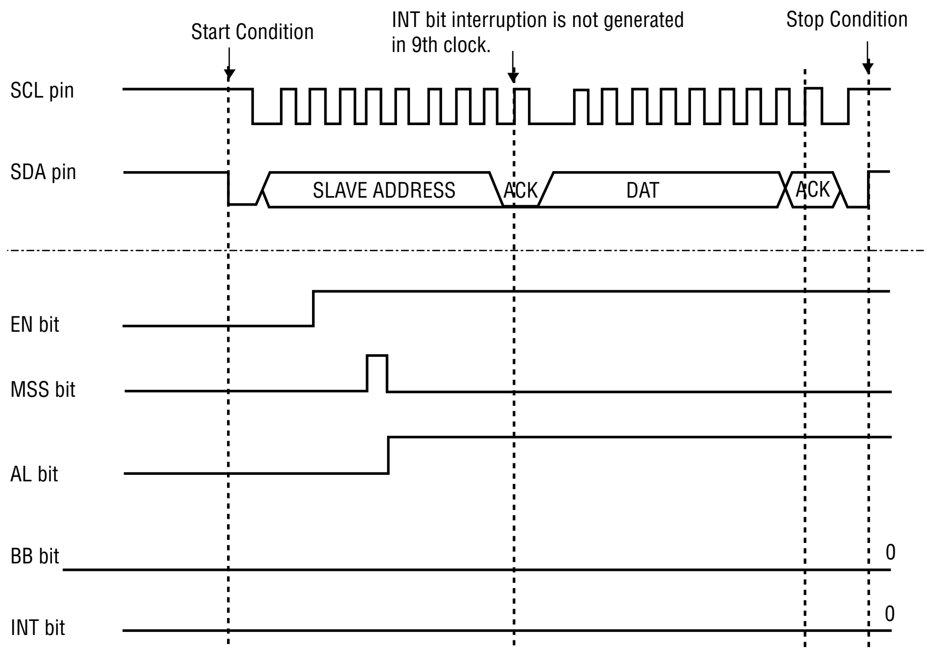


Figure 6-17: Diagram of timing at which an interrupt upon detection of "AL bit = 1" does not occur

If a symptom, as described above occurs, follow the procedure below for software processing.

1. Execute the instruction that generates a start condition (set the I2Cn_IBCSR:MSS bit to 1).
2. Use, for example, the timer function to wait for the time for three - bit data transmission at the I^{2}C transfer frequency set in the I2Cn_ICCR:CS register.*
   Example: Time for three-bit data transmission at an I^{2}C transfer frequency of 100 kHz = (1 / (100 $\times$ 10^3)) $\times$ 3 = 30 μs
3. Check the I2Cn_IBCSR:AL and I2Cn_IBCSR:BB bits and, if the I2Cn_IBCSR:AL and I2Cn_IBCSR:BB bits are 1 and 0, respectively, set the I2Cn_IEICR:EN bit to 0 to initialize I^{2}C. When the I2Cn_IBCSR:AL and I2Cn_IBCSR:BB bits are not so, perform normal processing.
   A sample flow is given below:

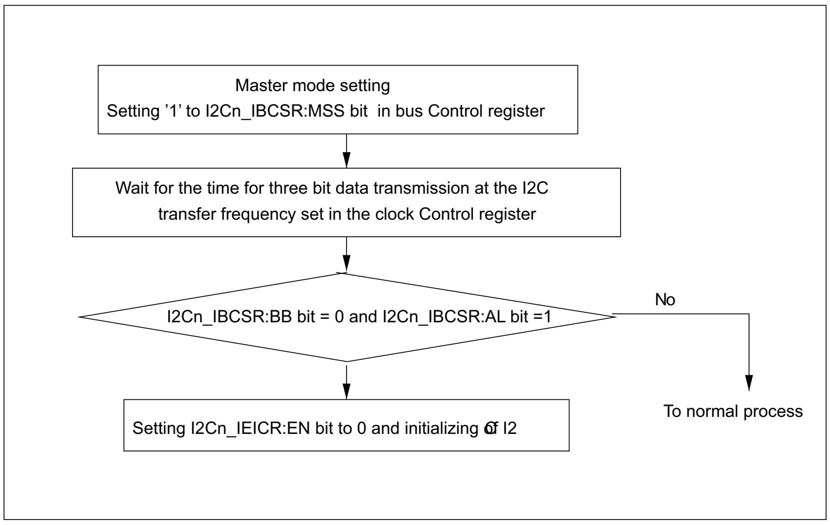

Master mode setting
Setting '1' to I2Cn_IBCSR:MSS bit in bus Control register
Wait for the time for three bit data transmission at the I2C transfer frequency set in the clock Control register
I2Cn_IBCSR:BB bit = 0 and I2Cn_IBCSR:AL bit =1
No
To normal process
Setting I2Cn_IEICR:EN bit to 0 and initializing of I2C

Figure 6-18: I2C Arbitration Flow diagram

*: Arbitration lost is detected within 3-bit time after setting the MSS bit to "1".

- Example of occurrence of an interrupt (I2Cn_IBCSR:INT bit = 1) upon detection of "I2Cn_IBCSR:AL bit = 1"

  When an instruction which generates a start condition is executed (setting the I2Cn_IBCSR:MSS bit to 1) with "bus busy" detected (I2Cn_IBCSR:BB bit = 1) and arbitration is lost, the I2Cn_IBCSR:INT bit interrupt occurs upon detection of "I2Cn_IBCSR:AL bit = 1".

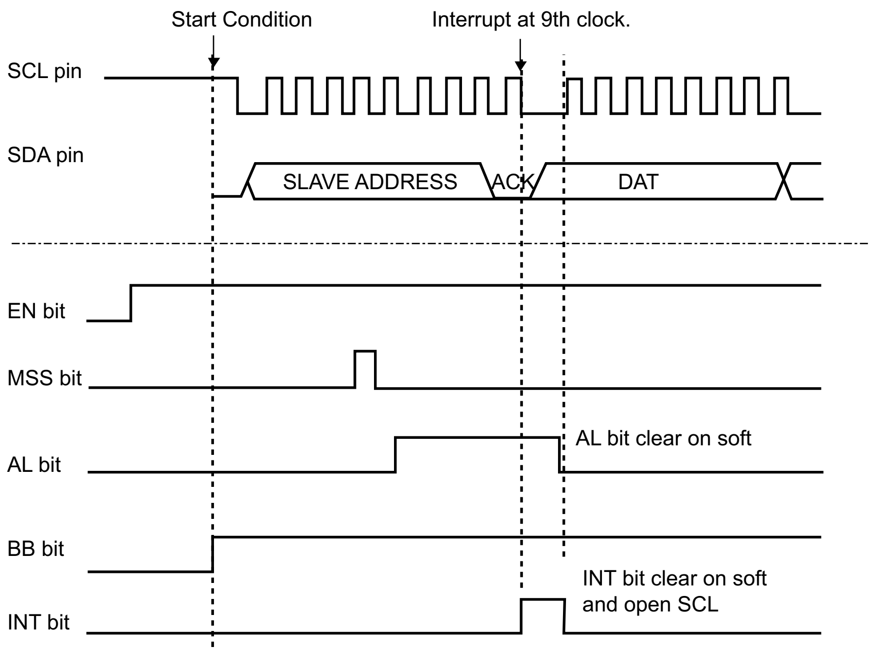


Figure 6-19: Diagram of timing at which an interrupt upon detection of "AL bit = 1" occurs

6.4.5.2 Clock Control Register (I2Cn_ICCR)

The Clock Control Register is used to configure prescaler.

6.4.5.2.1 Clock Prescaler Settings

The calculation formula for CS0 to CS5 is determined as follows:

$$\text{Bitrate} = \frac{\phi}{n \times 12 + 16}$$

n > 0
ϕ: RBUS clock, Noise filter disabled

$$\text{Bitrate} = \frac{\phi}{n \times 12 + (16 + y)}$$

n > 0
ϕ: RBUS clock, Noise filter enabled
y: Varies from 1 to 6, according to noise filter configuration

Table 6-10: Prescaler settings:

n	CS5	CS4	CS3	CS2	CS1	CS0
1	0	0	0	0	0	1
2	0	0	0	0	1	0
3	0	0	0	0	1	1
...						
63	1	1	1	1	1	1
Note: Do not use n=0 prescaler setting, it violates SDA/SCL timings.						

6.4.5.2.2 Common Peripheral Clock Frequencies

Table 6-11 shows the most common peripheral clock frequencies with their prescaler settings and the resulting sending bit rate:

Table 6-11: Common peripheral clock frequencies

RBUS Clock frequency [MHz]	100 Kbit (Noise filter disabled)		400 Kbit (Noise filter enabled) (y=1)	
	n	Bit rate [Kbit]	n	Bit rate [Kbit]
40	32	100	7	396.04
36	29	98.9	7	356.44
32	26	97.56	6	359.55
28	22	100	5	363.64
24	19	98.36	4	369.23
20	16	96.15	3	377.36
16	12	100	2	390.24
12	9	96.77	2	292.68
8	6	90.91	1	275.86

6.4.5.3 DMA Configuration Register (I2Cn_DDMACFG)

The DMA Configuration Register (I2Cn_DDMACFG) contains bits to enable/disable DMA request.

NOTE The request generation behavior is explained in details for different modes of operation:

1. Master mode as transmitter - the DMA_REQ_TX is generated after completing the transmission of data (address or data).
2. Master mode as receiver - the DMA_REQ_TX is generated after completing the transmission of address, however this is not required so it is suggested not to enable the DMA for transmission. The DMA_REQ_RX is generated after completing the reception of data.

6.5 Sound Generator

The Sound Generator unit is an advanced PWM-based Sound Generator for sound/melody generation. Its main feature is that frequency and amplitude can be increased or decreased automatically without additional interrupts. It supports both linear and exponential attack/decay of the sound. It can play warning sounds, sounds generated when pressing keys and melodies. The features are better implemented than a simple PPG/PWM, as it reduces the number of interrupts required.

6.5.1 Features of the Sound Generator

- Flexibility to program the Sound Generator depending on the application and ability to produce sound/melody with varying frequency and amplitude for the convenient duration
- Sound output is a square wave of 100 Hz - 6 KHz (at resolution of better than 20 Hz at input frequency of 16 MHz)
- PWM cycle width can be programmed to either 255 or 511 clocks and the duty cycle (i.e. amplitude) can be programmed in the range from 0% to 100%
- Frequency and amplitude counters driven by programmable pre scalers with clock division of /1, /2, /3, or /4
- Automatic linear or exponential amplitude increment or decrement without additional interrupts
- START/STOP/RESUME functionality to start, stop, and resume sound generation without reloading the configuration
- Supports automatic stop of sound output when amplitude becomes ‘0’
- Dedicated sequencer to support optimized DMA data transfer
- Programmable interrupt, DMA request, and Register Reload generation at the end of tone pulse counter, amplitude match condition and zero-amplitude condition

6.5.2 Block Diagram

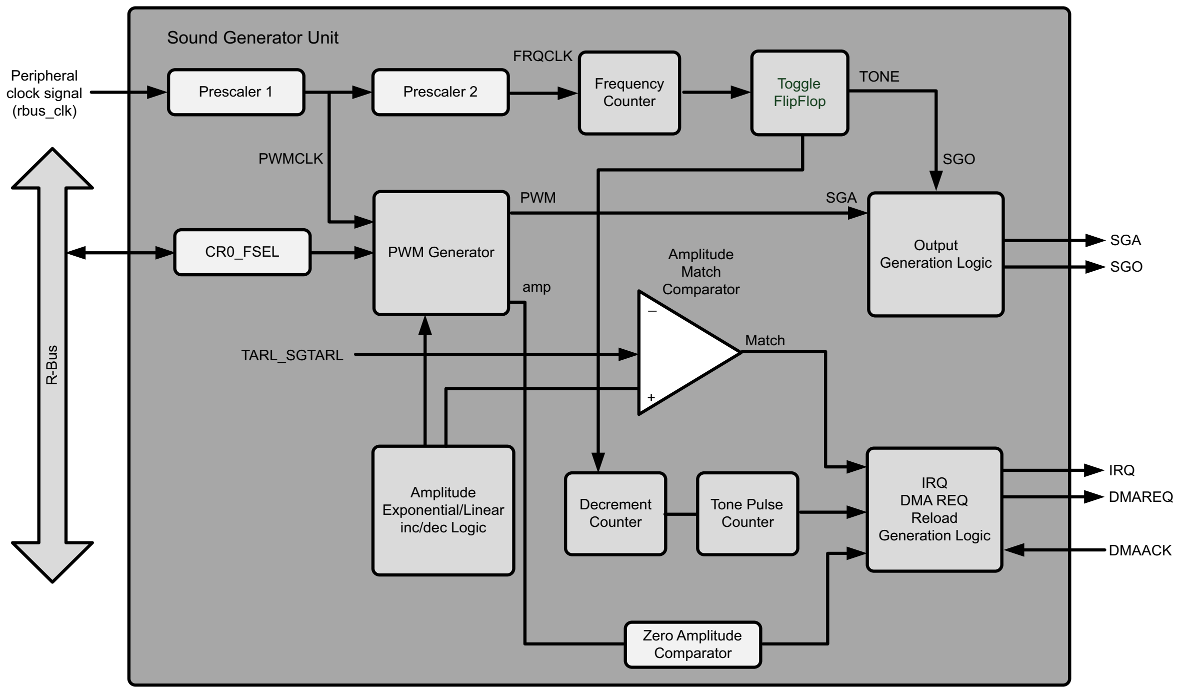


Figure 6-20: Block diagram

6.5.3 Operation of the Sound Generator

Following section describes the function of the various blocks in the Sound Generator.

6.5.3.1 PWM Generation

- The Sound Generator generates a PWM signal with a programmable period and duty cycle between: 0% (permanently low) and 100% (permanently high)
- The PWM cycle can contain 255 or 511 input clock cycles via the SGn_CR0:FSEL bit
- The PWM duty cycle configured in SGARL is incremented/decremented linearly/exponentially based on SGn_ECRL:AUTO, SGn_ECRL:IDS, and SGn_ECRL:ELS bit settings as per the equations below:
  - Linear: amp = amp $\pm$ SGn_IDRL value
  - Exponential: amp = amp $\pm$ (amp/32)

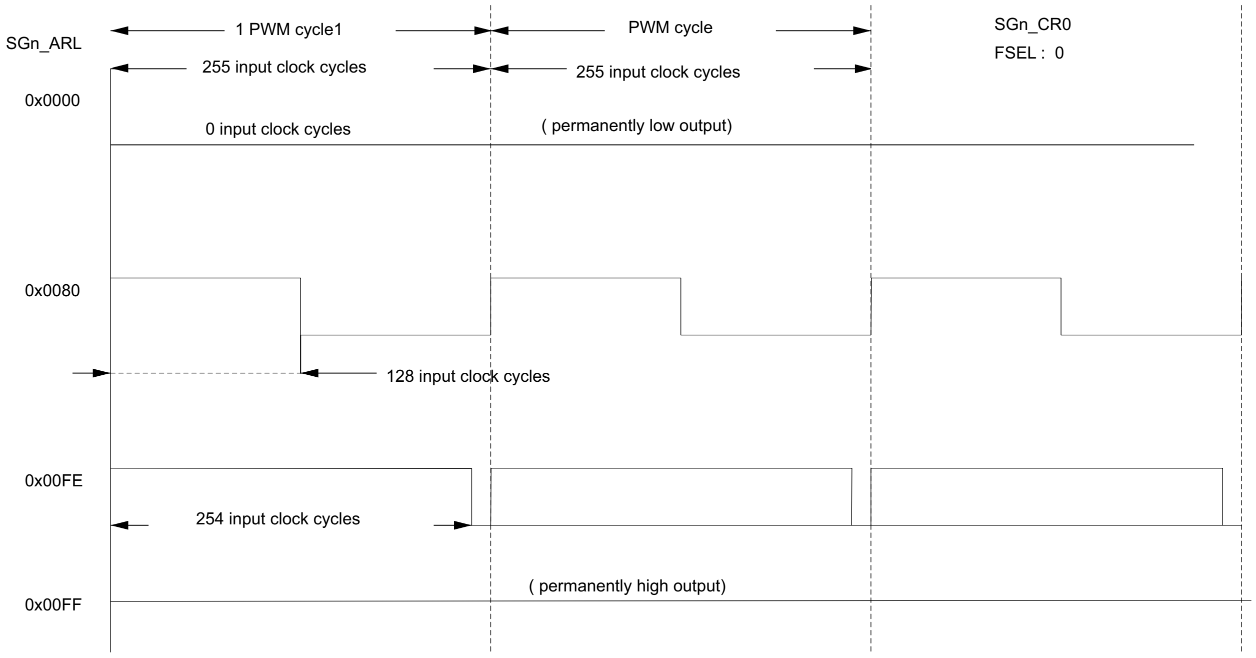


Figure 6-21: PWM pulse cycle timing details

6.5.3.2 Frequency Generation

- The Sound Generator generates a tone or frequency (i.e. SGO) of 100 Hz to 6 kHz with an incremental step of 20 Hz.
- The Sound Generator has a 15-bit frequency counter and a toggle flip-flop to generate the tone output of the previously mentioned frequency.
- The tone output (i.e. output of toggle flop) toggles after every (frequency data reload register value + 1) as shown in Figure 6-22: as follows.

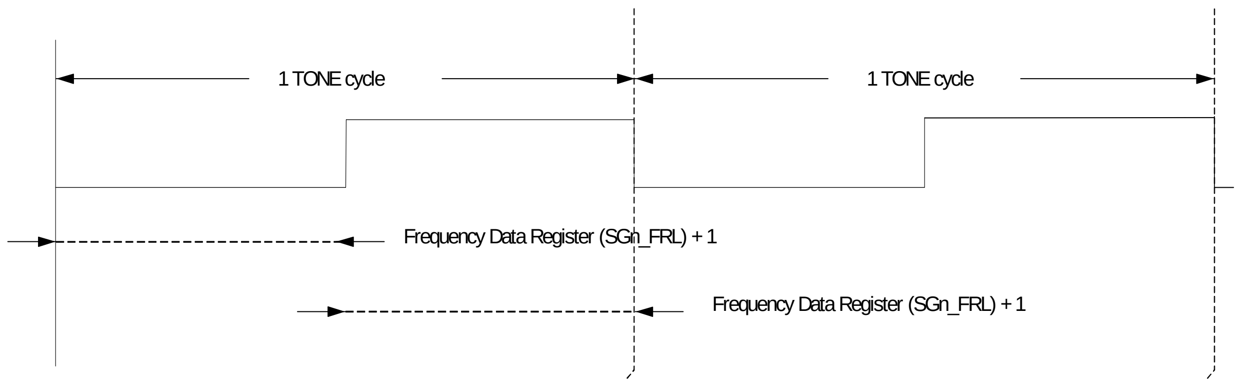

1 TONE cycle
1 TONE cycle
Frequency Data Register (SGn_FRL) + 1
Frequency Data Register (SGn_FRL) + 1

Figure 6-22: Tone pulse timing details

6.5.3.3 Interrupt, DMA Request, and Reload Generation

- The Sound Generator generates a CPU interrupt, DMA request and register reload condition according to the following conditions:
  - When the Reload Timer i.e tone pulse counter (SGn_NRL) reaches zero.
  - When the amplitude register value matches the target amplitude configured in SGn_TARL.
  - When the amplitude register in PWM generator (i.e sound amplitude) value becomes '0'.
- DMA request and register reload condition are generated during Sound Generator start operation also, apart from the above three conditions.
- Interrupt, DMA request, and register reload condition generation can be enabled or disabled by programming respective enable bits in SGn_ECRL register. Refer to Sound Generator Extended Control Reload Register (SGn_ECRL) description for interrupt enable/disable functionality.
- Refer to Figure 6-23 and Figure 6-24 in the following section to show IRQ interrupt generation logic and DMA request generation respectively.
- If the target amplitude is set to '0' and the amplitude match condition is used for reload, then the zero amplitude event may not be seen because the amplitude register is reloaded before it goes to 0. Thus it is recommended to use the same condition for interrupt or DMA request that is also used for reload.

6.5.3.4 Register Reload Operation

To support synchronized, on-the-fly reloading of the registers, the Sound Generator uses a shadow register concept.

Each reload register of the Sound Generator has a corresponding shadow register. The CPU or DMA can therefore write into the reload register at any time, but copying of the reload register to shadow register is done on a reload event. Refer to Figure 6-25

The loading of the shadow registers to respective counters is synchronized by the one pulse.

A list of the Sound Generator Reload registers with shadow registers is shown below:

1. SGn_ECRL
2. SGn_FRL
3. SGn_ARL
4. SGn_TARL
5. SGn_TCRL
6. SGn_IDRL

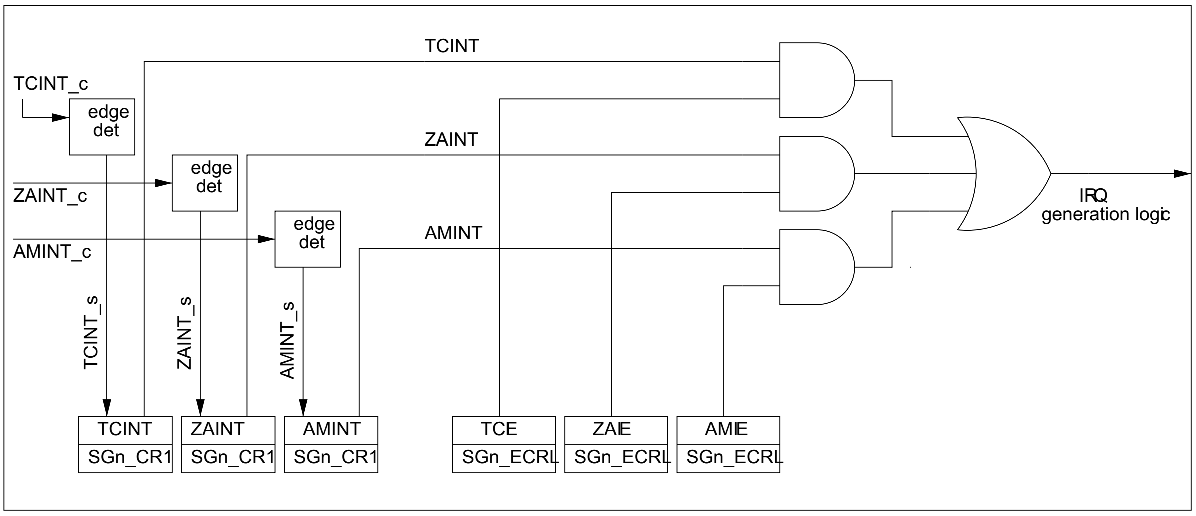


Figure 6-23: Interrupt request generation logic

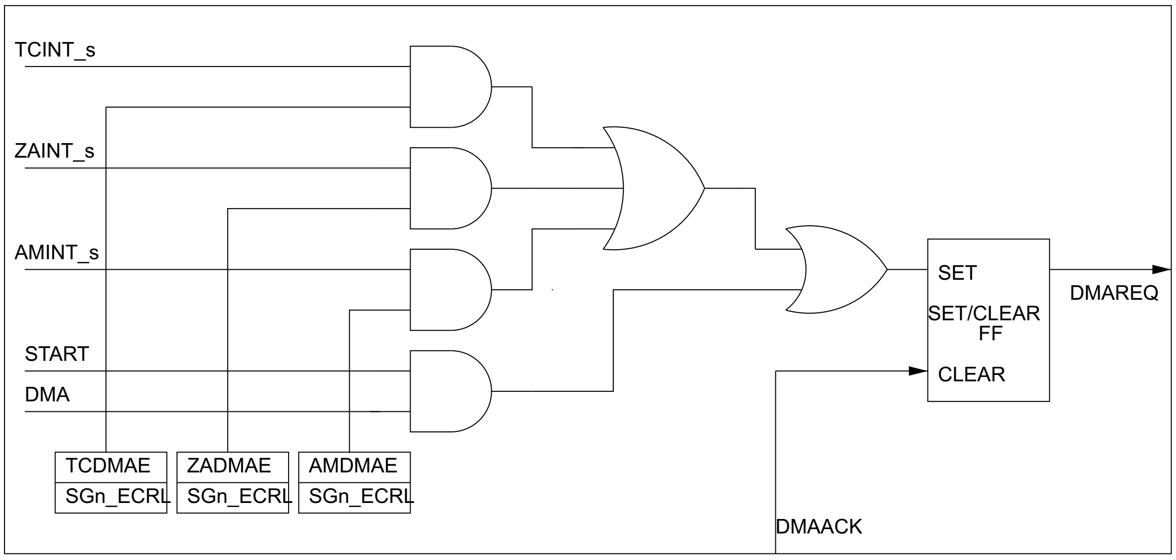


Figure 6-24: DMA request generation logic

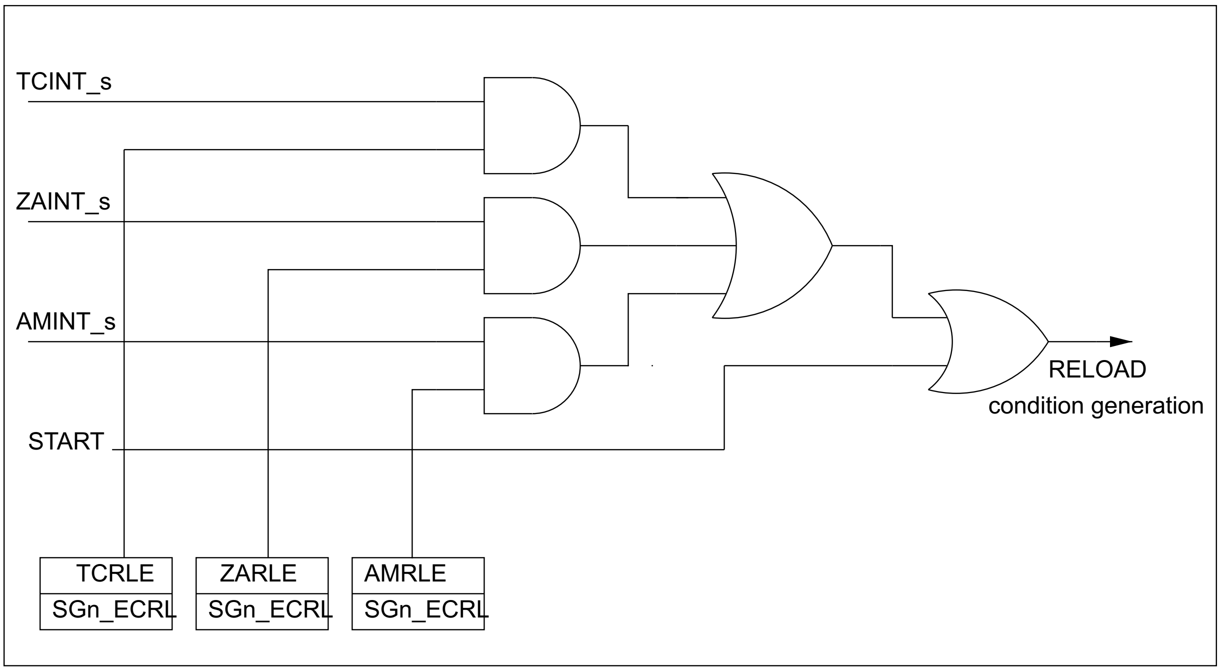


Figure 6-25: Register reload generation logic

6.5.3.5 Sound Generator Output Generation Logic

The Sound Generator output generation logic generates the SGA (amplitude), SGO (tone), mixed tone, and mixed PWM outputs.

- The Sound Generator output signals are multiplexed through the SGO and SGA outputs with SGn_CR1:TONE and AMP bits as shown in Figure 6-26
- All outputs are disabled when the Sound Generator is in stop mode.

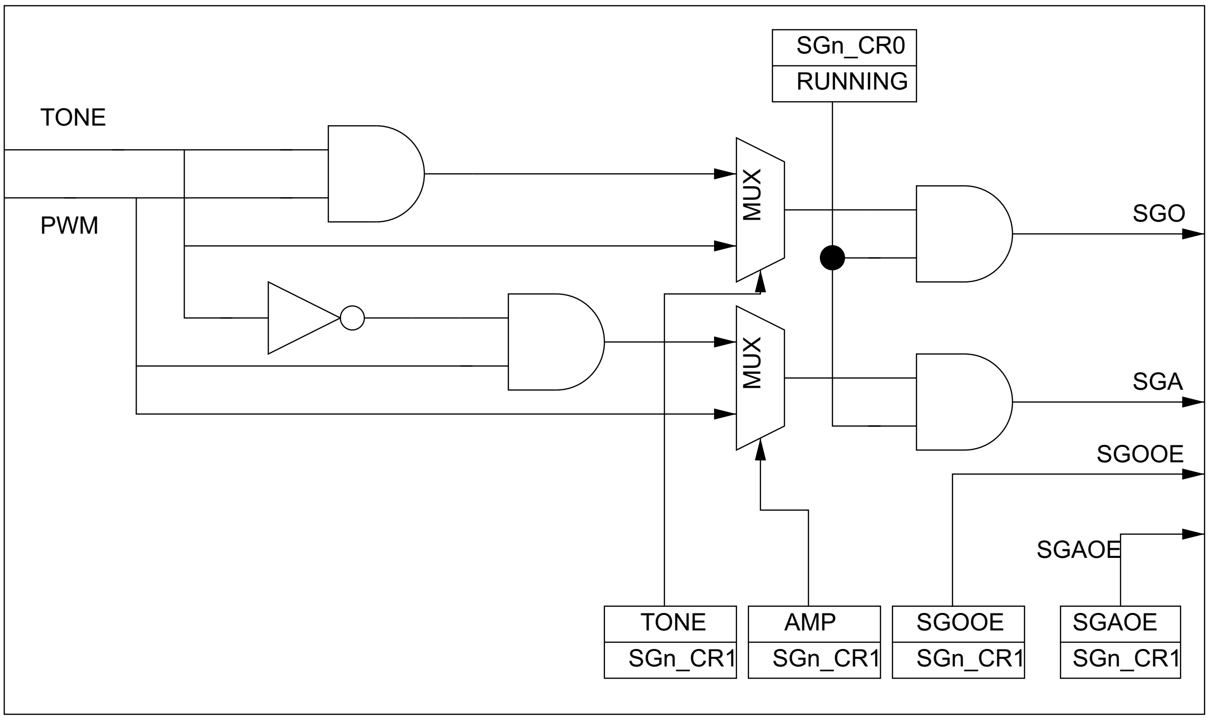


Figure 6-26: Output generation logic

6.5.3.6 Sound Generator Mode Control Logic

The Sound Generator has basically two modes of operation: stop and running mode.

Stop mode

This mode is entered either by setting the SGn_CR0:STOP bit or when the amplitude register value reaches ’0’. Sound Generator operation is stopped by disabling the sound outputs (SGA and SGO). Sound Generator operation can be restarted by reloading the configuration after setting the SGn_CR0:STARTbit or by setting SGn_CRO:RESUME to resume sound generation when it was paused.

Running mode

In this mode, the Sound Generator will be in the normal mode of operation with sound frequency and amplitude generated on the SGO and SGA outputs respectively.

6.5.3.7 DMA-based Sound Generator Register Update Operation

This section describes the relationship between the DMA Transfer Update Enable register (SGn_DER) and the DMA Transfer Indirect Register (SGn_DMAR). It also provides a brief overview of:

- The number of times of DMA transfer

- DMA transfer size
- Transfer byte positions

The Number of Times of DMA Transfer

The number of DMA transfers depends on the setting of the DMA Transfer Update Enable Register (SGn_DER), as given in below equation:

Number of DMA transfers = Number of ’1’s in SGn_DER/2 (i.e rounded up to next integer number)

Examples:

1. SGn_DER:CRE0 and SGn_DER:CRE1 bits are set: Number of DMA transfer = 2/2 = 1
2. SGn_DER:CRE0, SGn_DER:ARE0 and SGn_DER:FRE0 bits are set:
   Number of DMA transfer = 3/2 = 2 (after rounding up)
3. All the bits of SGn_DER are set: Number of DMA transfer = 11/2 = 6 (after rounding off)

DMA Transfer Size

One DMA transfer size is always a half-word (i.e. 2 bytes) irrespective of the DMA Transfer Update Enable Register (SGn_DER) settings.

Transfer Byte Position in the DMA Transfer Indirect Register

The DMA transfer byte position in the DMA Transfer Indirect Register (SGn_DMAR) depends on the setting of the DMA Transfer Update Enable Register (SGn_DER). The DMA transfer byte position is always right-aligned for some of the exemplary settings of SGn_DER register, as shown in Table 6-12 .

Table 6-12: Example SGn_DER settings and DMA transfer order settings

NRE	TCRE	IDRE	TARE1	TARE0	ARE1	ARE0	FRE1	FRE0	CRE1	CRE0	SGn_DMAR [15: 8]	SGn_DMAR [7:0]	No. of Transfers
									1	1	SGn_ECRL (15:8)	SGn_ECRL (7:0)	1
					1	1					SGn_ARL (15:8)	SGn_ARL (7:0)	1
								1		1	SGn_FRL (7:0)	SGn_ECRL (7:0)	1
						1	1		1	1	SGn_ECRL (15:8) 1st-transfer	SGn_ECRL (7:0) 1st-transfer	2
											SGn_ARL (7:0) 2nd-transfer	SGn_FRL(15:8) 2nd-transfer	
			1	1	1	1	1	1	1	1	SGn_ECRL (15:8) 1st-transfer	SGn_ECRL (7:0) 1st-transfer	4
											SGn_FRL (15:8) 2nd-transfer	SGn_FRL (7:0) 2nd-transfer	
											SGn_ARL (15:8) 3rd-transfer	SGn_ARL (7:0) 3rd-transfer	
											SGn_TARL (15:8) 4th-transfer	SGn_TARL (7:0) 4th-transfer	

6.5.3.8 DMA Transfer Flowchart

The state machine shown below shows the SG register update through DMA. The DMA transfer flow is shown for SGn_ECRL and SGn_FRL registers update and the state machine follows the sequence for the register update for the rest of the registers, based on respective enable bit settings in the SGn_DER registers.

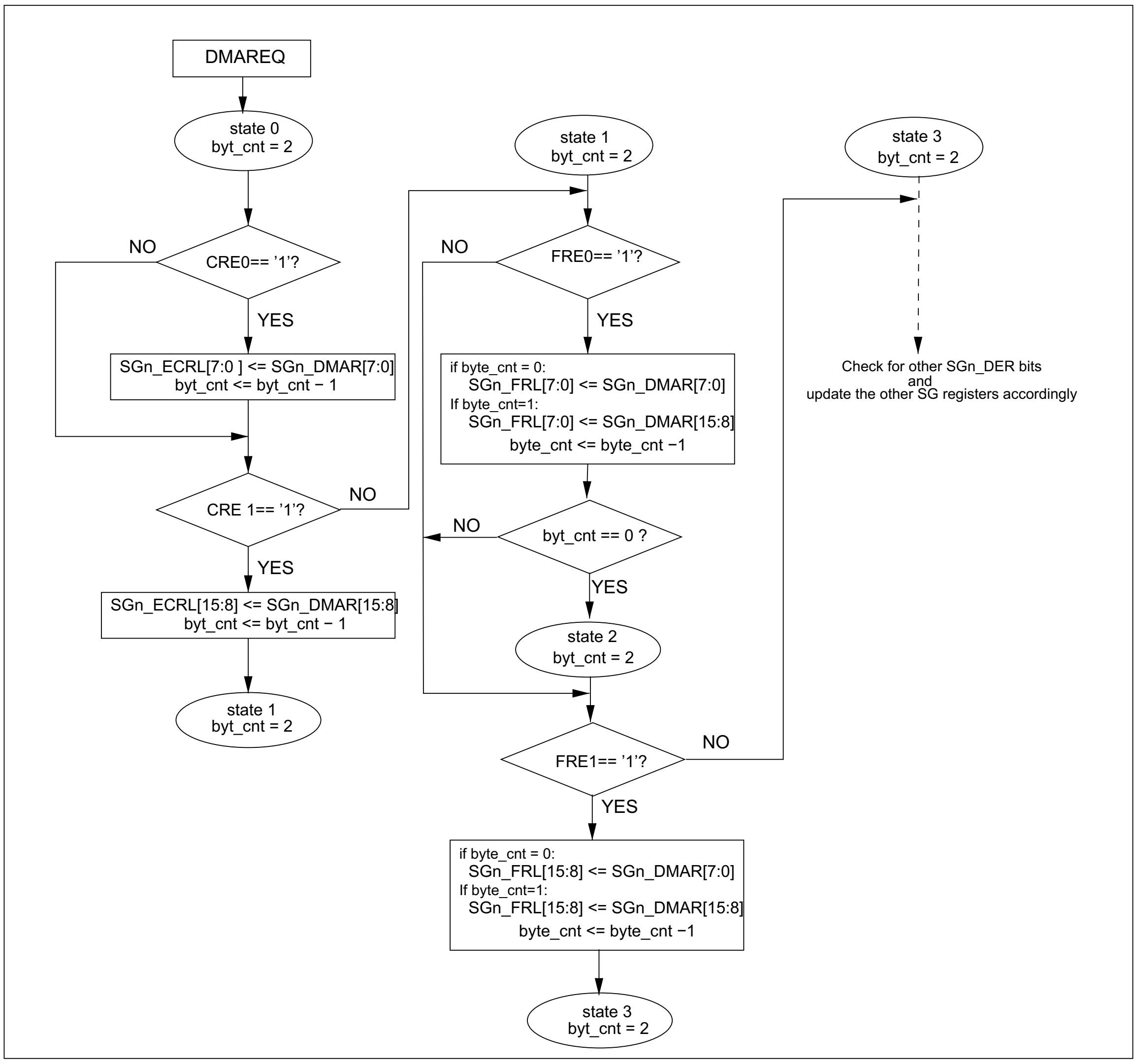


Figure 6-27: DMA-based Sound Generator register’s update FSM

6.5.3.9 Programming the Sound Generator Module

The Sound Generator can be programmed by the CPU for sound generation in a single or in a continuous operation mode depending on the sound output required. Single and continuous operation can be operated by the CPU, by DMA, or by both depending on the application. A DMA request can be used to update the Control and Data Registers through DMA transfer.

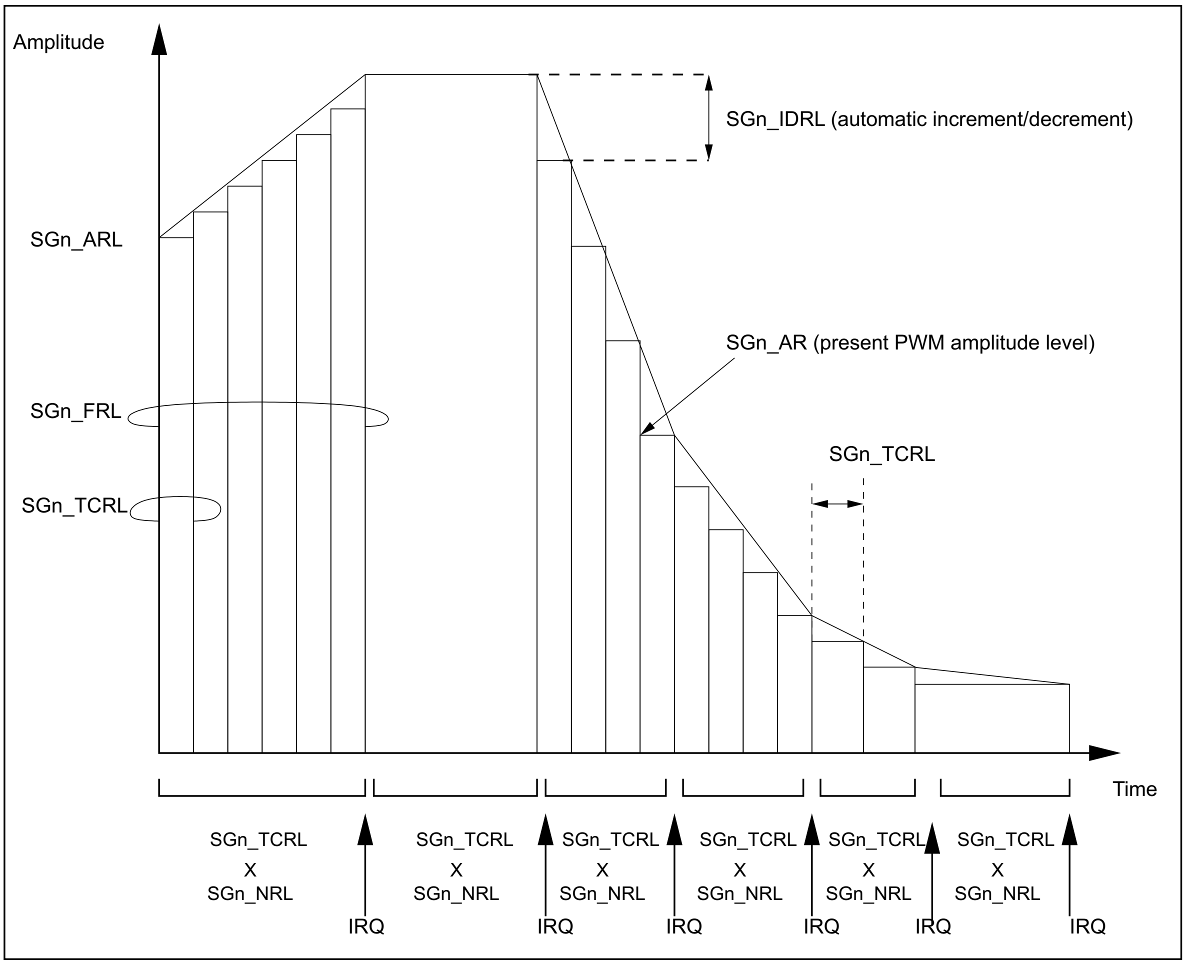


Figure 6-28: Conceptual Sound Generation operation

6.5.3.10 Using the CPU to Control Sound Generator Operation

The following steps can be followed to control the operation of the Sound Generator:

1. Program the Sound Generator control registers (SGn_CR0, SGn_CR1) by software first. Next, program all the reload registers; the amplitude data reload register (SGn_ARL), the frequency data reload register (SGn_FRL), the tone output number reload register (SGn_NRL), the time cycle and increment or decrement data reload register (SGn_TCRLIDRL), and the target amplitude data reload register (SGn_TARL) by software.
2. Set the Start bit (SGn_CR0:START) to ‘1’, which in turn generates a ‘register reload event’ and an update of the shadow registers from the reload registers takes place.
3. The SGO and SGA output of the Sound Generator starts.
4. The CPU reprograms the necessary registers of the Sound Generator.
5. Depending on the AMINT interrupt enable/disable SGn_ECRL:AMIE bit for interrupt generation, an interrupt is generated when the amplitude register value matches the target amplitude register, SGn_TARL. Sound generation begins with the next cycle when the Sound Generator reloads new configuration data into its counters from the shadow registers.
6. The CPU clears the interrupt bit.
7. The CPU reprograms the necessary registers of the Sound Generator.
8. Depending on the TCINT interrupt enable/disable SGn_ECRL:TCIE bit for interrupt generation, an interrupt is generated when the tone counter value matches the tone pulse output number register, SGn_NRL. Sound generation starts on the next cycle, when the Sound Generator reloads new configuration data into its counters from the shadow registers.
9. CPU clears the interrupt bit.
10. CPU programs the necessary registers of Sound Generator.
11. Depending on the ZAINT interrupt enable/disable SGn_ECRL:ZAIE bit for interrupt generation, an interrupt is generated when the amplitude value reaches zero. Sound generation starts on the next cycle, when the Sound Generator reloads new configuration data into its counters from the shadow registers.
12. When a ZERO amplitude value is reached in the amplitude register, sound generation is stopped.
13. The CPU clears the interrupt bit.
14. The Sound Generator is ready to be programmed for new sound wave generation.
15. The CPU can program SGn_CR0:STOP = ’1’ bit at any time during the above steps and this will stop sound generation immediately.

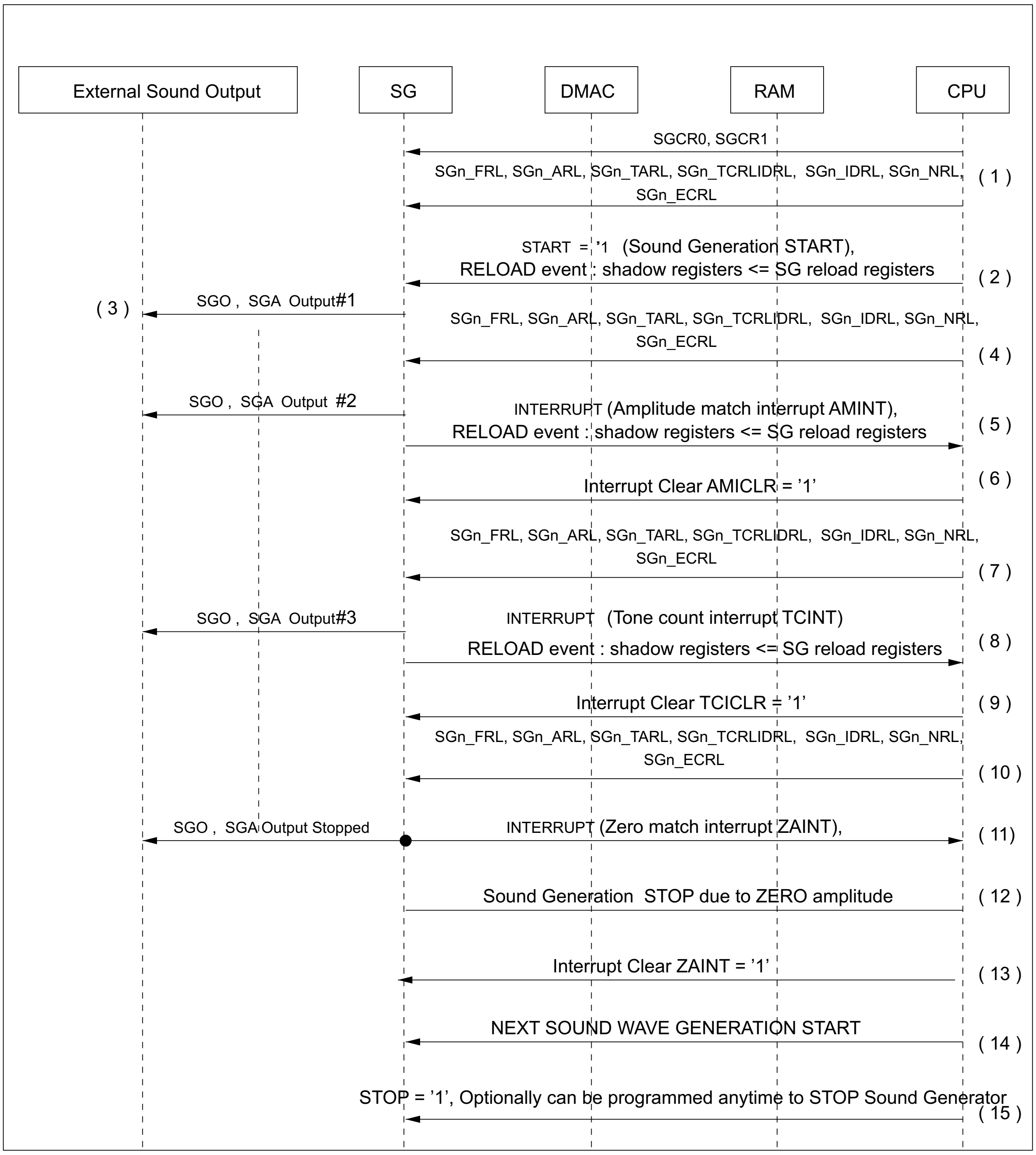

External Sound Output
SG
DMAC
RAM
CPU
SGCR0, SGCR1
SGn_FRL, SGn_ARL, SGn_TARL, SGn_TCRLIDRL, SGn_IDRL, SGn_NRL, SGn_ECRL
(1)
START = '1 (Sound Generation START),
RELOAD event : shadow registers <= SG reload registers
(2)
(3)
SGO , SGA Output#1
SGn_FRL, SGn_ARL, SGn_TARL, SGn_TCRLIDRL, SGn_IDRL, SGn_NRL, SGn_ECRL
(4)
SGO , SGA Output #2
INTERRUPT (Amplitude match interrupt AMINT),
RELOAD event : shadow registers <= SG reload registers
(5)
Interrupt Clear AMICLR = '1'
(6)
SGn_FRL, SGn_ARL, SGn_TARL, SGn_TCRLIDRL, SGn_IDRL, SGn_NRL, SGn_ECRL
(7)
SGO , SGA Output#3
INTERRUPT (Tone count interrupt TCINT)
RELOAD event : shadow registers <= SG reload registers
(8)
Interrupt Clear TCICLR = '1'
(9)
SGn_FRL, SGn_ARL, SGn_TARL, SGn_TCRLIDRL, SGn_IDRL, SGn_NRL, SGn_ECRL
(10)
SGO , SGA Output Stopped
INTERRUPT (Zero match interrupt ZAINT),
(11)
Sound Generation STOP due to ZERO amplitude
(12)
Interrupt Clear ZAINT = '1'
(13)
NEXT SOUND WAVE GENERATION START
(14)
STOP = '1', Optionally can be programmed anytime to STOP Sound Generator
(15)

Figure 6-29: Sound Generation by CPU control flow diagram

6.5.3.11 Using DMA to Control Sound Generator Operation

The following steps show how to control the operation of the Sound Generator by DMA:

1. Program the Sound Generator control registers (SGn_CR0, SGn_CR1) by software first. The SGn_CR0:DMA bit has to be set to’1’ to trigger a DMA request on a START condition. In the second step, program all the reload registers; amplitude data reload register (SGn_ARL), the frequency data reload register (SGn_FRL), the tone output number reload register (SGn_NRL), the time cycle and increment/decrement data reload register (SGn_TCRLIDRL), the target amplitude data reload register (SGn_TARL) by software.
2. Set the start bit (SGn_CR0:START) to ‘1’, which in turn generates a ‘register reload event’ and start an update of shadow registers from the reload registers. This also triggers a DMA request.
3. The SGO and SGA outputs of the Sound Generator start.
4. DMAREQ is generated due to START = ’1’.
5. DMAREQ is cleared when DMAACK = ’1’ from the DMA controller.
6. The DMA controller updates the reload registers depending on the settings in SGn_DER register.
7. Depending on the DMA amplitude match interrupt enable/disable SGn_ECRL:AMDMAE bit for DMA request generation, DMAREQ =’1’ is generated when the amplitude register value matches the target amplitude reload register, SGn_TARL. Sound generation starts on the next cycle, when the Sound Generator reloads new configuration data into its counters from the shadow registers.
8. DMAREQ is cleared when DMAACK = ’1’ from DMA controller.
9. The DMA controller updates the reload registers depending on the settings in SGn_DER register.
10. Depending on the DMA tone count interrupt enable/disable SGn_ECRL:TCDMAE bit for DMA request generation, DMAREQ = ’1’ is generated when the tone counter value matches the SGn_ECRL tone pulse output number reload register, SGn_NRL.SGn_FRLSGn_DMAR
11. DMAREQ is cleared when DMAACK = ’1’ from the DMA controller.
12. The DMA controller updates the reload registers depending on the settings in SGn_DER register.
13. Depending on the DMA zero interrupt enable/disable SGn_ECRL:ZADMAE bit for DMA request generation, a DMA request is generated when the amplitude register value reaches zero.
14. When a ZERO amplitude value is reached in the amplitude register, sound generation is stopped.
15. DMAREQ is cleared when DMAACK = ’1’ from DMA controller.
16. The Sound Generator is ready to be programmed for new sound wave generation.
17. The CPU can program SGn_CR0:STOP = ’1’ bit at any time during the above steps and this will stop sound generation immediately.

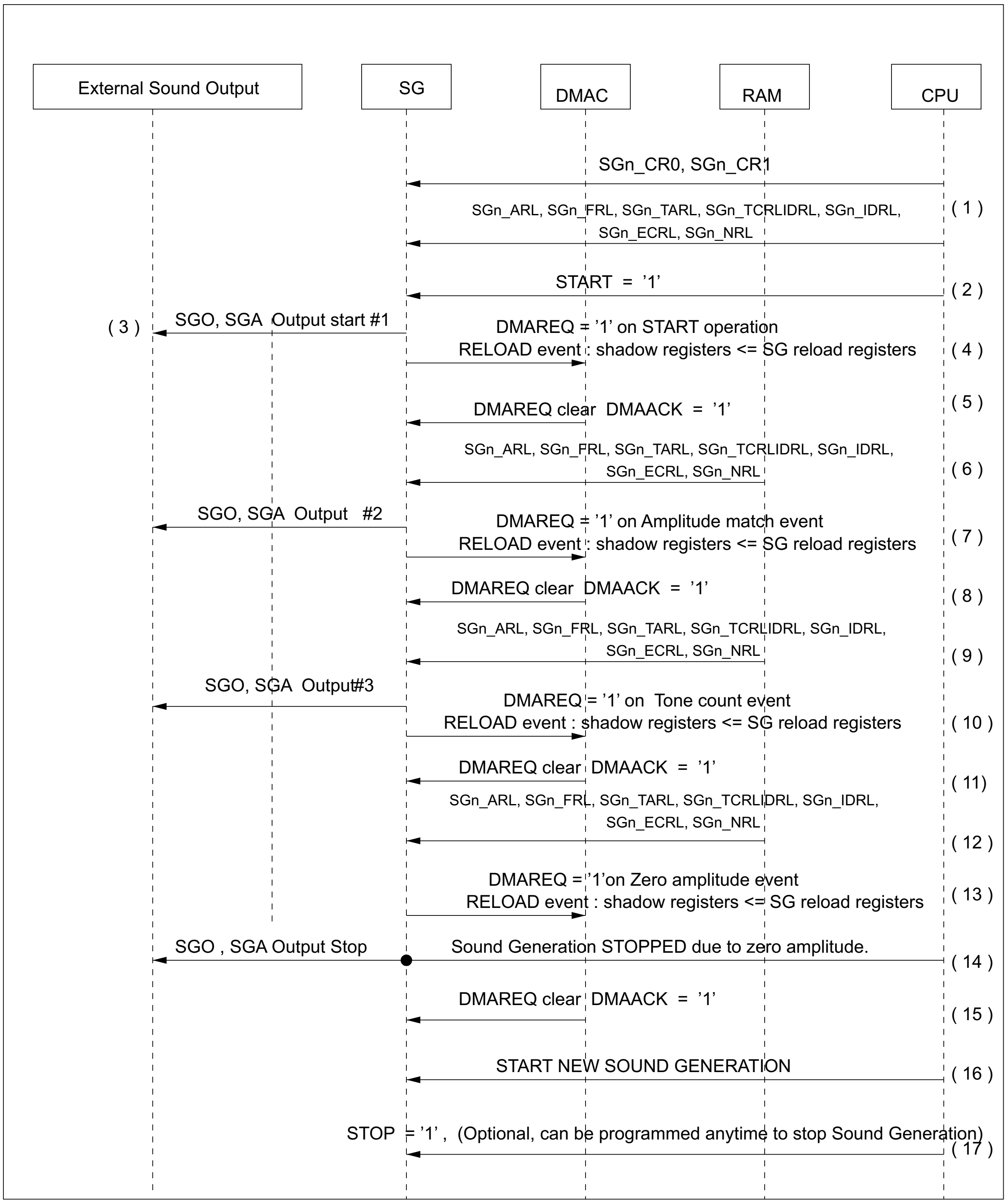

External Sound Output
SG
DMAC
RAM
CPU
SGn_CR0, SGn_CR1
SGn_ARL, SGn_FRL, SGn_TARL, SGn_TCRLIDRL, SGn_IDRL, SGn_ECRL, SGn_NRL
(1)
START = ’1’
(2)
(3)
SGO, SGA Output start #1
DMAREQ = ’1’ on START operation
RELOAD event : shadow registers <= SG reload registers
(4)
DMAREQ clear DMAACK = ’1’
(5)
SGn_ARL, SGn_FRL, SGn_TARL, SGn_TCRLIDRL, SGn_IDRL, SGn_ECRL, SGn_NRL
(6)
SGO, SGA Output #2
DMAREQ = ’1’ on Amplitude match event
RELOAD event : shadow registers <= SG reload registers
(7)
DMAREQ clear DMAACK = ’1’
(8)
SGn_ARL, SGn_FRL, SGn_TARL, SGn_TCRLIDRL, SGn_IDRL, SGn_ECRL, SGn_NRL
(9)
SGO, SGA Output#3
DMAREQ = ’1’ on Tone count event
RELOAD event : shadow registers <= SG reload registers
(10)
DMAREQ clear DMAACK = ’1’
(11)
SGn_ARL, SGn_FRL, SGn_TARL, SGn_TCRLIDRL, SGn_IDRL, SGn_ECRL, SGn_NRL
(12)
DMAREQ = ’1’on Zero amplitude event
RELOAD event : shadow registers <= SG reload registers
(13)
SGO , SGA Output Stop
Sound Generation STOPPED due to zero amplitude.
(14)
DMAREQ clear DMAACK = ’1’
(15)
START NEW SOUND GENERATION
(16)
STOP = ’1’ , (Optional, can be programmed anytime to stop Sound Generation)
(17)

Figure 6-30: Sound Generation, DMA register update flow diagram

6.5.3.12 Sound Generator Operation (Timing)

1. The reload values are written to the amplitude data reload register (SGn_ARL), the frequency data register (SGn_FRL), the tone output number reload register (SGn_NRL), and the time cycle data reload register and data increment/decrement data reload register (SGn_TCRLIDRL) by software. Initialize the interrupt status bits (SGn_ECRL:TCINT, AMINT & ZAINT) and set the interrupt enable bits (SGn_ECRL:TCIE, AMIE and ZAIE).
2. Set the start bit (SGn_CR0:START) to ‘1’. Setting START bit to ’1’ also sets SGn_CR0:RUNNING bit to ’1’, therefore indicating SG is running.
3. By setting ‘1’ to the start bit (SGn_CR0:START), the amplitude data register (SGn_ARL) value is loaded into the PWM pulse generator, the frequency data register (SGn_FRL) value into the frequency counter, the tone output number register (SGn_NRL) value into the tone pulse counter, the time cycle register (SGn_TCRL) value into the decrement counter.
4. Step 4 is a part of step 3.
5. Counter decrements occur on the negative edge of the tone pulse. An overflow in the decrement counter enables a decrement operation in the tone pulse counter. Also, an overflow condition in the decrement counter enables the amplitude increment/decrement logic to recalculate the amplitude value to be loaded into the PWM counter depending on the setting of the automatic increase/decrease enable bit (SGn_ECRL:AUTO/IDS/ELS).
6. If the tone pulse counter counts a number of tone pulses up to the values in the tone output number reload register (SGn_NRL) and the time cycle data reload register (SGn_TCRL) (if the tone pulse counter is ‘0x00’, the decrement counter is ‘0x00’ and also the timing of SGO turns ‘H’ from ‘L’), an interrupt setting request is generated. Then the interrupt status bit (SGn_CR0:TCINT) is set and the interrupt request is generated.
7. Set the SGn_CR0:STOP to ‘1’. The Sound Generator stops generating sound. When the STOP bit is set to’1’, then SGn_CR0: RUNNING is cleared, indicating that sound generation has been stopped.

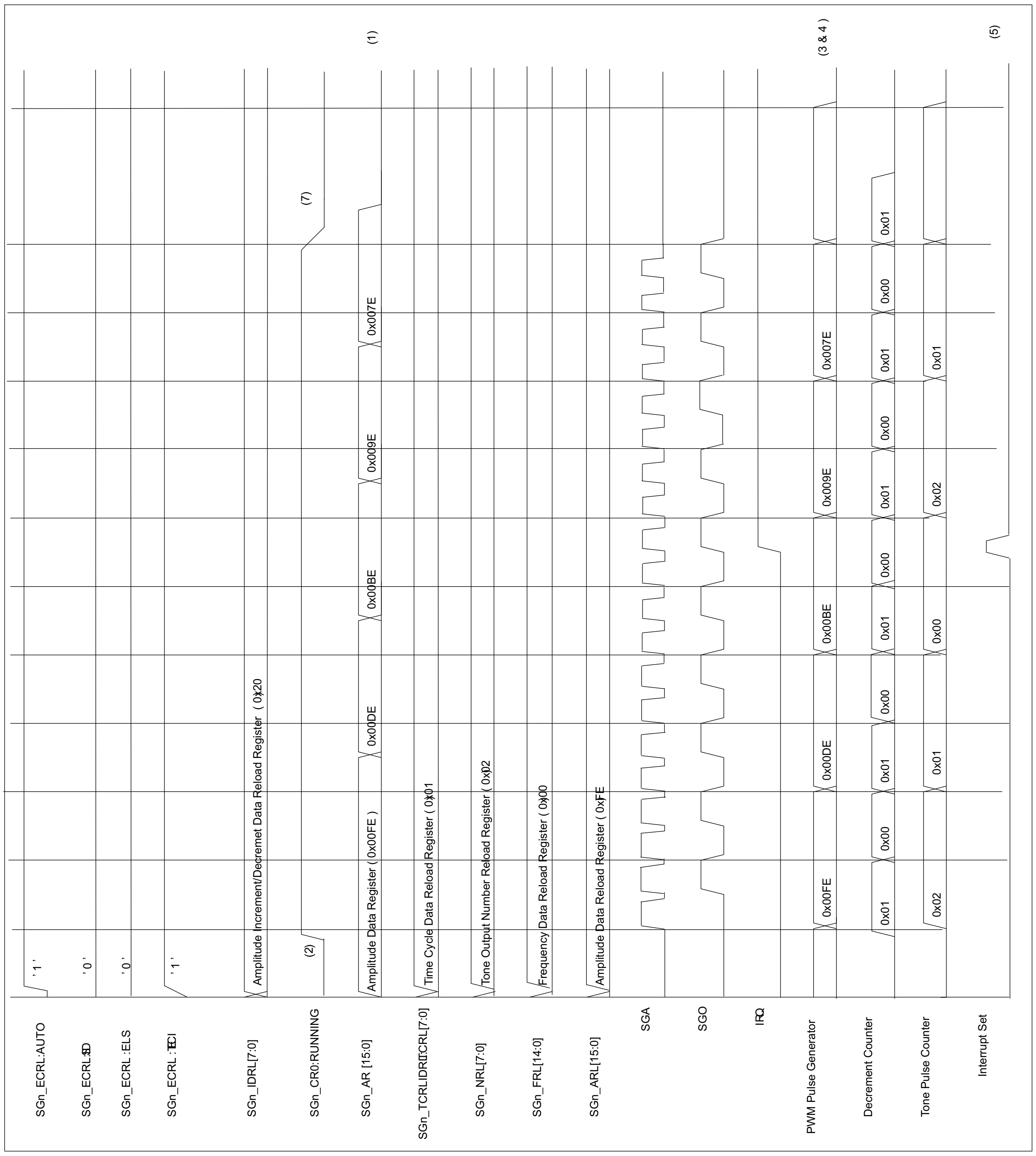


Figure 6-31: Sound Generator operation timing diagram

6.5.4 Sound Generator Register Overview

Table 6-13: Registers Overview

Base Address(es)	**Instance no 0: BASEADDR="00097000"**	
Absolute Address	**Register Name**	**Register Description**
BASEADDR + 0x0000	SGn_CR0	Sound Generator Control Register 0
BASEADDR + 0x0002	SGn_CR1	Sound Generator Control Register 1
BASEADDR + 0x0003	Reserved	Do not modify
BASEADDR + 0x0004	SGn_ECRL	Sound Generator Extended Control Reload Register
BASEADDR + 0x0006	SGn_FRL	Sound Generator Frequency Data Reload Register
BASEADDR + 0x0008	SGn_ARL	Sound Generator Amplitude Data Reload Register
BASEADDR + 0x000A	SGn_AR	Sound Generator Amplitude Status Register
BASEADDR + 0x000C	SGn_TARL	Sound Generator Target Amplitude Data Reload Register
BASEADDR + 0x000E	SGn_TCRLIDRL	Sound Generator Time Cycle Data Reload Register and Increment or Decrement Data Reload Register
BASEADDR + 0x0010	SGn_NRL	Sound Generator Tone Output Number Reload Register
BASEADDR + 0x0011	Reserved	Do not modify
BASEADDR + 0x0012	SGn_DER	Sound Generator DMA Transfer Update Enable Register
BASEADDR + 0x0014	SGn_DMAR	Sound Generator DMA Transfer Indirect Data Register

6.6 LIN / U(S)ART Interface

The LIN-USART with LIN (Local Interconnect Network) unit is a general-purpose, serial data communication interface used for performing synchronous or asynchronous communication with external devices. The LIN-USART provides bidirectional communication (normal mode), master-slave communication (multiprocessor mode in master/slave systems) and special features for LIN-bus systems (working both as a master or as a slave device).

6.6.1 Features of the LIN/U(S)ART Interface

- Full-duplex data buffer
- Serial Input 5 times oversampling in asynchronous mode
- Clock synchronous (start-stop synchronization and start-stop-bit option) and clock asynchronous (using start-, stop-bits) transfer mode
- Automatic baud rate adjust is available in LIN mode.
- 7 bits (not in synchronous or LIN mode) or 8 bits data rate
- Non-return to Zero (NRZ) or Non-return to Zero Inverted (NRZI) signal mode
- Reception error detection
  - Framing error
  - Overrun error
  - Parity error in Normal mode
  - Checksum error in LIN mode
  - Sync. field time out error in LIN mode
  - Parity error in Frame-ID in LIN mode
- Independent 16 byte FIFO for transmission and reception
- Reception interrupt, Transmission interrupt and Error interrupt
- Transmission and reception DMA request support
- Synchronous mode Function as Master or Slave LIN-USART
- LIN bus options
  - Operation as master device
  - Operation as slave device
  - Generation of LIN-Sync-break
  - Detection of LIN-Sync-break
  - Auto baud rate detection and adjustment
  - Interrupt at the end of transmission/reception of complete header of LIN frame.
  - Checksum generation and checksum verification.
  - Detection of physical bus error
  - Transmission /Reception FIFO of configurable depth 1 to 16 bytes

6.6.2 Block Diagram

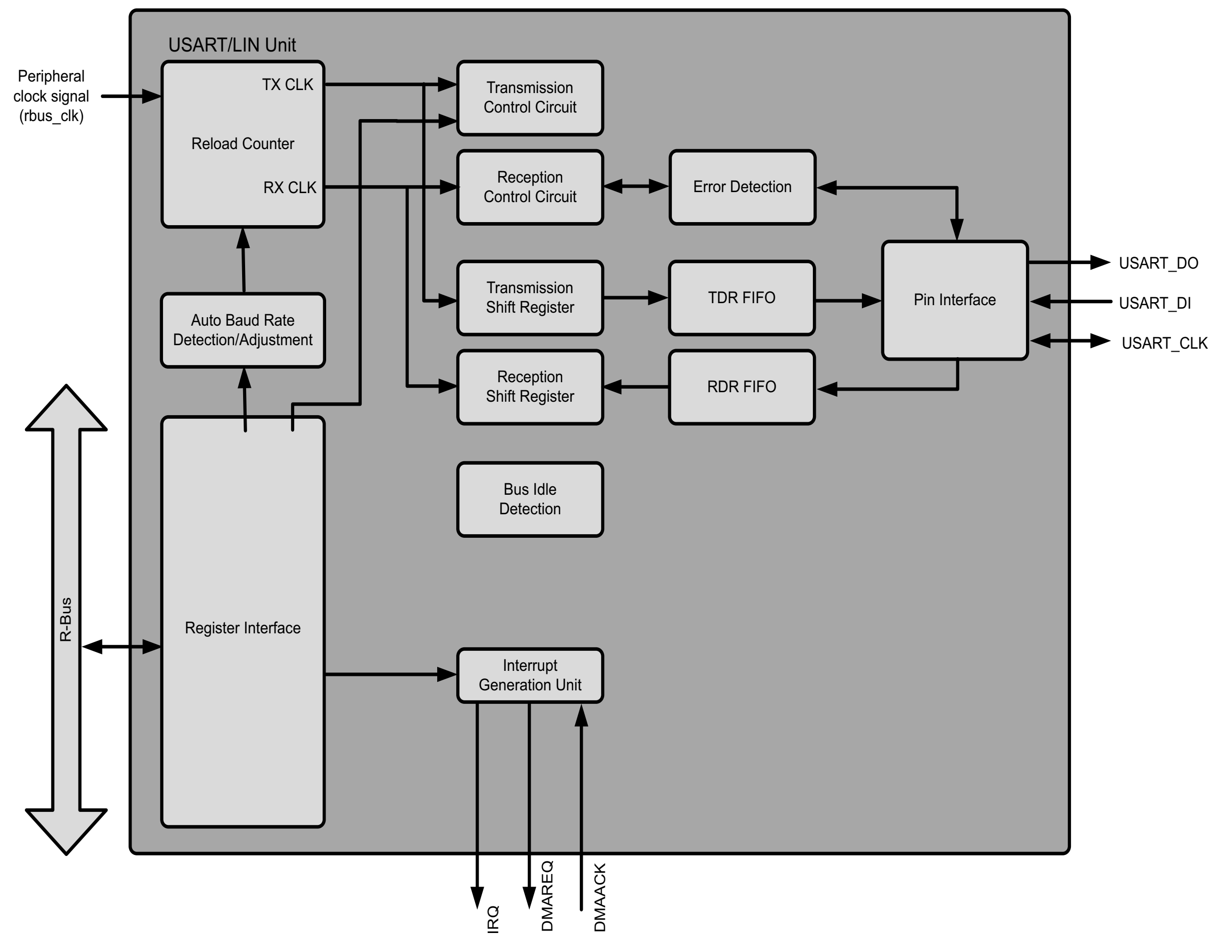


Figure 6-32: Block diagram of LIN-USART

6.6.3 Functional Description

LIN-USART consists of the following blocks:

- Reload counter
- Reception control circuit
- Reception Shift Register
- Reception Data Register (RDRn)
- Transmission control circuit
- Transmission Shift Register
- Transmission Data Register (TDRn)
- Error detection circuit
- Oversampling unit
- Interrupt generation circuit
- LIN sync break/sync field detection
- Bus idle detection circuit
- Checksum generation/verification circuit
- Automatic baud rate detection and adjustment circuit
- FIFO control circuit
- Serial Mode Register (SMRn)
- Serial Control Register (SCRn)
- Serial Status Register (SSRn)
- Extended Communication Control Register (ECCRn)
- Extended Status/Control Register (ESCRn)
- Extended Serial Interrupt Register (ESIRn)
- Extended Interrupt Enable Register (EIERn)
- Transmission FIFO Control Register (TFCRn)
- Reception FIFO Control Register (RFCRn)
- Transmission FIFO Status Register (TFSRn)
- Reception FIFO Status Register (RFSRn)
- Extended Feature Enable Register (EFERn)
- Extended Status Register (ESRn)
- Checksum Status and Control Register (CSCRn)
- Frame-ID Register (FIDRn)
- Sync Field Timeout Register (SFTRn)
- Baud Rate Generation Reload Register (BGRLn)
- Baud Rate Generation Register (BGRn)
- Serial RX-DMA Configuration Register (SRXDRn)
- Serial TX-DMA Configuration Register (STXDRn)
- Set/Clear registers to configure the respective registers

Note:

The suffix ‘n’ denotes the LIN-USART number.

Reload Counter

The reload counter works as the dedicated baud rate generator. It can select external input clock or internal clock for the transmitting and receiving clocks. The reload counter has a 19-bit register for the reload value. The current counter value of the transmission reload counter can be read via the BGRn.

When the auto baud rate detection/adjustment feature is enabled, the BGRn is loaded with value from the auto baud rate detection circuit.

Reception Control Circuit

The reception control circuit consists of a received bit counter, start bit detection circuit, and received parity counter. The received bit counter counts reception data bits. When reception of one data frame for the specified data length is complete, the received bit counter sets the reception data register full flag. The start bit detection circuit detects start bits from the serial input signal and sends a signal to the reload counter to synchronize it to the falling edge of these start bits. The reception parity counter calculates the parity of the reception data.

Reception Shift Register

The Reception Shift Register fetches reception data input from the USART_DI pin, shifting the data bit by bit. When reception is complete, the Reception Shift Register transfers receive data to the RDRn register.

Reception Data Register (RDRn)

This register retains reception data. Serial input data is converted and stored in this register.

Transmission Control Circuit

The transmission control circuit consists of a transmission bit counter, transmission start circuit, and transmission parity counter. The transmission bit counter counts transmission data bits. When the transmission of one data frame of the specified data length is complete, the transmission bit counter sets the Transmission Data Register full flag. The transmission start circuit starts transmission when data is written to TDRn. The transmission parity counter generates a parity bit for data to be transmitted if parity is enabled.

Transmission Shift Register

The Transmission Shift Register loads data written to the TDRn and outputs the data to the USART_DO pin, shifting the data bit by bit.

Transmission Data Register (TDRn)

This register sets transmission data. Data written to this register is converted to serial data and is output to USART_DO.

Error Detection Circuit

The error detection circuit checks if there was any error during the last reception/transmission. If an error has occurred it sets the corresponding error flags.

Oversampling Unit

The oversampling unit oversamples the incoming data at the USART_DI pin for five times. It is switched off in synchronous operation mode.

Interrupt Generation Circuit

The interrupt generation circuit administers all cases of generating a reception or transmission or error interrupt. If a corresponding enable flag is set and an interrupt case occurs the interrupt will be generated immediately.

LIN Sync Break and Synchronization Field Detection Circuit

The LIN break and LIN synchronization field detection circuit detects a LIN break, if a LIN master node is sending a message header. If a LIN break is detected a special flag bit is generated. The first and the fifth falling edge of the synchronization field is recognized by auto baud rate detection circuit to measure the actual serial clock time of the transmitting master node.

LIN Sync Break Generation Circuit

The LIN break generation circuit generates a LIN break of a determined length.

Bus Idle Detection Circuit

The bus idle detection circuit recognizes if neither reception nor transmission is going on. In this case, the circuit generates the special flag bits TBI and RBI.

Checksum Generation/Verification Circuit

The checksum generation/verification circuit generates the checksum bytes during transmission of data bytes only in mode 3 (LIN mode). Similarly it verifies the calculated checksum against the received one during reception.

Auto Baud Rate Detection and Adjustment Circuit

The auto baud rate detection and adjustment circuit programs the BGRn register by calculating the bit time of the sync field for synchronization to the LIN master.

Transmission/Reception FIFO

The transmission/reception FIFO are of 16 bytes each. It is used for storing the data during transmission and reception.

Serial Mode Register (SMRn)

This register performs the following operations:

– Selecting the LIN-USART operation mode
– Selecting a clock input source
– Selecting if an external clock is connected ‘one-to-one’ or connected to the reload counter
– Resetting dedicated Reload Timer
– Resetting the LIN-USART (preserving the settings of the registers)

Serial Control Register (SCRn)

This register performs the following operations:

– Specifying whether to provide parity bits
– Selecting parity bits
– Specifying a stop bit length
– Specifying a data length
– Selecting a frame data format in mode 1
– Clearing the error flags
– Specifying whether to enable transmission
– Specifying whether to enable reception

Serial Status Register (SSRn)

This register performs the following functions:

– Indicating status of receive/transmit operations and errors
– Specifying LSB first or MSB first
– Receive interrupt enable/disable

- Transmit interrupt enable/disable

Extended Status/Control Register (ESCRn)

This register performs the following functions:

- LIN sync break interrupt enable/disable
- Indicating LIN sync break detection
- Specifying LSB bits of LIN sync break length
- Directly accessing USART_DI and USART_DO pins
- Specifying continuous clock output operation
- Specifying sampling clock edge

Extended Communication Control Register (ECCRn)

This register performs the following functions:

- Indicating bus idle state
- Specifying synchronous clock
- Specifying LIN sync break generation

Extended Interrupt Enable Register (EIERn)

This register performs the following function:

- Indicates the Interrupt enable for various interrupt sources

Extended Serial Interrupt Register (ESIRn)

This register performs the following function:

- Change handling of interrupts to enable usage together with DMA
- Indicates the Interrupt Enable for a few of the receive errors

Transmission FIFO Control Register (TFCRn)

This register performs the following function:

- Enables transmission FIFO
- Specify trigger levels for interrupt during transmission

Reception FIFO Control Register (RFCRn)

This register performs the following function:

- Enables reception FIFO
- Specify trigger levels for interrupt during reception

Transmission FIFO Status Register (TFSRn)

This register performs the following function:

- Indicates the number of valid data bytes in transmission FIFO

Reception FIFO Status Register (RFSRn)

This register performs the following function:

- Indicates the number of valid data bytes in reception FIFO

Extended Feature Enable Register (EFERn)

This 16-bit register performs the following function:

– Enables interrupt at the end of transmission and reception of header in LIN mode
– Enables auto baud rate detection and adjustment feature in LIN mode
– Enables edge sensitive detection of LIN break when operating in LIN mode
– Disables resetting of reception state machine when errors are cleared by CRE
– Disables detection of low level of USART_DI after framing error as a valid start signal
– Enables detection of bus error
– Enables Frame-ID register, when operating in LIN mode
– Specifies MSB of LIN break length, when operating in LIN mode

Checksum Status and Control Register (CSCRn)

This register performs the following function in LIN mode:

– Enables checksum generation at transmission side and checksum validation at the reception side
– Selects between classic and enhanced checksum according to LIN specification 2.1
– Indicates the status of checksum during reception
– Specifies the number of data bytes in a LIN frame
– Specifies the Interrupt Enable for CRC reception error interrupt

Extended Status Register (ESRn)

This register performs the following function:

– Indicates the status of bus error detection
– Indicates the status of reception of Frame-ID in LIN mode
– Indicates the status of sync field detection in LIN mode

Frame-ID Register (FIDRn)

This register performs the following function:

– Stores the value of Frame-ID when LIN-USART is acting as a LIN node

Sync Field Timeout Register (SFTRn)

This register performs the following function in LIN mode:

– Contains the 16-bit timeout value considered for sync field detection

Serial RX-DMA Configuration Register (SRXDRn)

This register performs the following function:

– Contains control bits to configure the RX-DMA request for single and demand transfers

Serial TX-DMA Configuration Register (STXDRn)

This register performs the following function:

– Contains control bits to configure the TX-DMA request for single and demand transfers

6.6.4 Operation of LIN-USART

LIN-USART operates in operation mode 0 for normal bidirectional serial communication, in mode 2 and 3 for bidirectional communication as master or slave, and in mode 1 as master or slave in multiprocessor communication.

The LIN-USART pins are shared with general purpose. Please refer to the pin list.

6.6.4.1 LIN-USART Operation Modes

The LIN-USART operates in four different modes, which are determined by the MD0-bit and the MD1-bit of the Serial Mode Register (**SMR**n). Mode 0 and 2 are used for bidirectional serial communication, mode 1 for master/slave communication and mode 3 for LIN master/slave communication.

Table 6-14: LIN-USART operation modes

Operation Mode		Data Length		Synchronization of Mode	Length of Stop Bit	Data Bit Direction [*1]
		Parity Disabled	Parity Enabled			
0	Normal mode	7 or 8		Asynchronous	1 or 2	L/M
1	Multiprocessor	7 or 8 + 1 [*2]	--	Asynchronous	1 or 2	L/M
2	Normal mode	8 or 9 (if SSM = 1)		Synchronous	0, 1 or 2	L/M
3	LIN mode	8	--	Synchronous	1	L

*1: means the data bit transfer format: LSB or MSB first.
*2: ‘+1’ means the indicator bit of the address/data selection in the multiprocessor mode, instead of parity.

NOTE Mode 1 operation is supported both for master or slave operation of the LIN-USART in a master-slave connection system. In Mode 3 the LIN-USART function is locked to 8N1-format, LSB first.
If the mode is changed, LIN-USART cuts off all possible transmission or reception and awaits then new action.
The MD1 and MD0 bit of the Serial Mode Register (**SMR**n) determine the operation mode of the LIN-USART as shown in the following table.

Table 6-15: Mode bit setting

MD1	MD0	Mode	Description
0	0	0	Asynchronous (normal mode)
0	1	1	Asynchronous (multiprocessor mode)
1	0	2	Synchronous (normal mode)
1	1	3	Asynchronous (LIN mode)

6.6.4.2 Inter-CPU Connection Method

External clock one-to-one connection (normal mode) and master-slave connection (multiprocessor mode) can be selected. For either connection method, the data length, whether to enable parity, and the synchronization method must be common to all CPUs. Select an operation mode as follows:

- In the one-to-one connection method, operation mode 0 or 2 must be used in the two CPUs. Select operation mode 0 for asynchronous transfer mode and operation mode 2 for synchronous transfer mode.

NOTE One CPU has to be configured as master and the other has to be configured as slave in synchronous mode 2.

NOTE Select operation mode 1 for the master-slave connection method and use it either for the master or slave system.

6.6.4.3 Synchronization Methods

In asynchronous operation LIN-USART reception clock is automatically synchronized to the falling edge of a received start bit.

In synchronous mode the synchronization is performed either by the clock signal of the master device or by LIN-USART itself if operating as master.

6.6.4.4 Signal Mode

LIN-USART can treat data in Non-Return to Zero (NRZ) format or Non-Return to Zero Inverted (NRZI) format.

6.6.4.5 Operation Enable Bit

LIN-USART controls both transmission and reception using the operation enable bit for transmission (SCRn: TXE) and reception (SCRn: RXE).

- If reception operation is disabled during reception (data is input to the Reception Shift Register), finish frame reception and read the received data of the Reception Data Register (RDRn). Then stop the reception operation.
- If the transmission operation is disabled during transmission (data is output from the Transmission Shift Register), wait until there is no data in the Transmission Data Register (TDRn) before stopping the transmission operation.

6.6.4.6 Operation in Asynchronous Mode (Operation Modes 0 and 1)

When LIN-USART is used in operation mode 0 (normal mode) or operation mode 1 (multiprocessor mode), the asynchronous transfer mode is selected.

6.6.4.6.1 Operation in Asynchronous Mode

Transfer Data Format

Generally, each data transfer in the asynchronous mode operation begins with the start bit (low-level on bus) and ends with at least one stop bit (high-level). The direction of the bit stream (LSB first or MSB first) is determined by the BDS bit of the Serial Status Register (SSRn). The parity bit (if enabled) is always placed between the last data bit and the (first) stop bit.

In operation mode0, the length of the data frame can be 7- or 8-bits, with or without parity, and 1 or 2 stop bits.

In operation mode1, the length of the data frame can be 7- or 8-bits with a following address-/data-selection bit instead of a parity bit. 1 or 2 stop bits can be selected.

The calculation formula for the bit length of a transfer frame is:

Length = 1 + d + p + s

(d = number of data bits [7 or 8], p = parity [0 or 1], s = number of stop bits [1 or 2].

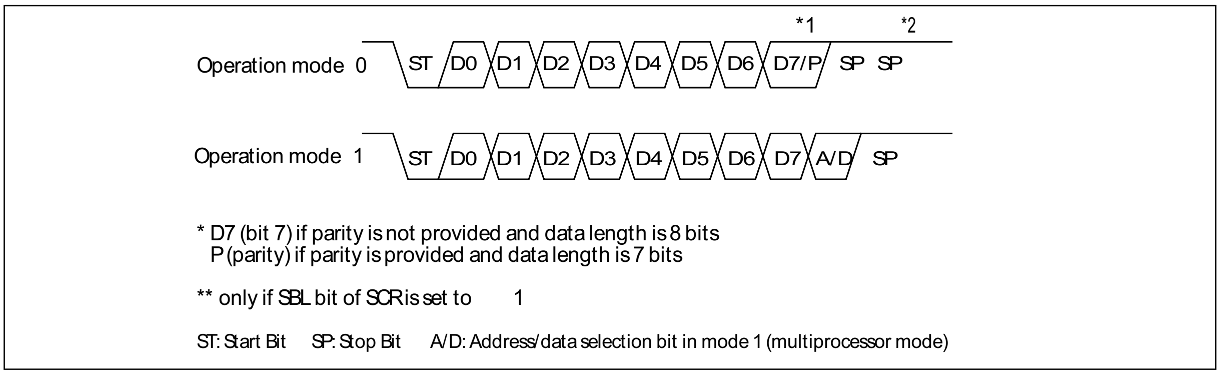

*1
*2
Operation mode 0
ST
D0
D1
D2
D3
D4
D5
D6
D7/P
SP
SP
Operation mode 1
ST
D0
D1
D2
D3
D4
D5
D6
D7
A/D
SP
* D7 (bit 7) if parity is not provided and data length is 8 bits
P (parity) if parity is provided and data length is 7 bits
** only if SBL bit of SCR is set to 1
ST: Start Bit
SP: Stop Bit
A/D: Address/data selection bit in mode 1 (multiprocessor mode)

Figure 6-33: Transfer data format (operation modes 0 and 1))

NOTE If BDS bit of the Serial Status Register (**SSR**n) is set to ‘1’ (MSB first), the bit stream processes as: D7, D6,..., D1, D0, (P).

During reception both stop bits are detected, if selected. But the Reception Data Register Full (RDRF) flag will go ‘1’ at the first stop bit. The bus idle flag (ECCRn:RBI) goes ‘1’ after the second stop bit if no further start bit is detected. (The second stop bit belongs to ‘bus activity’, although it is just mark level.)

Transmission Operation

If the Transmission Data Register Empty (TDRE) flag bit of the Serial Status Register (**SSR**n) is ‘1’, transmission data is allowed to be written to the Transmission Data Register (**TDR**n). When data is written, the TDRE flag goes ‘0’. If the transmission operation is enabled by the TXE-bit (‘1’) of the Serial Control Register (**SCR**n), the data is written next to the Transmission Shift Register and the transmission starts at the next clock cycle of the serial clock, beginning with the start bit. Thereby, the TDRE flag goes ‘1’, so that new data can be written to the **TDR**n.

If transmission interrupt is enabled (TIE = 1), the interrupt is generated by the TDRE flag.

NOTE The initial value of the TDRE flag is ‘1’, so that in this case if TIE is set to ‘1’ an interrupt will occur immediately.

When the character length is set to 7-bits (CL = 0), the unused bit of the **TDR**n is always the MSB, independently from the bit direction setting in the BDS bit (LSB first or MSB first).

Reception Operation

Reception operation is performed when it is enabled by the Reception Enable (RXE) flag bit of the **SCR**n. If a start bit is detected, a data frame is received according to the format specified by the **SCR**n. In case of errors, the corresponding error flags are set (PE, ORE, FRE). After the reception of the data frame, the data is transferred from the Serial Shift Register to the Reception Data Register (**RDR**n) and the Receive Data Register Full (RDRF) flag bits of **SSR**n and **ESIR**n registers are set.

If receive interrupt is enabled (RIE = 1) and ESIRn:AICD = 0, the interrupt is generated by SSRn:RDRF.

If receive interrupt is enabled (RIE = 1) and ESIRn:AICD = 1, the interrupt is generated by ESIRn:RDRF.

The data then has to be read by the CPU. By doing so, the SSRn:RDRF flag is cleared.

When ESIRn:AICD = 0, this also clears the interrupt.

When ESIRn:AICD = 1, writing ‘1’ to ESIRn:RDRF clears the interrupt

When the character length is set to 7-bits (CL = 0), the unused bit of the **RDR**n is always the MSB, independently from the bit direction setting in the BDS bit (LSB first or MSB first).

NOTE Only when the RDRF flag bit is set and no errors have occurred the Reception Data Register (**RDR**n) contains valid data.

Used Clock

Use the internal clock or external clock. Select the baud rate generator (SMRn:EXT = 0 or 1, SMRn:OTO = 0) for desired baud rate.

Stop Bit, Error Detection, and Parity

Number of stop bit, 1 or 2 can be specified by the SBL bit of the **SCR**n register. When receiving and 2-bit is set to the stop bit, the second stop bit is checked in addition to the first stop bit. The RBI (bus idle) flag is set after the second stop bit. However, the RDRF flag is set when the first stop bit is received. In mode 0, parity error, overrun error, and framing error are checked. In mode 1, parity check is not supported and overrun error and framing error are checked. The PEN bit of the **SCR**n register enables/ disables the parity bit and the P bit specifies even or odd parity in mode 0.

6.6.4.7 Operation in Synchronous Mode (Operation Mode 2)

The clock synchronous transfer method is used for LIN-USART operation mode 2 (normal mode).

Transfer Data Format

In the synchronous mode, 8-bit data is transferred without start or stop bits if the SSM bit of the Extended Communication Control Register (**ECCR**n) is 0. The following figure illustrates the data format during a transmission in the synchronous operation mode.

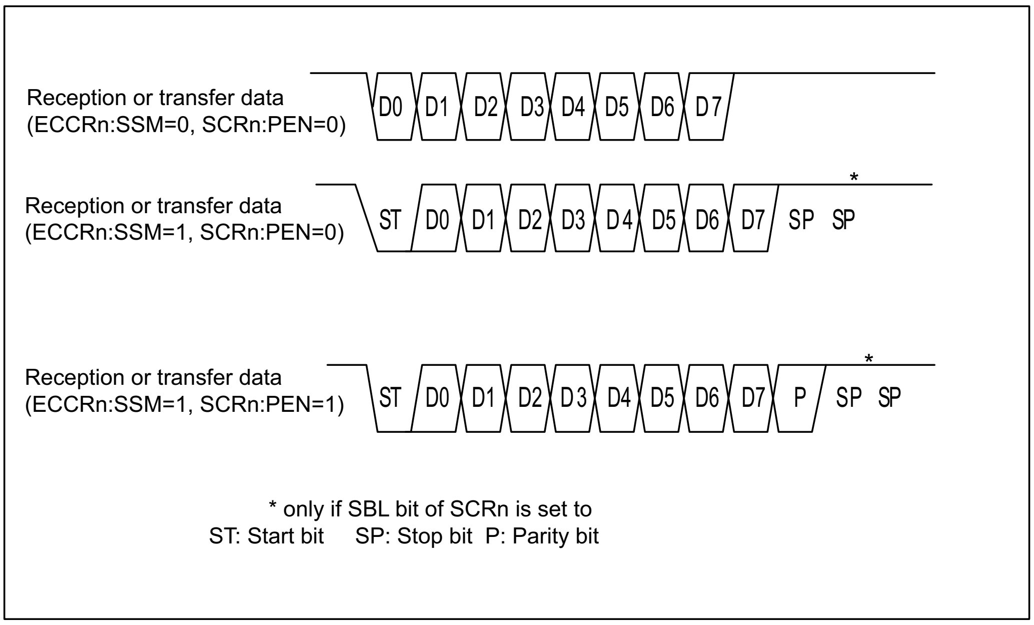

Reception or transfer data
(ECCRn:SSM=0, SCRn:PEN=0)
D0 D1 D2 D3 D4 D5 D6 D7
Reception or transfer data
(ECCRn:SSM=1, SCRn:PEN=0)
ST D0 D1 D2 D3 D4 D5 D6 D7 SP SP
*
Reception or transfer data
(ECCRn:SSM=1, SCRn:PEN=1)
ST D0 D1 D2 D3 D4 D5 D6 D7 P SP SP
*
* only if SBL bit of SCRn is set to
ST: Start bit SP: Stop bit P: Parity bit

Figure 6-34: Transfer data format (operation mode 2)

Clock Inversion and Start/stop Bits in Mode 2

If the SCES bit of the Extended Status/Control Register (**ESCR**n) is set, the serial clock is inverted. Therefore, in slave mode, LIN-USART samples the data bits at the falling edge of the received serial clock.

NOTE

- In master mode if SCES is set, the clock signal’s mark level is ‘0’.
- If the SSM bit of the Extended Communication Control Register (**ECCR**n) is set, the data format gets additional start and stop bits like in asynchronous mode.

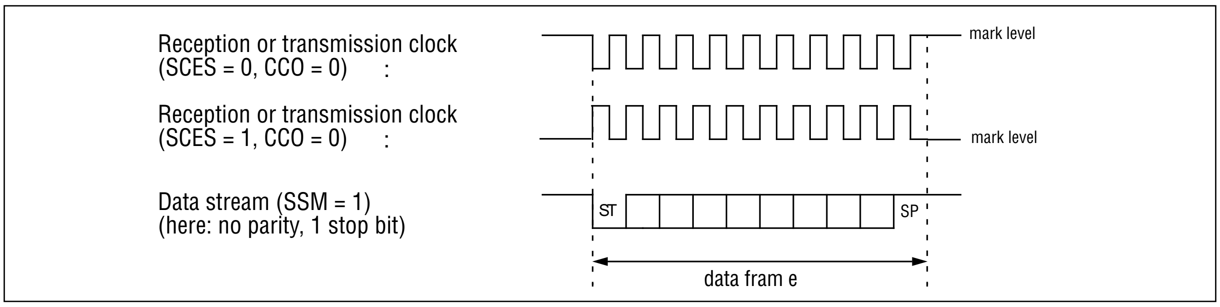


Figure 6-35: Transfer data format with clock inversion

Clock Supply

In operation mode 2, the number of clock cycles for the clock signal must be the same as the number of bits for the data including start and stop bits.

If the MS bit of the **ECCR**n register is ‘0’ (master mode), the consistent clock cycles are generated automatically.

If the MS bit of the **ECCR**n register is ‘1’ (slave mode), ensure that correct clock cycles are generated by the other communication device. While there is no communication, the clock signal must be kept at ‘1’ as the mark level.

If the SCDE bit of the **ECCR**n register is ‘1’, the clock output signal is delayed by the half of the serial clock cycle as shown in Figure 6-36. The operation is prepared for communication devices which use the falling edge of the serial clock signal for data sampling.

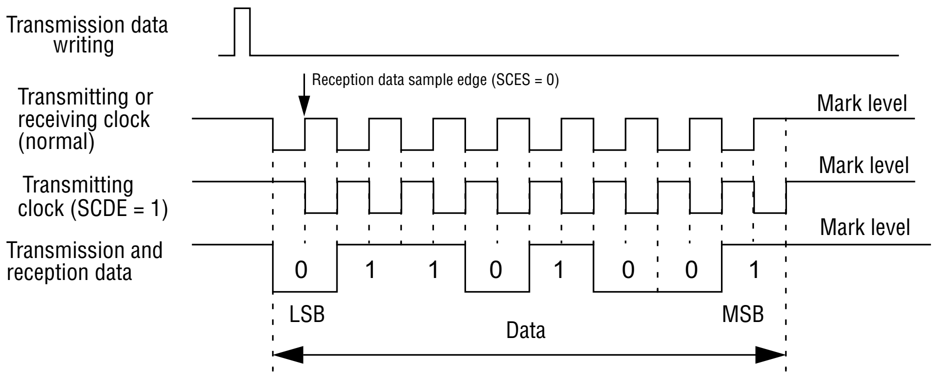

Transmission data writing
Reception data sample edge (SCES = 0)
Transmitting or receiving clock (normal)
Mark level
Transmitting clock (SCDE = 1)
Mark level
Transmission and reception data
Mark level
0
1
1
0
1
0
0
1
LSB
MSB
Data

Figure 6-36: Delayed transmitting clock signal (SCDE = 1)

If the SCES bit of the **ESCR**n register is ‘1’, the serial clock signal is inverted. Receiving data is sampled at the falling edge of the serial clock.

If the MS bit of the **ECCR**n register is ‘0’ (master mode), the output clock signal is also inverted.

While there is no communication, the clock signal must be kept at ‘0’ as the mark level.

If the CCO bit of the **ESCR**n register is ‘1’, the serial clock is signaled even while there is no data communication. Therefore, it is recommended to specify the start/stop bits as shown in
Figure on page 70.

Table 6-16: Serial data input sampling depending on ECCRn:SCDE and ESCRn:SCES

Sr. No	ECCRn: SCDE	ESCRn: SCES	Serial Data Sampling Edge	Serial Data Transmitting Edge
1	0	0	Rising edge	Falling edge
2	0	1	Falling edge	Rising edge
3	1	0	Falling edge	Rising edge
4	1	1	Rising edge	Falling edge

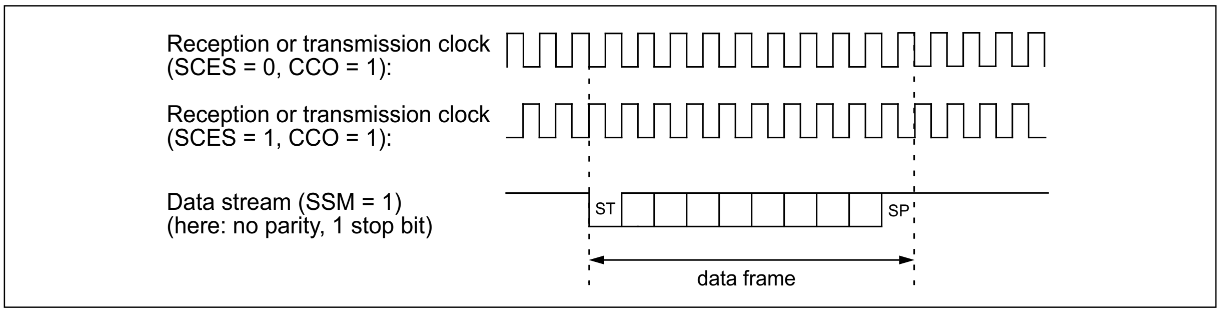

Reception or transmission clock
(SCES = 0, CCO = 1):
Reception or transmission clock
(SCES = 1, CCO = 1):
Data stream (SSM = 1)
(here: no parity, 1 stop bit)
ST
SP
data frame

Figure 6-37: Continuous clock output in mode 2

Error Detection

If no start/stop bits are selected (ECCRn: SSM = 0) only overrun errors are detected.

Communication

For initialization of the synchronous mode, following settings have to be done:

Baud Rate Generator Reload Registers (BGRLn)

Set the desired reload value for the dedicated baud rate reload counter.

Serial Mode Control Register (SMRn)

MD1, MD0: ‘10_B’ (mode 2)

Serial Control Register (SCRn)

RXE, TXE: set both of these flags to ‘0’

A/D: no address/data selection - don’t care

CL: automatically fixed to 8-bit data - don’t care

CRE: ‘1’ to clear receive error flags.

- when SSM=0 (default)
  PEN, P, SBL: don’t care
- when SSM=1
  PEN: ‘1’ if parity bit is added/detected, ‘0’ if not
  P: ‘0’ for even parity, ‘1’ odd parity
  SBL: ‘1’ for 2 stop bits, ‘0’ for 1 stop bit.

Serial Status Register (SSRn)

BDS: ‘0’ for LSB first, ‘1’ for MSB first

RIE: ‘1’ if interrupts are used; ‘0’ receive interrupts are disabled

TIE: ‘1’ if interrupts are used; ‘0’ transmission interrupts are disabled

Extended Communication Control Register (ECCRn)

SSM: ‘0’ if no start/stop bits are desired (normal); ‘1’ for adding start/stop bits (special)

MS: ‘0’ for master mode (LIN-USART generates the serial clock); ‘1’ for slave mode (LIN-USART receives serial clock from the master device)

Serial Control Register (SCRn)

RXE, TXE: set one or both of these control bits to ‘1’ to begin communication.

6.6.4.8 Features of LIN-USART in LIN Mode

LIN-USART in LIN mode supports several additional features which reduces the interrupt load on the CPU:

- Automatic header transmission
- Automatic header detection
- Automatic baud rate detection/adjustment
- LIN-CRC handling with respect to message length
- Detection of bus error
- Transmission/reception FIFO of configurable depth of 16 bytes each
- Variable LIN break length generation

Interrupt at End of Transmission of Complete Header as LIN Master

This feature is enabled by setting the bit (EFERn:ENTXHR to ‘1’ and EIERn:TXHDIE to ‘1’) and started by writing ‘1’ to ECCRn:LBR. LIN will complete the transmission of header and interrupts the CPU at the end of transmission. Enabling transmission, asserting LIN break request and writing Frame-ID in Frame-ID Data Register (if enabled by EFERHn:FIDE = 1) or in TDRn or TX FIFO (if enabled) will result in the following actions:

1. Generation of LIN break as per the length specified in Extended Feature Enable Register (EFERHn:LBL2) and Extended Status and Control Register (ESCRn:LBL[1] and ESCRn:LBL[0] bits).
2. Send sync character 0x55.
3. Send Frame-ID programmed in Frame-ID register, TX-FIFO or TDRn.
4. A transmission interrupt (ESRn:TXHRI = 1) is generated at the end of the Frame-ID transmission (when stop bit is being sent).

Interrupt at the End of Reception of Complete Header as LIN Slave

This feature is enabled by setting the bit (EFERLn:ENRXHR to ‘1’ and EIERn:RXHDIE to ‘1’). LIN slave will complete the reception of the header and assert a reception interrupt at the end of the header reception. Enabling auto baud rate feature and the interrupt generation for reception will result in the following things:

1. Detection of LIN break.
2. Reception of sync field and getting adjusted to LIN network with the help of auto baud rate detection/ adjustment circuitry.
3. Reception of Frame-ID into FIDRn.
4. A reception interrupt (ESRn:RXHRI = 1) is generated at the end of reception of Frame-ID. Possible reception errors are indicated in the corresponding status flags.

Here, the auto baud rate detection/adjustment block is used for calculating the 1-bit time of sync field received. By enabling (EFERLn:ENRXHR and EIERn:RXHDIE), LIN-Break Detect interrupt is not required when header reception is ongoing.

Automatic Baud Rate Adjustment

This feature is enabled by setting EFERLn:ABRE = 1. In the auto baud rate detection/adjustment circuitry, the time elapsed between first and fifth falling edge of sync field is calculated. The calculated value is divided by 8 for determining bit time of sync field**.** The 19-bit reload value of the baud rate counter is loaded into reload counter for generation of reception clock and transmission clock.

LIN-CRC Handling in Regard to Message Length

- CRC Generation
  This feature is enabled by setting the bit CSCRn:CRCGEN to ‘1’. For CRC generation (master sends data to slave) the number of data bytes in the LIN frame is programmed as the data length in the Checksum Status/Control Register (CSCRn:DL[2:0]). By enabling this feature, the checksum byte is generated and transmitted after the data bytes. Both classic and enhanced types of checksum types according to LIN specification 2.1 are supported. In case of classic checksum, only data bytes are considered for calculation of checksum value. In enhanced checksum both data byte and Frame-ID are considered for calculation of checksum value.

  The selection of classic or enhanced checksum is done with the help of programmable bit CRCTYPE in Checksum Status/Control Register. The checksum contains the inverted eight bit sum with carry over all data bytes (classic checksum) or all data bytes and the protected identifier (enhanced checksum).
- CRC Verification
  This feature is enabled by setting the bit CSRCn:CRCCHECK to ‘1’. For CRC verification the number of data bytes in the LIN frame is programmed as the data length in the Checksum Status/Control Register (CSCRn:DL2-DL0). By enabling this feature, the checksum byte received at the end of reception of data bytes (configured by CSCRn:DL[2:0]) is added with internally calculated checksum and checked whether it is equal to 0xFF. If the calculated sum is not equal to 0xFF, the CSCRn:CRCERR flag is set. This will result in a error interrupt when EIERn:CRCERRIE is set to ‘1’.

  CRC generation and verification are done according to LIN specification 2.1.

NOTE

- When master sends data to slave this verification can be used for self checking
- Checksum verification is not performed if a parity error in the corresponding Frame-ID has been received (ESRn:PEFRD= ‘1’). To enable checksum verification, clear ESRn:PEFRD before receiving the frame data

- Detection of Bus Error
  This feature is enabled by setting the bit EFERLn:DBE to ‘1’. The detection of bus error is done by enabling both transmission and reception, so that LIN node can read back its own transmission (as the LIN is single wire network). The physical bus error such as shorted to ground or Vcc can be found by comparing the value of transmitted and received data. If there is a difference between the transmitted and received value then ESRn:BUSERR flag is set. This will result in an error interrupt if EIERn:BUSERRIE is set to ‘1’.

NOTE

- Detection of bus error has to be disabled in internal loopback mode (EFERHn.INTLBEN = “1”).

- FIFO block for transmission and reception
  This feature is enabled by setting the bits EFERn:TXFE, EFERn:RXFE to ‘1’. Transmission and reception FIFO of configurable length 16 bytes each is available for storing the data bytes. The trigger level for TX/RX FIFO interrupts are set by programming the bits TFCRn:TXFLC[4:0], RFCRn:RXFLC[4:0] respectively to the required value in the range from 1 to 16. The number of valid data bytes in the Transmission and reception FIFO are indicated by the following register bits TFSRn:TXFVD[4:0] and RFSRn:RXFVD[4:0] respectively.
- Variable length LIN break generation
  LIN break of variable length from 13-20 bits can be generated. This is possible by setting the bits (EFERHn:LBL[2], ESCRn:LBL[1:0]) to the required value.

6.6.4.9 Operation with LIN Function (Operation Mode 3)

LIN-USART can be used either as LIN-master or LIN-slave. For this LIN function, a special mode is provided. Setting the LIN-USART to mode 3 configures the data format to 8N1-LSB-first format.

6.6.4.9.1 Operation in Asynchronous LIN Mode (Operation Mode 3)

6.6.4.9.2 LIN-USART as LIN master

In LIN master mode the master determines the baud rate of the whole sub bus, therefore slave devices have to synchronize to the master. The desired baud rate remains fixed in master operation after initialization.

If the automatic header transmission is disabled (EFERLn:ENTXHR = ‘0’)

Writing a ‘1’ into the LBR bit of the Extended Communication Control Register (**ECCR**n) generates a 13 to 20-bit time low-level on the USART_DO pin, which is the LIN synchronization break and the start of a LIN message. Thereby the TDRE flag of the Serial Status Register (**SSR**n) goes ‘0’ and is reset to ‘1’ after the break, and generates a transmission interrupt for the CPU (if TIE of SSRn is ‘1’). The length of the synchronization break to be sent can be determined by the LBL2 bit of EFERn and LBL1/0 bits of the ESCRn as follows:

Table 6-17: LIN break length

LBL2	LBL1	LBL0	Length of break
0	0	0	13-bit times
0	1	0	14-bit times
0	0	1	15-bit times
0	1	1	16-bit times
1	0	0	17-bit times
1	0	1	18-bit times
1	1	0	19-bit times
1	1	1	20-bit times

The sync field is sent as byte data of 0x55 after the LIN break. To prevent a transmission interrupt, the 0x55 can be written to the TDRn just after writing the ‘1’ to the LBR bit, although the TDRE flag is ‘0’. The internal transmission shifter waits until the LIN break has finished and shifts the TDRn value out afterwards. In this case no interrupt is generated after the LIN break and before the start bit of the sync field (0x55).

If automatic header transmission is enabled (EFERLn:ENTXHR = 1):

When ECCRn:LBR is set to ’1’, LIN break of length programmed is transmitted. After the transmission of LIN break is finished, sync field value (0x55) is written internally by the LIN-USART to the Transmission Shift Register. After the transmission of sync field, LIN-USART transmits the Frame-ID in the Frame-ID Data Register (when EFERH:FIDE = 1) or TX FIFO (when TFCRn:TXFE = 1) or TDRn. When there is a parity error in the received Frame-ID, the ESRn:PEFRD flag is set. This will result in an error interrupt when EIERn:PEFRDIE is set to ‘1’.
A header transmission interrupt is asserted, after Frame-ID is transferred to the Shift Register (if TIE = 1 and LBSOIE = 0) or after Frame-ID is shifted out (TIE = 0 and LBSOIE = 1).

6.6.4.9.3 LIN-USART - Automatic Header Detection

When automatic header detection is enabled by setting EFERLn:ENRXHR to ‘1’, LIN break, sync field and Frame-ID are detected by LIN-USART automatically. If only EIERn:RXHDIE is enabled only one reception interrupt is set after receiving Frame-ID.

If EFERLn:ABRE = 0, baud rate is not automatically adjusted to the master baud rate.

If EFERLn:ABRE = 1, baud rate is adjusted automatically by the auto baud rate detection/adjustment circuitry after LIN break gets detected.

Detection of sync field is independent of EFERLn:ABRE.

If the sync field is not detected within the timeout value programmed in the Sync Field Timeout Register (SFTRn), ESRn:SYNFE flag is set. This will result in an error interrupt when EIERn:SYNFEIE is set to '1'. Else if the sync field gets detected within the timeout value, the calculated bit time of sync field is loaded to the reload counter for getting synchronized to LIN master. Detection of sync field time out is independent of EFERLn:ABRE.

6.6.4.9.4 LIN Sync Break Detection Interrupt and Flags

If a LIN sync synchronization break is detected in the slave mode, the LIN-Break Detected (LBD) flag of the ESCRn is set to '1'. This causes an interrupt, if the LIN Break Interrupt Enable (LBIE) bit is set.

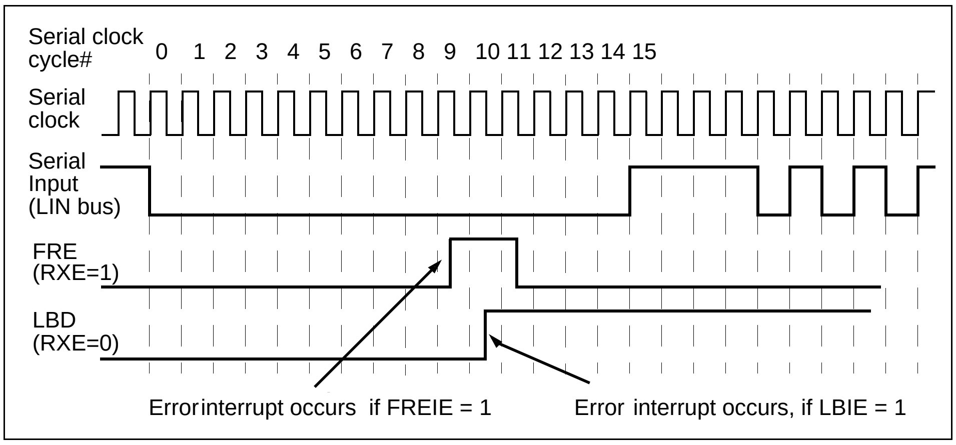


Figure 6-38: LIN sync break detection and flag set timing

The figure above demonstrates the LIN sync break detection and flag set timing.

NOTE

- If reception is enabled (RXE = 1) and framing error interrupt is enabled (EIERn:FREIE = 1), the Reception Data Framing Error (FRE) flag bit of the SSRn causes an error interrupt 2-bit times (‘8N1’) earlier than the LIN break interrupt. So it is recommended to turn off RXE, to avoid that the SSRn:FRE flag is set, if a LIN break is expected (if automatic header reception is not used).
- LIN break can be detected even if RXE is disabled.
- LBD is only supported in operation mode 3.

6.6.4.9.5 LIN Bus Timing

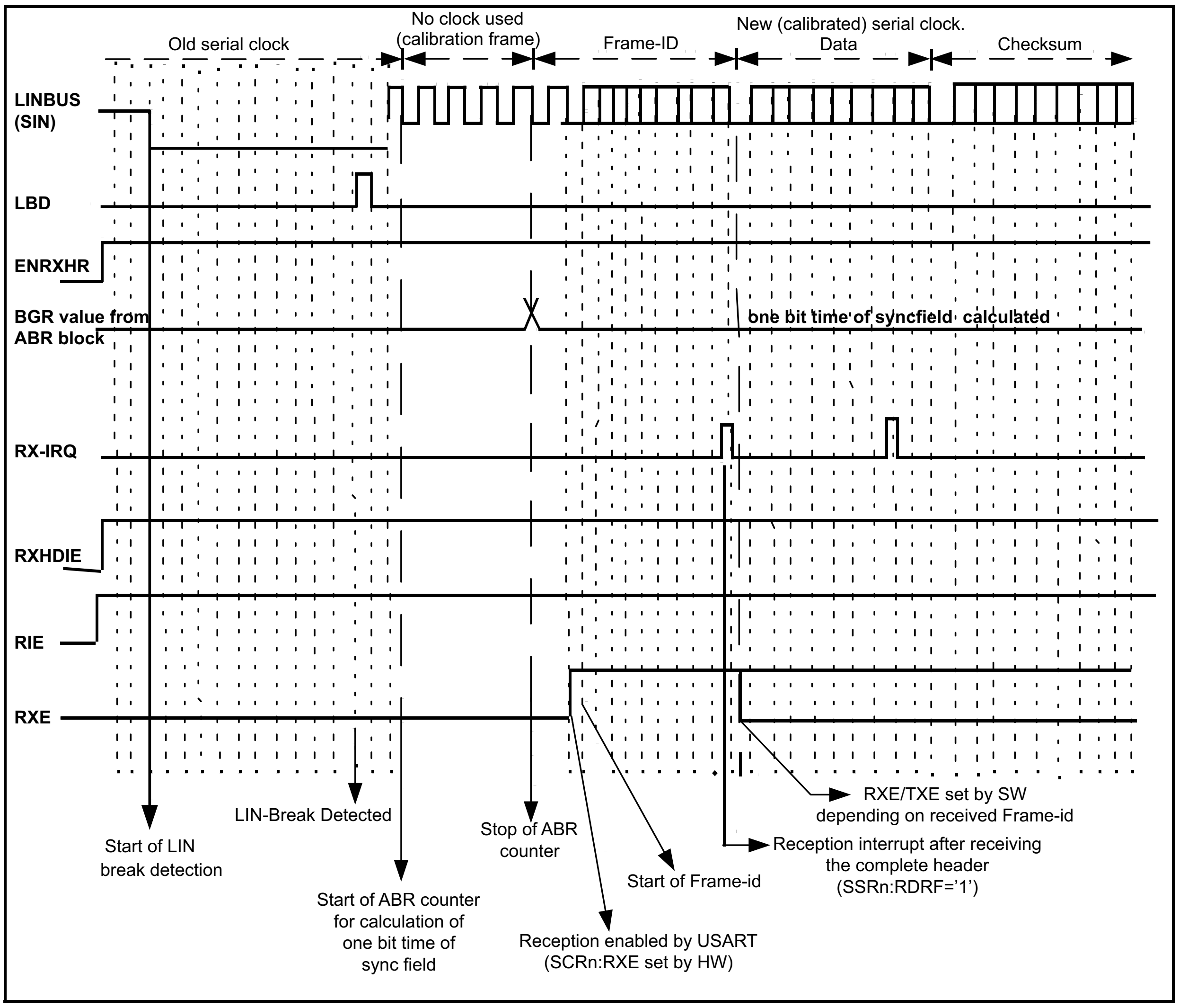

Old serial clock
No clock used (calibration frame)
Frame-ID
New (calibrated) serial clock. Data
Checksum
LINBUS (SIN)
LBD
ENRXHR
BGR value from ABR block
one bit time of syncfield calculated
RX-IRQ
RXHDIE
RIE
RXE
Start of LIN break detection
LIN-Break Detected
Start of ABR counter for calculation of one bit time of sync field
Stop of ABR counter
Reception enabled by USART (SCRn:RXE set by HW)
Start of Frame-id
RXE/TXE set by SW depending on received Frame-id
Reception interrupt after receiving the complete header (SSRn:RDRF=’1’)

Figure 6-39: LIN bus timing and LIN-USART signals with automatic header detection

6.6.4.10 Direct Access to Serial Pins

LIN-USART allows the user to directly access the Transmission Pin (USART_DO or the Reception Pin (USART_DI).

6.6.4.10.1 LIN-USART Direct Pin Access

The LIN-USART provides the ability for the software to access directly serial input or output pins. The software can always monitor the incoming serial data by reading ESCRn:SIOP bit. By setting the Serial Output Pin Direct Access Enable (ESCRn:SOPE) bit, the software can force the USART_DO pin to a desired value.

NOTE This access is only possible if the Transmission Shift Register is empty (i. e. no transmission activity). In LIN mode, this function can be used for reading back the own transmission and is used for error handling, if something is physically wrong with the single-wire LIN-bus.

Write the desired value to ESCSRn:SIOPS (to set the output pin) and ESCCRn:SIOPC (to clear the output pin) before enabling the output pin direct access (ESCRn:SOPE) to prevent undesired output level because ESCSRn:SIOPS and ESCCRn:SIOPC bits hold the last written value.

6.6.4.11 Bidirectional Communication Function (Normal Mode)

In operation mode 0 or 2, normal serial bidirectional communication is available. Select operation mode 0 for asynchronous communication and operation mode 2 for synchronous communication.

6.6.4.11.1 Bidirectional Communication Function

The settings shown in Figure 6-40 on page 77 are required to operate LIN-USART in normal mode (operation mode 0 or 2).

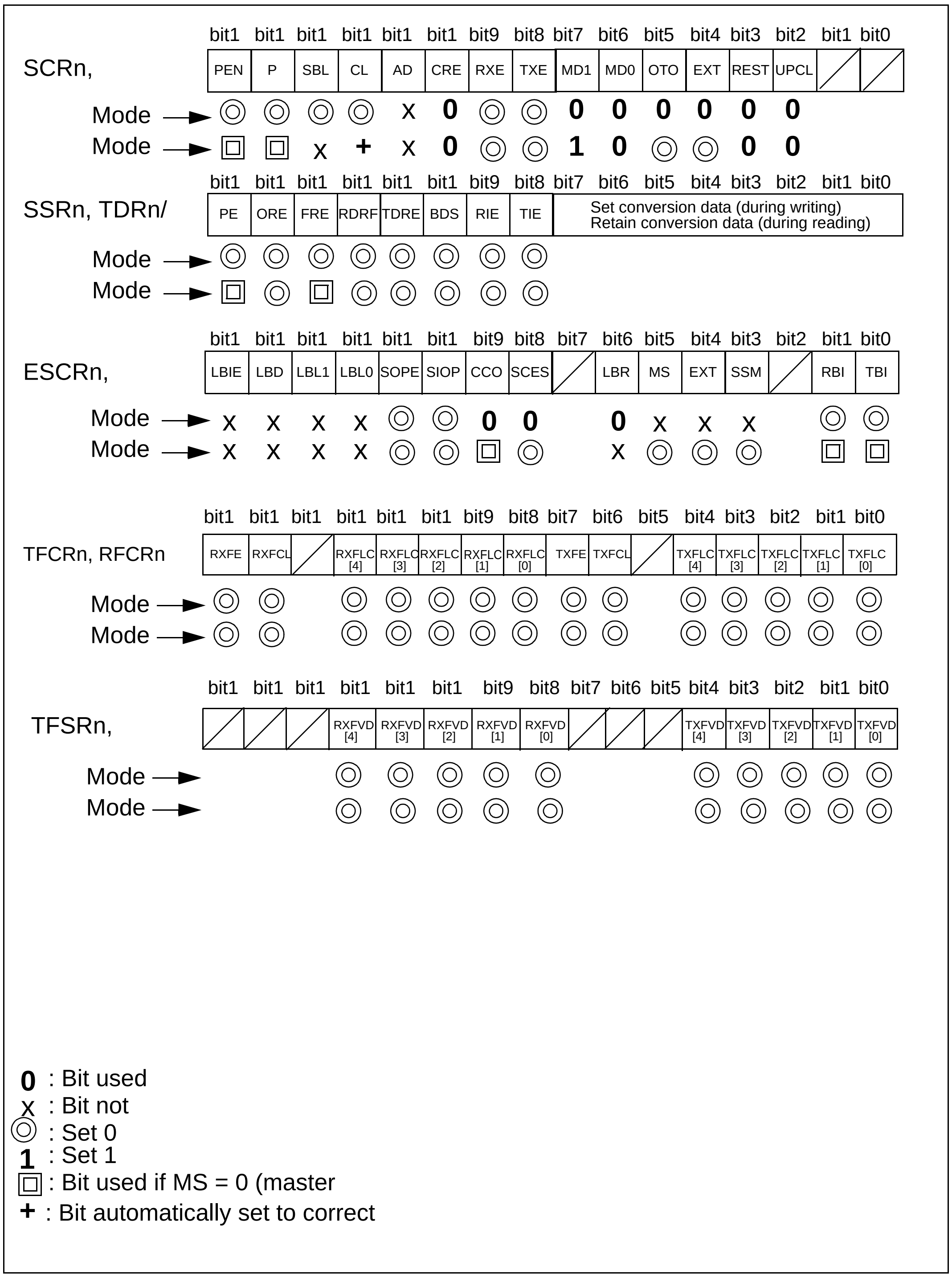

SCRn,
PEN P SBL CL AD CRE RXE TXE MD1 MD0 OTO EXT REST UPCL
SSRn, TDRn/
PE ORE FRE RDRF TDRE BDS RIE TIE
Set conversion data (during writing)
Retain conversion data (during reading)
ESCRn,
LBIE LBD LBL1 LBL0 SOPE SIOP CCO SCES LBR MS EXT SSM RBI TBI
TFCRn, RFCRn
RXFE RXFCL RXFLC[4] RXFLC[3] RXFLC[2] RXFLC[1] RXFLC[0] TXFE TXFCL TXFLC[4] TXFLC[3] TXFLC[2] TXFLC[1] TXFLC[0]
TFSRn,
RXFVD[4] RXFVD[3] RXFVD[2] RXFVD[1] RXFVD[0] TXFVD[4] TXFVD[3] TXFVD[2] TXFVD[1] TXFVD[0]
Mode
0 : Bit used
x : Bit not
◎ : Set 0
1 : Set 1
▣ : Bit used if MS = 0 (master
+ : Bit automatically set to correct

Figure 6-40: Settings for LIN-USART operation mode 0 and 2

EFERLn	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
		OSDE	DTSTART	RSTRFM	LBEDGE	ABRE	ENTXHR	ENRXHR
Mode 0		◎	◎	◎	x	x	x	x
Mode 2		◎	◎	◎	x	x	x	x

EFERH	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
			INTLBEN	BRGR	FIDPE	DBE	FIDE	LBL2
Mode 0			◎	x	x	x	x	x
Mode 2			◎	x	x	x	x	x

EIERn	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
	TXFIE	RXFIE	SYNFDIE	RXHDIE	TXHDIE	PEFRDIE	BUSERRIE	LBSOIE
Mode 0	◎	◎	x	x	x	x	x	x
Mode 2	◎	◎	x	x	x	x	x	x

ESRn	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
			TXHRI	RXHRI	LBSOF	BUSERR	PEFRD	SYNFE
Mode 0			x	x	◎	x	x	x
Mode 2			x	x	◎	x	x	x

CSCRn	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
	CRCERRIE	CRCERR	CRCHECK	CRCTYPE	CRCGEN	DL2	DL1	DL0
Mode 0	x	x	x	x	x	x	x	x
Mode 2	x	x	x	x	x	x	x	x

◎	: Bit used
x	: Bit not
0	: Set 0
1	: Set 1
⊡	: Bit used if MS = 0 (master mode)
+	: Bit automatically set to correct value

Figure 6-41: Settings for LIN-USART operation mode 0 and 2 (contd).

6.6.4.11.2 Inter-CPU Connection

As shown in Figure 6-42, interconnect two devices in LIN-USART mode 2.

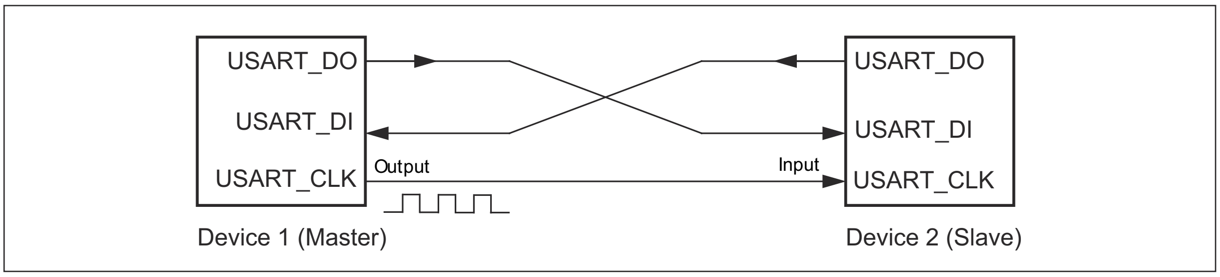


Figure 6-42: Connection example of LIN-USART mode 2 bidirectional communication

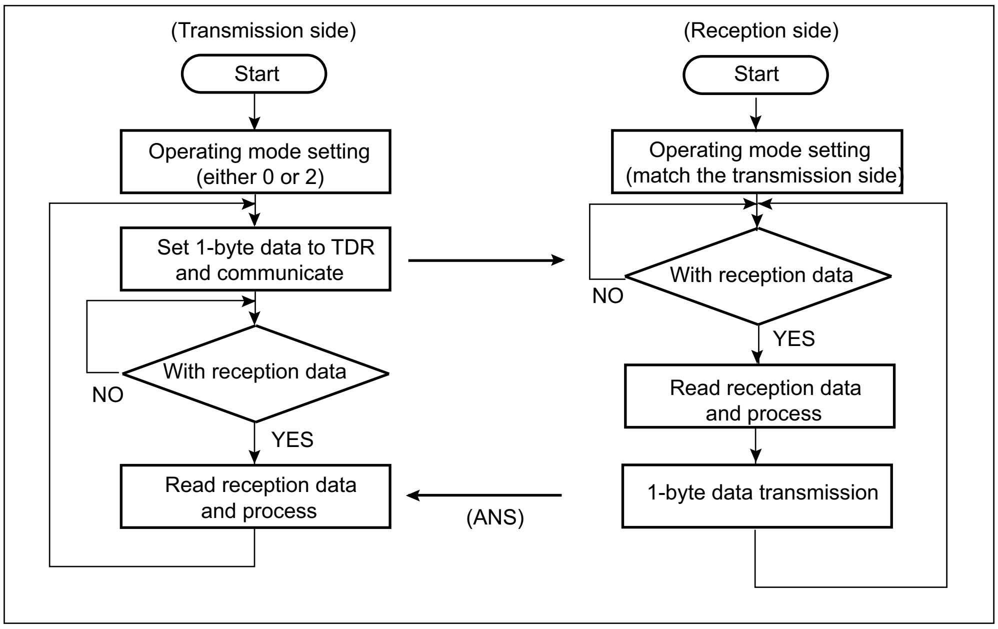

(Transmission side)
Start
Operating mode setting (either 0 or 2)
Set 1-byte data to TDR and communicate
With reception data
NO
YES
Read reception data and process
(ANS)
(Reception side)
Start
Operating mode setting (match the transmission side)
With reception data
NO
YES
Read reception data and process
1-byte data transmission

Figure 6-43: Example of master-slave communication flowchart

6.6.4.12 Master-Slave Communication Function (Multiprocessor Mode)

LIN-USART communication with multiple devices connected in master-slave mode is available for both master or slave systems.

6.6.4.12.1 Master-Slave Communication Function

The settings shown in Figure 6-44 are required to operate LIN-USART in multiprocessor mode (operation mode 1).

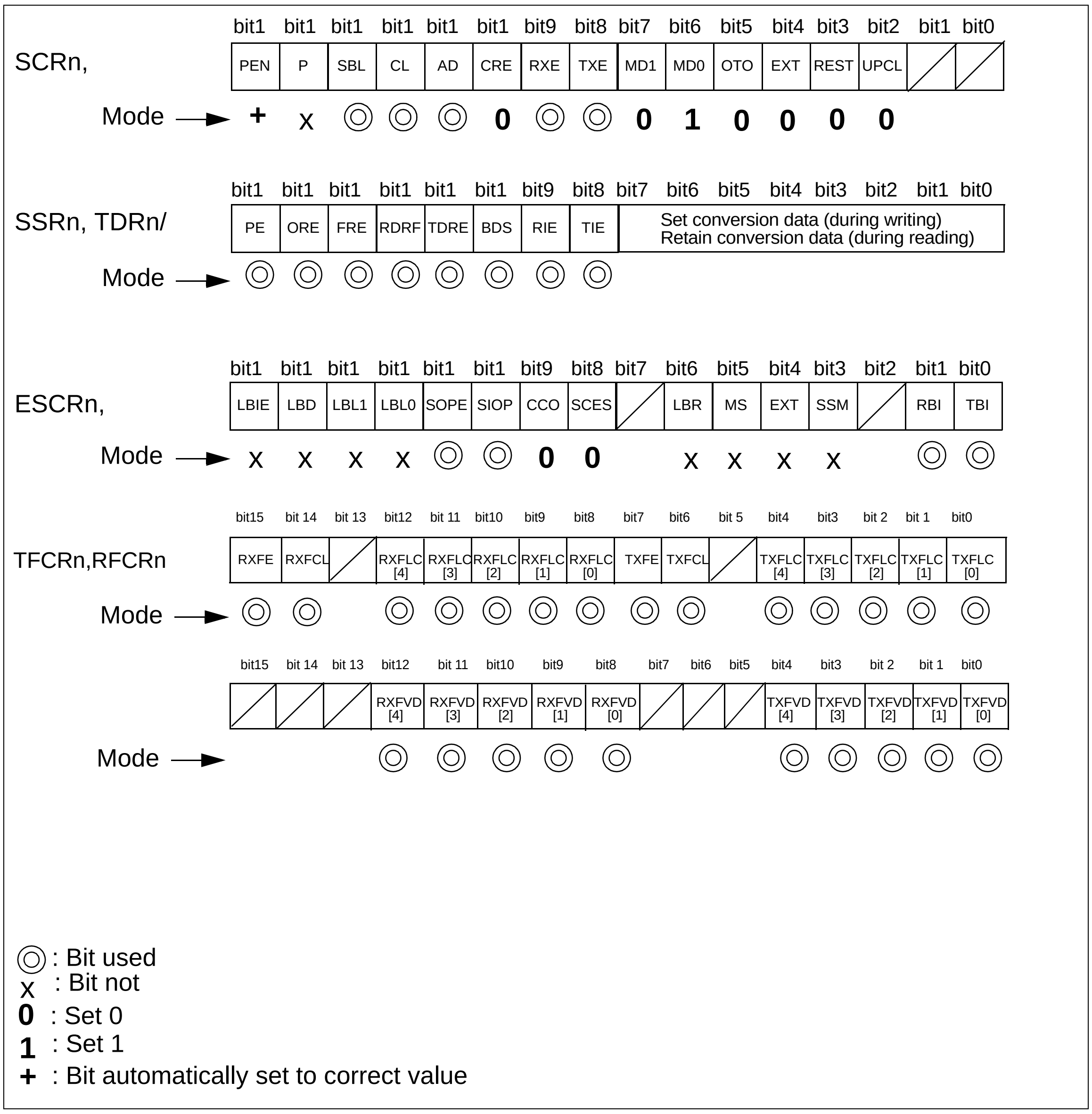

SCRn,
bit1 bit1 bit1 bit1 bit1 bit1 bit9 bit8 bit7 bit6 bit5 bit4 bit3 bit2 bit1 bit0
PEN P SBL CL AD CRE RXE TXE MD1 MD0 OTO EXT REST UPCL
Mode + x ◎ ◎ ◎ 0 ◎ ◎ 0 1 0 0 0 0
SSRn, TDRn/
bit1 bit1 bit1 bit1 bit1 bit1 bit9 bit8 bit7 bit6 bit5 bit4 bit3 bit2 bit1 bit0
PE ORE FRE RDRF TDRE BDS RIE TIE
Set conversion data (during writing)
Retain conversion data (during reading)
Mode ◎ ◎ ◎ ◎ ◎ ◎ ◎ ◎
ESCRn,
bit1 bit1 bit1 bit1 bit1 bit1 bit9 bit8 bit7 bit6 bit5 bit4 bit3 bit2 bit1 bit0
LBIE LBD LBL1 LBL0 SOPE SIOP CCO SCES LBR MS EXT SSM RBI TBI
Mode x x x x ◎ ◎ 0 0 x x x x ◎ ◎
TFCRn,RFCRn
bit15 bit 14 bit 13 bit12 bit 11 bit10 bit9 bit8 bit7 bit6 bit 5 bit4 bit3 bit 2 bit 1 bit0
RXFE RXFCL RXFLC[4] RXFLC[3] RXFLC[2] RXFLC[1] RXFLC[0] TXFE TXFCL TXFLC[4] TXFLC[3] TXFLC[2] TXFLC[1] TXFLC[0]
Mode ◎ ◎ ◎ ◎ ◎ ◎ ◎ ◎ ◎ ◎ ◎ ◎ ◎ ◎
bit15 bit 14 bit 13 bit12 bit 11 bit10 bit9 bit8 bit7 bit6 bit5 bit4 bit3 bit 2 bit 1 bit0
RXFVD[4] RXFVD[3] RXFVD[2] RXFVD[1] RXFVD[0] TXFVD[4] TXFVD[3] TXFVD[2] TXFVD[1] TXFVD[0]
Mode ◎ ◎ ◎ ◎ ◎ ◎ ◎ ◎ ◎ ◎
◎ : Bit used
x : Bit not
0 : Set 0
1 : Set 1
+ : Bit automatically set to correct value

Figure 6-44: Settings for LIN-USART operation mode 1

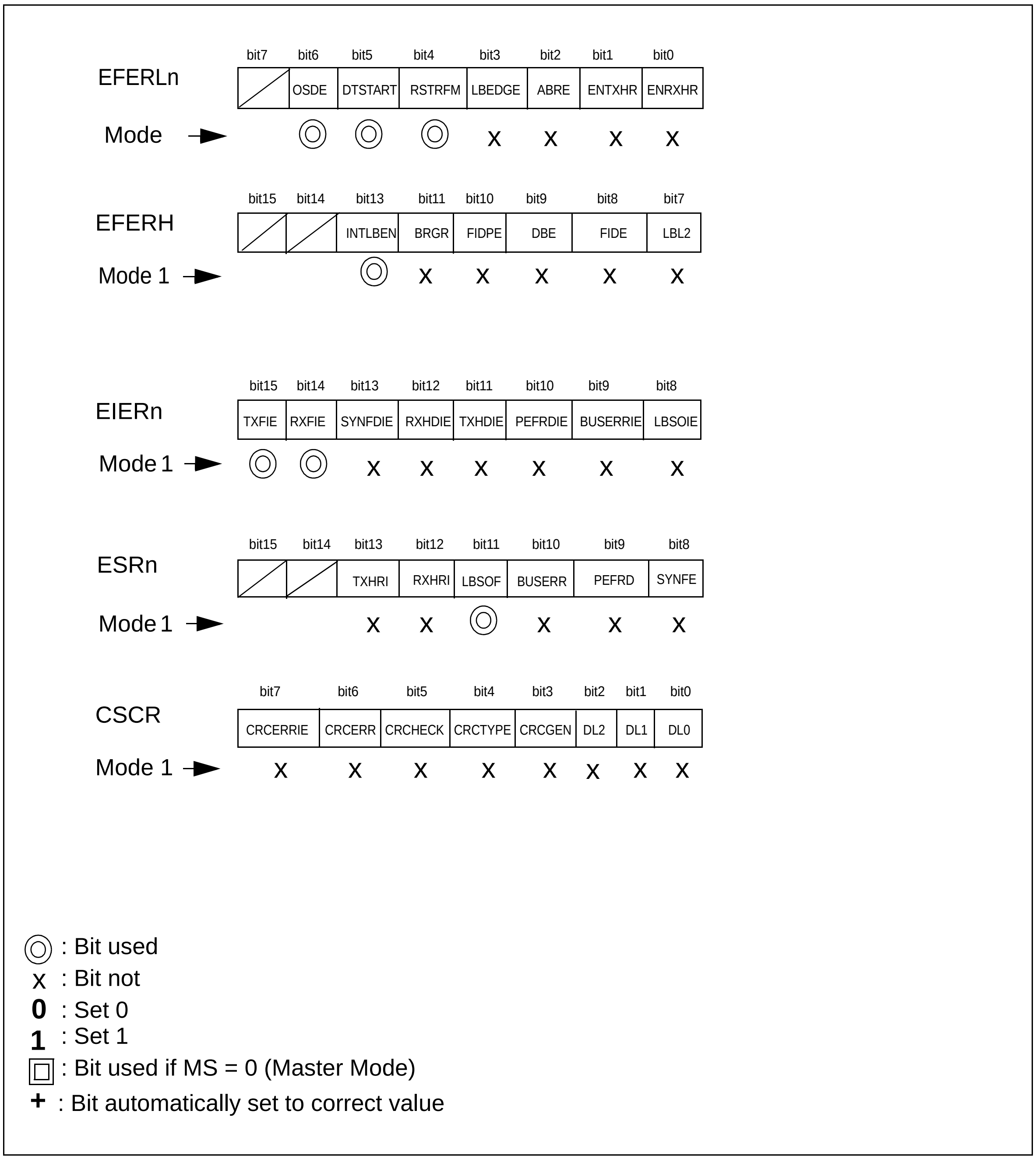

EFERLn
bit7 bit6 bit5 bit4 bit3 bit2 bit1 bit0
OSDE DTSTART RSTRFM LBEDGE ABRE ENTXHR ENRXHR
Mode
◎ ◎ ◎ x x x x
EFERH
bit15 bit14 bit13 bit11 bit10 bit9 bit8 bit7
INTLBEN BRGR FIDPE DBE FIDE LBL2
Mode 1
◎ x x x x x
EIERn
bit15 bit14 bit13 bit12 bit11 bit10 bit9 bit8
TXFIE RXFIE SYNFDIE RXHDIE TXHDIE PEFRDIE BUSERRIE LBSOIE
Mode 1
◎ ◎ x x x x x x
ESRn
bit15 bit14 bit13 bit12 bit11 bit10 bit9 bit8
TXHRI RXHRI LBSOF BUSERR PEFRD SYNFE
Mode 1
x x ◎ x x x
CSCR
bit7 bit6 bit5 bit4 bit3 bit2 bit1 bit0
CRCERRIE CRCERR CRCHECK CRCTYPE CRCGEN DL2 DL1 DL0
Mode 1
x x x x x x x x
◎ : Bit used
x : Bit not
0 : Set 0
1 : Set 1
▢ : Bit used if MS = 0 (Master Mode)
+ : Bit automatically set to correct value

Figure 6-45: Settings for LIN-USART operation mode 1 (Contd.)

6.6.4.12.2 Inter-CPU Connection

As shown in Figure 6-46 on page 82, a communication system consists of one master CPU and multiple slave CPUs connected to two communication lines. LIN-USART can be used for the master or slave CPU.

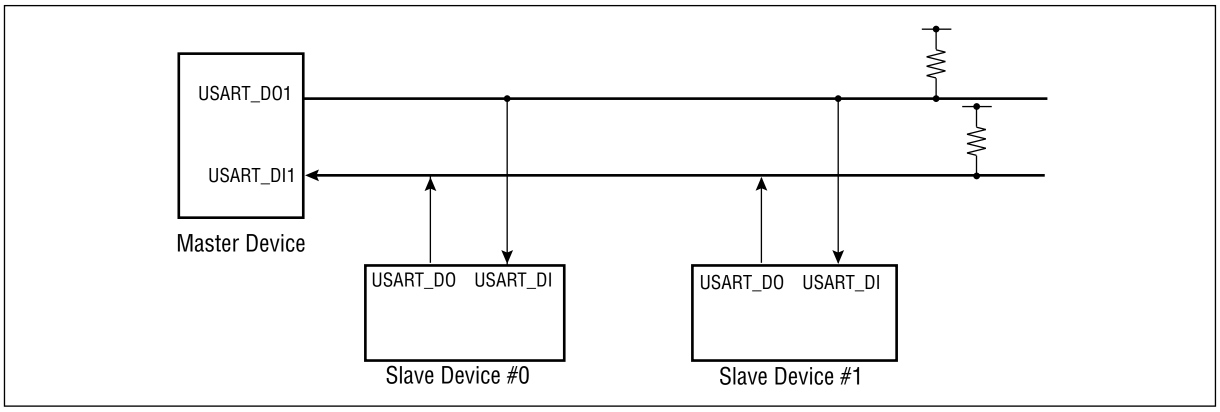


Figure 6-46: Connection example of LIN-USART master-slave communication

6.6.4.12.3 Function Selection

Select the operation mode and data transfer mode for master-slave communication as shown in Table 6-18:.

Table 6-18: Selection of the master-slave communication function

	Operation mode		Data	Parity	Synchronization method	Stop bit	Bit direction
	Master CPU	Slave CPU					
Address transmission and reception	Mode 1 (transmit/ receive AD-bit)	Mode 1 (transmit/ receive AD-bit)	AD = ‘1’ + 7- or 8-bit address	None	Asynchronous	1 or 2-bits	LSB or MSB first
Data transmission and reception			AD = ‘0’ + 7- or 8-bit data				

Communication Procedure

When the master CPU transmits address data, communication starts. The A/D bit in the address data is set to ‘1’, and the communication destination slave CPU is selected. Each slave CPU checks the address data using a program. When the address data indicates the address assigned to a slave CPU, the slave CPU communicates with the master CPU.

Figure 6-47 shows a flowchart of master-slave communication (multiprocessor mode).

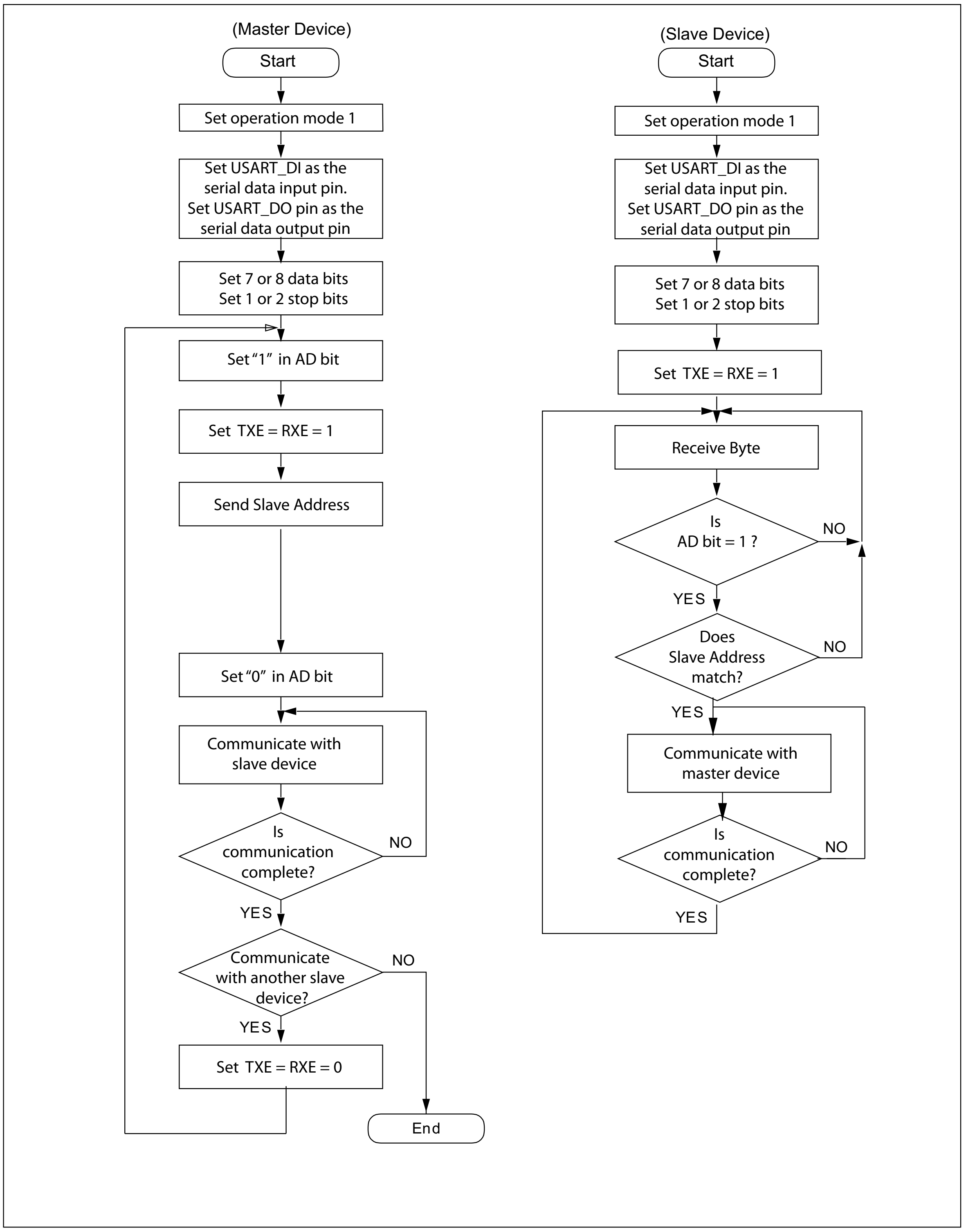

(Master Device)
Start
Set operation mode 1
Set USART_DI as the serial data input pin. Set USART_DO pin as the serial data output pin
Set 7 or 8 data bits Set 1 or 2 stop bits
Set “1” in AD bit
Set TXE = RXE = 1
Send Slave Address
Set “0” in AD bit
Communicate with slave device
Is communication complete?
NO
YES
Communicate with another slave device?
NO
YES
Set TXE = RXE = 0
End
(Slave Device)
Start
Set operation mode 1
Set USART_DI as the serial data input pin. Set USART_DO pin as the serial data output pin
Set 7 or 8 data bits Set 1 or 2 stop bits
Set TXE = RXE = 1
Receive Byte
Is AD bit = 1 ?
NO
YES
Does Slave Address match?
NO
YES
Communicate with master device
Is communication complete?
NO
YES

Figure 6-47: Master-slave communication flowchart

6.6.4.13 LIN Communication Function

LIN-USART communication with LIN devices is available for both LIN master or LIN slave systems.

6.6.4.13.1 LIN Master-slave Communication Function

The settings shown in the following figure are required to operate LIN-USART in LIN communication mode (operation mode 3).

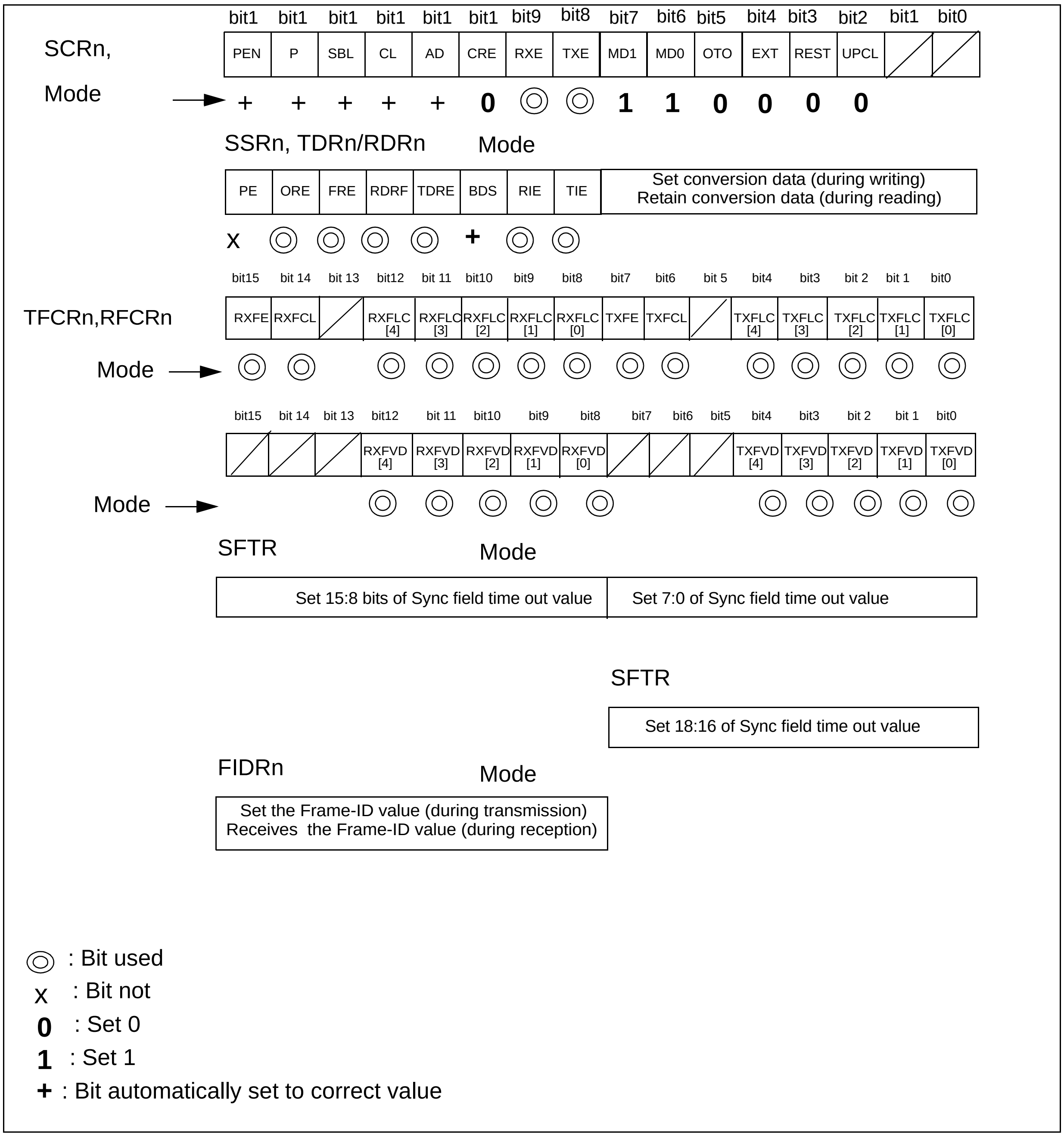

SCRn, Mode
bit1 bit1 bit1 bit1 bit1 bit1 bit9 bit8 bit7 bit6 bit5 bit4 bit3 bit2 bit1 bit0
PEN P SBL CL AD CRE RXE TXE MD1 MD0 OTO EXT REST UPCL
+ + + + + 0 ◎ ◎ 1 1 0 0 0 0
SSRn, TDRn/RDRn Mode
PE ORE FRE RDRF TDRE BDS RIE TIE
Set conversion data (during writing)
Retain conversion data (during reading)
x ◎ ◎ ◎ ◎ + ◎ ◎
TFCRn,RFCRn
bit15 bit 14 bit 13 bit12 bit 11 bit10 bit9 bit8 bit7 bit6 bit 5 bit4 bit3 bit 2 bit 1 bit0
RXFE RXFCL RXFLC [4] RXFLC [3] RXFLC [2] RXFLC [1] RXFLC [0] TXFE TXFCL TXFLC [4] TXFLC [3] TXFLC [2] TXFLC [1] TXFLC [0]
Mode ◎ ◎ ◎ ◎ ◎ ◎ ◎ ◎ ◎ ◎ ◎ ◎ ◎ ◎
bit15 bit 14 bit 13 bit12 bit 11 bit10 bit9 bit8 bit7 bit6 bit5 bit4 bit3 bit 2 bit 1 bit0
RXFVD [4] RXFVD [3] RXFVD [2] RXFVD [1] RXFVD [0] TXFVD [4] TXFVD [3] TXFVD [2] TXFVD [1] TXFVD [0]
Mode ◎ ◎ ◎ ◎ ◎ ◎ ◎ ◎ ◎ ◎
SFTR Mode
Set 15:8 bits of Sync field time out value
Set 7:0 of Sync field time out value
SFTR
Set 18:16 of Sync field time out value
FIDRn Mode
Set the Frame-ID value (during transmission)
Receives the Frame-ID value (during reception)
◎ : Bit used
x : Bit not
0 : Set 0
1 : Set 1
+ : Bit automatically set to correct value

Figure 6-48: Settings for LIN-USART in operation mode 3 (LIN)

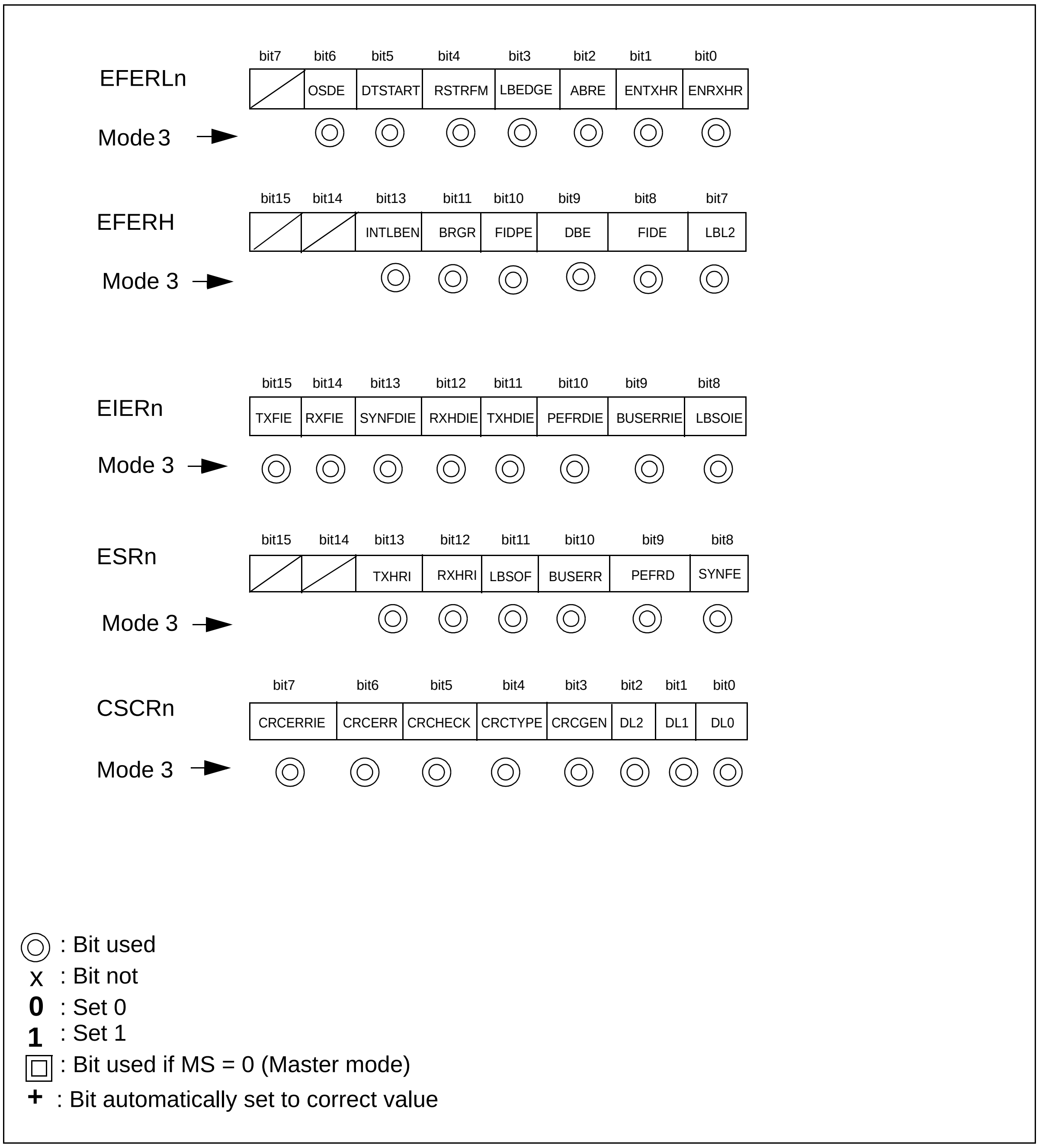

EFERLn
bit7 bit6 bit5 bit4 bit3 bit2 bit1 bit0
OSDE DTSTART RSTRFM LBEDGE ABRE ENTXHR ENRXHR
Mode 3
EFERH
bit15 bit14 bit13 bit11 bit10 bit9 bit8 bit7
INTLBEN BRGR FIDPE DBE FIDE LBL2
Mode 3
EIERn
bit15 bit14 bit13 bit12 bit11 bit10 bit9 bit8
TXFIE RXFIE SYNFDIE RXHDIE TXHDIE PEFRDIE BUSERRIE LBSOIE
Mode 3
ESRn
bit15 bit14 bit13 bit12 bit11 bit10 bit9 bit8
TXHRI RXHRI LBSOF BUSERR PEFRD SYNFE
Mode 3
CSCRn
bit7 bit6 bit5 bit4 bit3 bit2 bit1 bit0
CRCERRIE CRCERR CRCHECK CRCTYPE CRCGEN DL2 DL1 DL0
Mode 3
◎ : Bit used
x : Bit not
0 : Set 0
1 : Set 1
▣ : Bit used if MS = 0 (Master mode)
+ : Bit automatically set to correct value

Figure 6-49: :Settings for LIN-USART in operation mode 3 (Continued)

LIN Device Connection

As shown in the following figure, a communication system of one LIN-master device and a LIN-slave device. LIN-USART can operate both as LIN-master or LIN-slave.

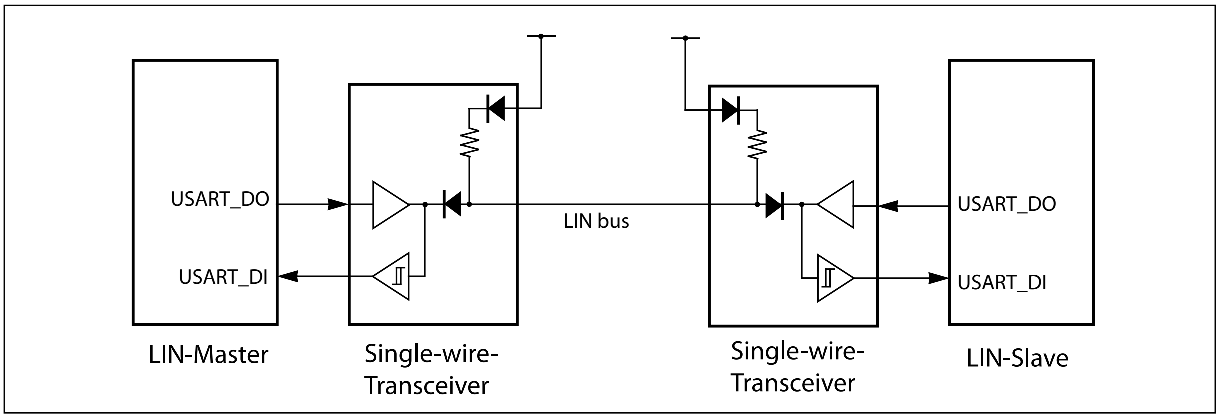


Figure 6-50: Connection example of a small LIN-bus system

6.6.4.14 Flowcharts for LIN-USART in LIN Communication (Operation Mode 3)

This section contains sample flowcharts for LIN-USART in LIN communication.

6.6.4.14.1 LIN-USART as Master Device

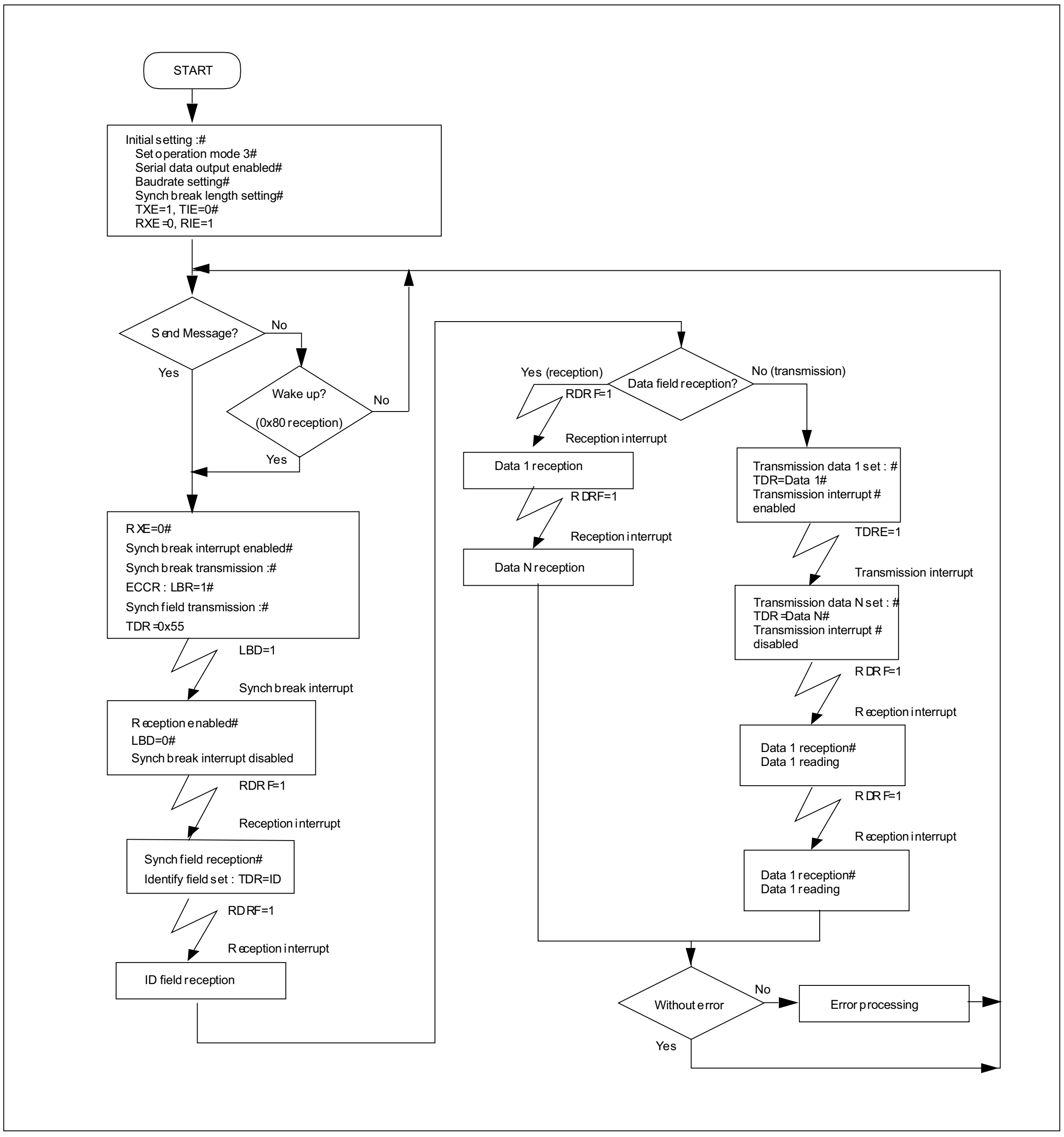
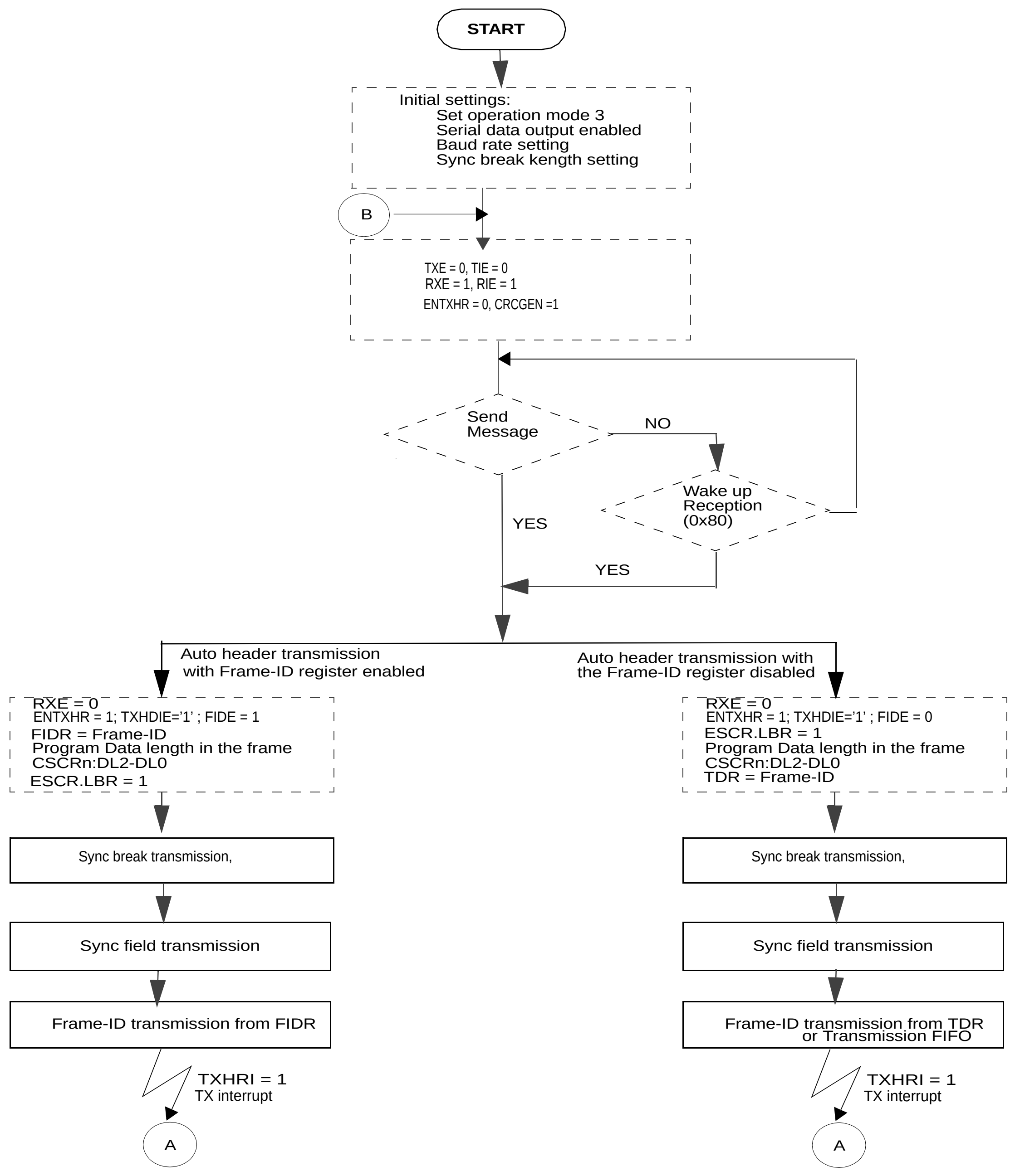


Figure 6-51: LIN-USART LIN master flow chart

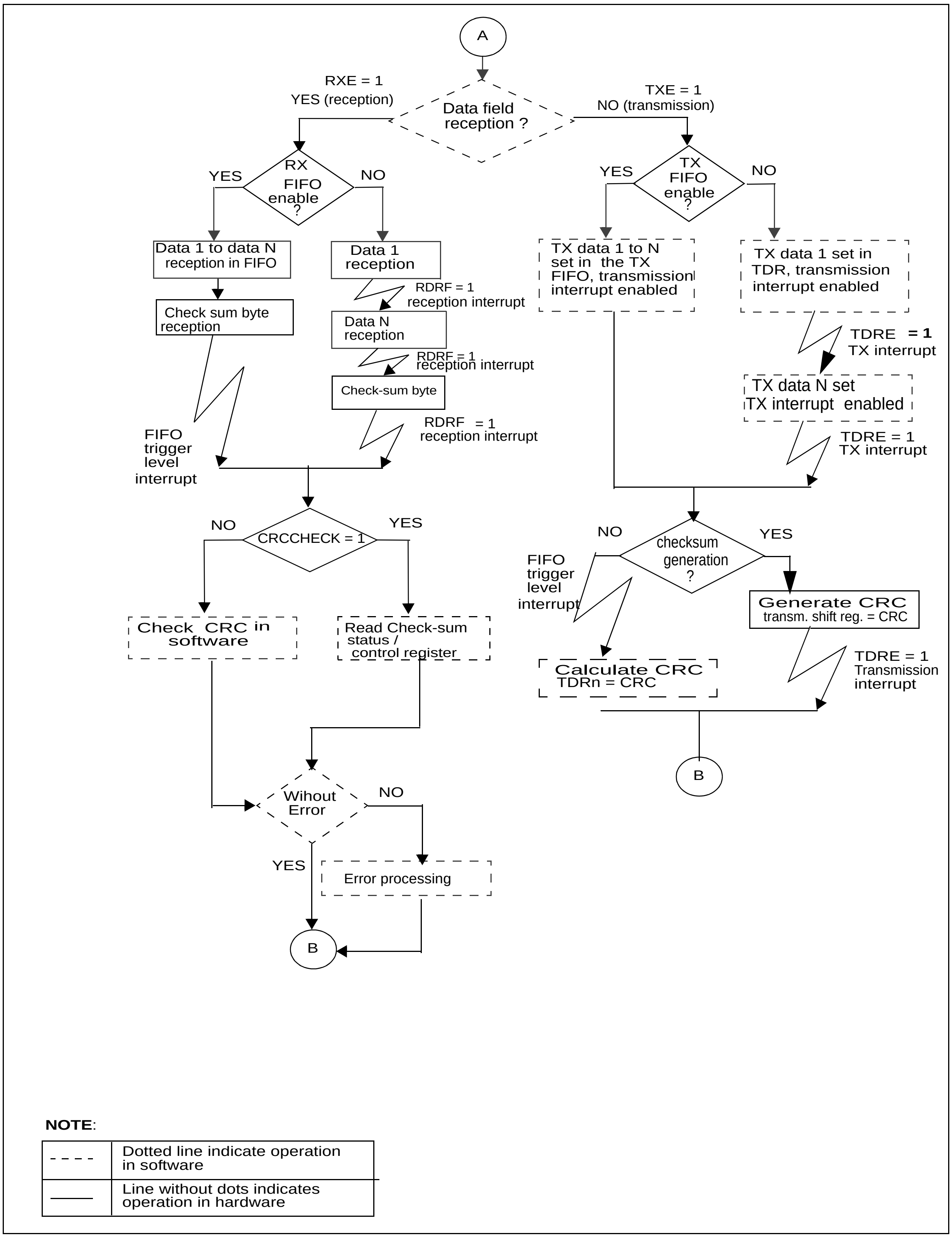
6.6.4.14.2 LIN-USART as Master Device with Additional Features


START
Initial settings:
Set operation mode 3
Serial data output enabled
Baud rate setting
Sync break kength setting
B
TXE = 0, TIE = 0
RXE = 1, RIE = 1
ENTXHR = 0, CRCGEN =1
Send Message
NO
Wake up Reception (0x80)
YES
YES
Auto header transmission with Frame-ID register enabled
Auto header transmission with the Frame-ID register disabled
RXE = 0
ENTXHR = 1; TXHDIE='1' ; FIDE = 1
FIDR = Frame-ID
Program Data length in the frame CSCRn:DL2-DL0
ESCR.LBR = 1
RXE = 0
ENTXHR = 1; TXHDIE='1' ; FIDE = 0
ESCR.LBR = 1
Program Data length in the frame CSCRn:DL2-DL0
TDR = Frame-ID
Sync break transmission,
Sync break transmission,
Sync field transmission
Sync field transmission
Frame-ID transmission from FIDR
Frame-ID transmission from TDR or Transmission FIFO
TXHRI = 1
TX interrupt
TXHRI = 1
TX interrupt
A
A

NOTE:

- - - -	Dotted line indicate operation in software
———	Line without dots indicates operation in hardware

Figure 6-52: LIN-USART as master device flow chart with all additional features used


A
RXE = 1
YES (reception)
Data field reception ?
TXE = 1
NO (transmission)
RX FIFO enable ?
YES
NO
TX FIFO enable ?
YES
NO
Data 1 to data N reception in FIFO
Data 1 reception
RDRF = 1 reception interrupt
Check sum byte reception
Data N reception
RDRF = 1 reception interrupt
Check-sum byte
RDRF = 1 reception interrupt
FIFO trigger level interrupt
TX data 1 to N set in the TX FIFO, transmission interrupt enabled
TX data 1 set in TDR, transmission interrupt enabled
TDRE = 1 TX interrupt
TX data N set TX interrupt enabled
TDRE = 1 TX interrupt
CRCCHECK = 1
NO
YES
Check CRC in software
Read Check-sum status / control register
checksum generation ?
NO
YES
FIFO trigger level interrupt
Generate CRC transm. shift reg. = CRC
Calculate CRC TDRn = CRC
TDRE = 1 Transmission interrupt
B
Wihout Error
NO
YES
Error processing
B

NOTE:

- - - -	Dotted line indicate operation in software
——	Line without dots indicates operation in hardware

Figure 6-53: LIN-USART as a master device with additional features (Contd.)

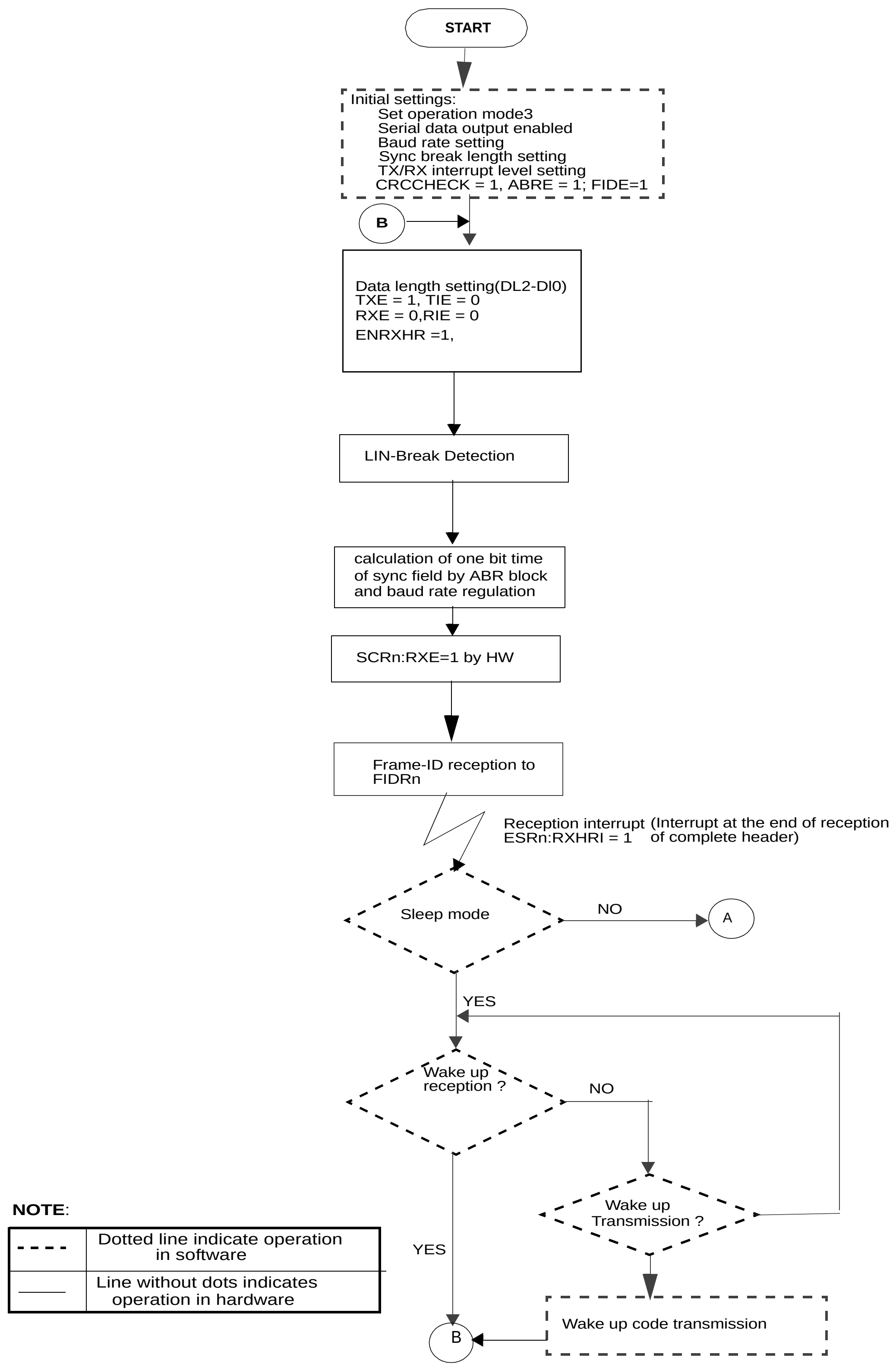

START
Initial settings:
Set operation mode3
Serial data output enabled
Baud rate setting
Sync break length setting
TX/RX interrupt level setting
CRCCHECK = 1, ABRE = 1; FIDE=1
B
Data length setting(DL2-Dl0)
TXE = 1, TIE = 0
RXE = 0,RIE = 0
ENRXHR =1,
LIN-Break Detection
calculation of one bit time of sync field by ABR block and baud rate regulation
SCRn:RXE=1 by HW
Frame-ID reception to FIDRn
Reception interrupt ESRn:RXHRI = 1
(Interrupt at the end of reception of complete header)
Sleep mode
NO
A
YES
Wake up reception ?
NO
Wake up Transmission ?
YES
Wake up code transmission
B
NOTE:
Dotted line indicate operation in software
Line without dots indicates operation in hardware

Figure 6-54: LIN-USART as slave device with all additional features used

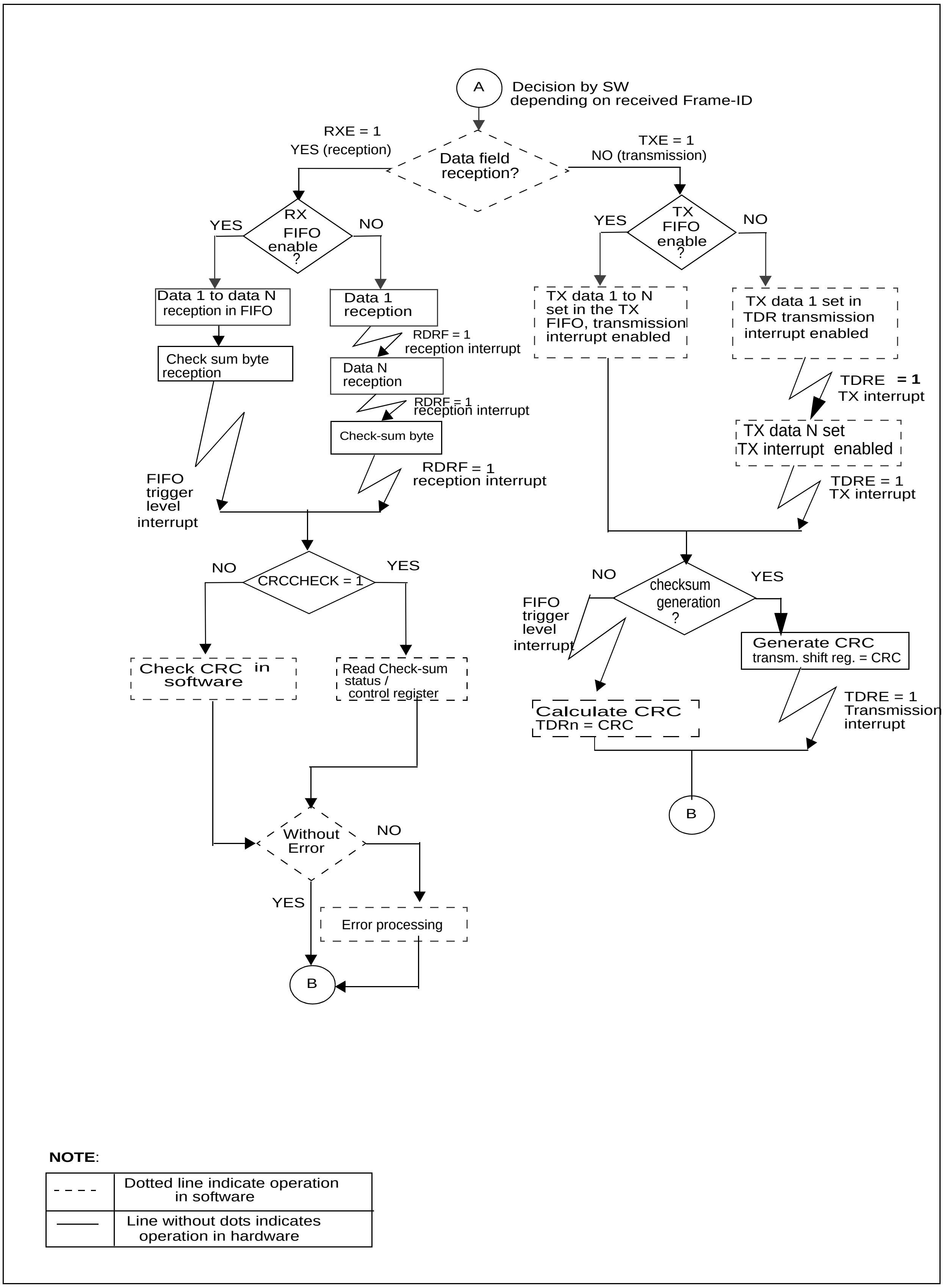

A
Decision by SW depending on received Frame-ID
RXE = 1
YES (reception)
Data field reception?
TXE = 1
NO (transmission)
RX FIFO enable ?
YES
NO
TX FIFO enable ?
YES
NO
Data 1 to data N reception in FIFO
Check sum byte reception
Data 1 reception
RDRF = 1 reception interrupt
Data N reception
RDRF = 1 reception interrupt
Check-sum byte
RDRF = 1 reception interrupt
FIFO trigger level interrupt
TX data 1 to N set in the TX FIFO, transmission interrupt enabled
TX data 1 set in TDR transmission interrupt enabled
TDRE = 1 TX interrupt
TX data N set TX interrupt enabled
TDRE = 1 TX interrupt
CRCCHECK = 1
NO
YES
Check CRC in software
Read Check-sum status / control register
checksum generation ?
NO
YES
FIFO trigger level interrupt
Generate CRC transm. shift reg. = CRC
Calculate CRC TDRn = CRC
TDRE = 1 Transmission interrupt
B
Without Error
NO
YES
Error processing
B
NOTE:
- - - - Dotted line indicate operation in software
—— Line without dots indicates operation in hardware

Figure 6-55: LIN-USART as slave device with all additional features used (Contd.)

6.6.5 Important Notes on Using LIN-USART

The following section describes notes on using LIN-USART.

6.6.5.1 Enabling Operation

In LIN-USART, the Control Register (**SCR**n) has TXE (transmission) and RXE (reception) operation enable bits. Both, data transmission and reception operations, must be enabled before the communication starts because they have been disabled as the default value (initial value). The operation can also be canceled by disabling these bits.

Automatic LIN header transmission and reception in mode 3 are independent of TXE and RXE.

6.6.5.2 Auto Header Detection in LIN Mode

During reception of Frame-ID in auto header detection feature, the read to SCR register returns RXE bit as ‘1’, though the SCR[1] is written with the value of ‘0’.

6.6.5.3 Communication Mode Setting

Set the communication mode while the system is not operating. If the mode is changed during transmission or reception, the transmission or reception is stopped and possible data will be lost.

6.6.5.4 Transmission Interrupt Enabling Timing

The default (initial value) of the Transmission Data Empty Flag bit (SSRn:TDRE) is ‘1’ (no transmission data and transmission data write enable state). A transmission interrupt request is generated as soon as the transmission interrupt request is enabled (SSRn:TIE=1). Ensure to set the TIE flag to ‘1’ after setting the transmission data to avoid an immediate interrupt.

6.6.5.5 Using LIN Operation Mode 3

The LIN features are available in mode 3, but using mode 3 sets the LIN-USART data format automatically to LIN format (8N1, LSB first).

NOTE The length of the sync break for transmission is variable but for reception it is fixed 11-bit times.

6.6.5.6 Changing Operation Settings

It is strongly recommended to reset the LIN-USART after changing operation settings. Particularly, if (for example) start-/stop-bits added to or removed from the data format.

It is recommended to disable the communication (RXE = ‘0’, TXE = ‘0’), if the LIN-USART setting or mode is changed or the LIN-USART is reset.

6.6.5.7 Using Synchronous Slave Mode without Continuous Clock (ESCRn:CCO = 0)

In synchronous slave mode without continuous clock, the write to Transmission Data Register (**TDR**n) must be done before providing the clock for transmission operation.

The approximate time before in which the data must be written into the TDR, should be greater than half serial clock time period plus one rbus_clk time period (Assuming that it takes one rbus_clk time period for data to be written to the TDR register).

6.6.5.8 Using Transmission/Reception FIFO

FIFO has to be cleared using TFCRn:TXFCL/RFCRn:RXFCL before enabling or disabling the respective FIFO.

6.6.5.9 Using Auto Header Transmission without Enabling Frame-ID Register in LIN Mode

For auto header transmission with the Frame-ID register disabled (EFERHn:FIDE = 0), the below specified order of programming is recommended.

1. EFERLn:ENTXHR = 1
2. ECCRn:LBR = 1
3. TDRn= Frame-ID value

If Frame-ID is written into TDR, before setting ECCRn:LBR =1, data from TDR is transmitted before the LIN break and sync field because it is handled as normal data. This is similar to the flow utilized when auto header transmission is disabled.

6.6.5.10 Using Last Bit Shift Out Interrupt

In all the modes, synchronization of status flag ESRn:LBSOF in bus bridge will add up to the interrupt latency, and also the ISR (Interrupt Service Routine) call will add up to the interrupt latency.

6.6.5.11 LIN Slave Settings

Set the baud rate before receiving the first LIN sync break for the slave operation. Otherwise, duration of the sync break cannot be correctly checked against the minimum requirement of the LIN specification (13 master bit time and 11 slave bit time).

6.6.5.12 Bus Idle Function

The bus idle function cannot be used in synchronous slave mode 2.

6.6.5.13 AD Bit (Serial Control Register (SCRn): address/data type select bit)

Special care has to be taken when using the SCRn:AD bit (address-data-bit for multiprocessor mode 1). Writing to it sets the AD bit for transmission and reading from it returns the status of the AD bit written. SCRn:AD bit can also be set and cleared by the corresponding set/clear bits SCSRn:ADS and SCCRn:ADC. Whereas, ESRn:AD bit is a read-only bit and reading from it returns the AD bit of the last received frame. Writing to ESRn:AD bit is ignored.

6.6.5.14 Clearing Reception Errors

Clearing reception errors resets the reception state machine when EFERn:RSTRFM = 0. Therefore, check any reception errors before the next start-bit or start condition is met, to not disturb any ongoing reception.

If EFERn:RSTRFM = ’1’, reception state machine is not reset by SCRn:CRE.

6.6.5.15 LIN Sync Field Wait State

In mode 3 (LIN operation), the LBD bit in the ESCRn register is set to ‘1’ if the input signal is kept at ‘0’ for more than or equal to 10-bit times. Then the LIN-USART waits for the following sync field to be received. If the LIN-USART is set into this state for other reasons than the sync break, it should be initialized by the software reset (SMRn:UPCL=1).

In mode 3, LIN-Break Detection is always working in the background and is level sensitive or edge sensitive depending on EFERn:LBEDGE. Be careful in case of a bus error (bus always dominant). The LIN-Break Detection Flag (LBD) will go ‘1’ or stay ‘1’ after each 10.5 bit times when LBEDGE = 0 independent from enabled or disabled LIN-Break Detection interrupt. If you use LIN-Break Detection interrupt, ensure to check and clear always this flag in your reception interrupt handler.

If EFERn:LBEDGE is ’0’ (level sensitive detection of LIN-break start) LIN-Break Detection is restarted with every high level on USART_DI till a valid LIN-sync field has been detected.

If EFERn:LBEDGE is ’1’ (edge sensitive detection of LIN-break start) LIN-Break Detection is restarted with every falling edge on USART_DI till a valid LIN-sync field has been detected.

The following figure shows the behavior of the LIN-Break Detection counter. This figure does not distinguish between EFERn:LBEDGE settings because the behavior in this scenario is the same.

After a LIN-break has been detected by the LIN-USART, the LIN-Break Detection counter must be reset by SMRn:UPCL before it can detect a new LIN-break.

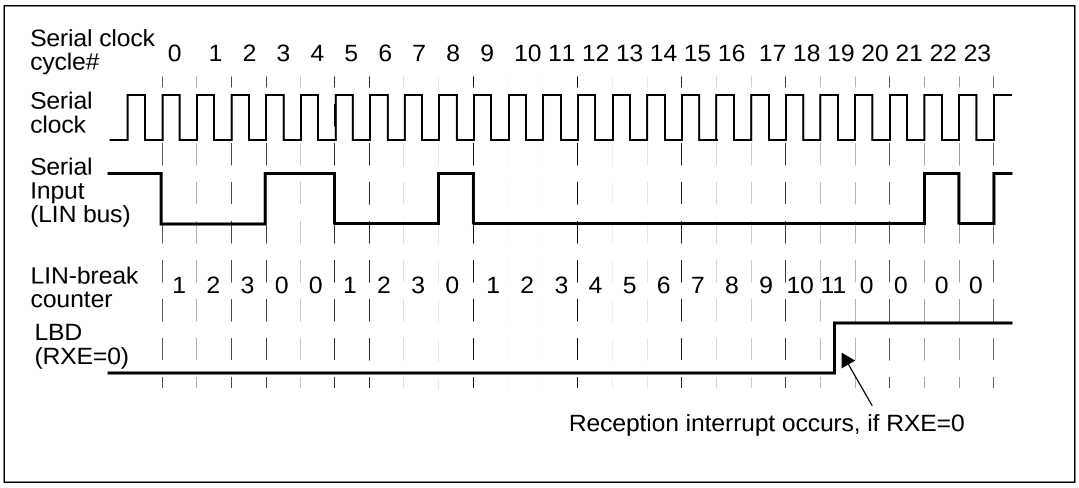


Figure 6-56: LIN Sync Break Detection

6.6.5.15.1 Effects of Reception Errors and CRE Bit

If EFERLn:RSTRFM = 0

CRE resets reception state machine and next falling edge at USART_DI starts reception of new byte. Therefore, either set CRE bit immediately (within half bit time) after receiving errors to prevent data stream de-synchronization or wait an application dependent time after receiving errors and set CRE, when USART_DI is idle.

If EFERLn:RSTRFM = 1

When CRE is set and the register bit EFERLn.RSTRFM is set as ‘1’, reception state machine is not reset.

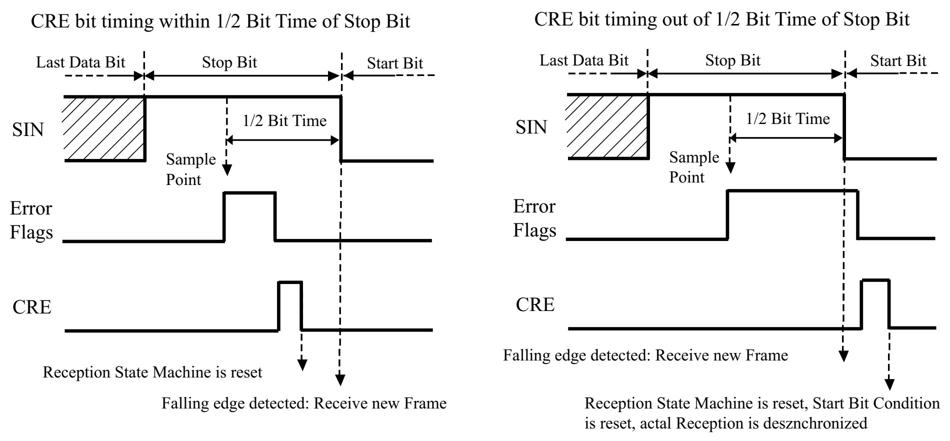

CRE bit timing within 1/2 Bit Time of Stop Bit
Last Data Bit
Stop Bit
Start Bit
SIN
1/2 Bit Time
Sample Point
Error Flags
CRE
Reception State Machine is reset
Falling edge detected: Receive new Frame
CRE bit timing out of 1/2 Bit Time of Stop Bit
Last Data Bit
Stop Bit
Start Bit
SIN
1/2 Bit Time
Sample Point
Error Flags
CRE
Falling edge detected: Receive new Frame
Reception State Machine is reset, Start Bit Condition is reset, actal Reception is desznchronized

Figure 6-57: Timing of the CRE bit (EFERn:RSTRFM = 0)

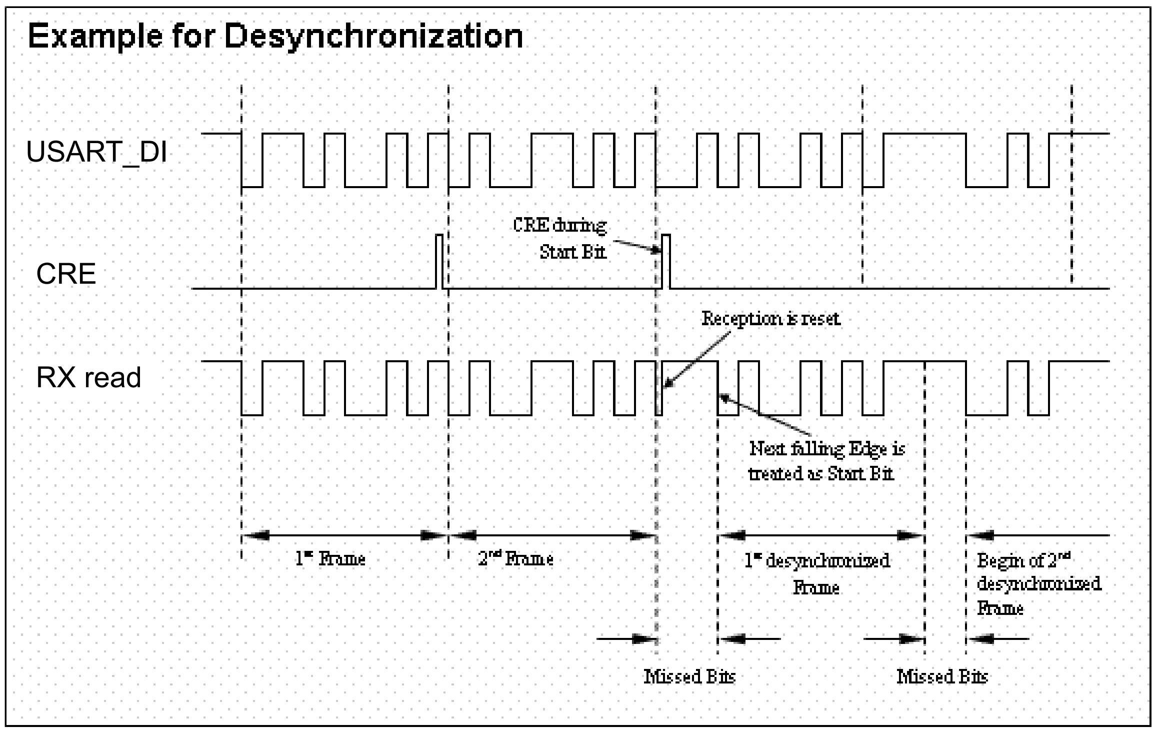


Figure 6-58: Data stream de-synchronization example (EFERn:RSTRFM = 0)

6.6.5.15.2 Start Bit Detection

If EFERLn.DTSTART = 0

In case a framing error occurred (stop bit: USART_DI = '0') and the next start bit (USART_DI = '0') follows immediately, this start bit is recognized regardless of no falling edge before. This is used to keep the LIN-USART synchronized to the data stream and to determine bus always dominant errors (see Figure 6-59 upper figure) by producing next framing errors, if a recessive stop bit is expected. If this behaviour is not wanted, disable the reception temporarily (RXE = 1 -> 0 -> 1) after framing error. In this case, reception goes on at next falling edge on USART_DI. (See Figure 6-59: lower figure).

If EFERLn.DTSTART = '1':

Next falling edge of USART_DI input after FRE is considered as valid start bit.

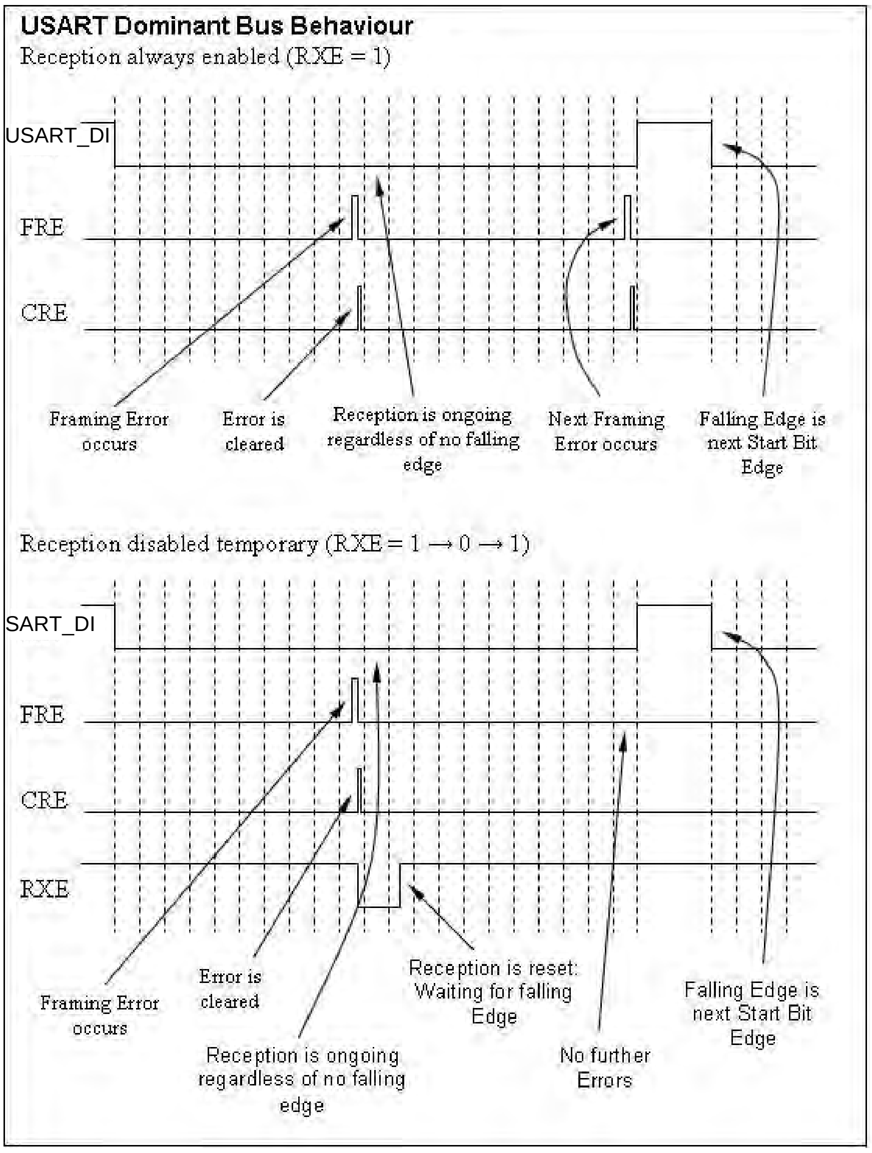

USART Dominant Bus Behaviour
Reception always enabled (RXE = 1)
USART_DI
FRE
CRE
Framing Error occurs
Error is cleared
Reception is ongoing regardless of no falling edge
Next Framing Error occurs
Falling Edge is next Start Bit Edge
Reception disabled temporary (RXE = 1 → 0 → 1)
SART_DI
FRE
CRE
RXE
Framing Error occurs
Error is cleared
Reception is reset: Waiting for falling Edge
Falling Edge is next Start Bit Edge
Reception is ongoing regardless of no falling edge
No further Errors

Figure 6-59: LIN-USART dominant bus behaviour (EFERLn:DTSTART = 0)

6.6.6 Lin-USART Register Overview

Table 6-19: Registers Overview

Base Address(es)	**Instance no 0: BASEADDR="00096000"**	
Absolute Address	**Register Name**	**Register Description**
BASEADDR + 0x0000	USARTn_SMR	Serial Mode Register
BASEADDR + 0x0001	USARTn_SCR	Serial Control Register
BASEADDR + 0x0002	USARTn_SMSR	Serial Mode Set Register
BASEADDR + 0x0003	USARTn_SCSR	Serial Control Set Register
BASEADDR + 0x0004	Reserved	Do not modify
BASEADDR + 0x0005	USARTn_SCCR	Serial Control Clear Register
BASEADDR + 0x0006	USARTn_TDR	Transmission Data Register
BASEADDR + 0x0007	USARTn_SSR	Serial Status Register
BASEADDR + 0x0008	USARTn_RDR	Reception Data Register
BASEADDR + 0x0009	USARTn_SSSR	Serial Status Set Register
BASEADDR + 0x000A	Reserved	Do not modify
BASEADDR + 0x000B	USARTn_SSCR	Serial Status Clear Register
BASEADDR + 0x000C	USARTn_ECCR	Extended Communication Control Register
BASEADDR + 0x000D	USARTn_ESCR	Extended Status/Control Register
BASEADDR + 0x000E	USARTn_ECCSR	Extended Communication Control Set Register
BASEADDR + 0x000F	USARTn_ESCSR	Extended Status/Control Set Register
BASEADDR + 0x0010	USARTn_ECCCR	Extended Communication Control Clear Register
BASEADDR + 0x0011	USARTn_ESCCR	Extended Status/Control Clear Register
BASEADDR + 0x0012	USARTn_ESIR	Extended Serial Interrupt Register
BASEADDR + 0x0013	USARTn_EIER	Extended Interrupt Enable Register
BASEADDR + 0x0014	USARTn_ESISR	Extended Serial Interrupt Set Register
BASEADDR + 0x0015	USARTn_EIESR	Extended Interrupt Enable Set Register
BASEADDR + 0x0016	USARTn_ESICR	Extended Serial Interrupt Clear Register
BASEADDR + 0x0017	USARTn_EIECR	Extended Interrupt Enable Clear Register
BASEADDR + 0x0018	USARTn_EFERL	Extended Feature Enable Register L
BASEADDR + 0x0019	USARTn_EFERH	Extended Feature Enable Register H
BASEADDR + 0x001A	USARTn_RFCR	Reception FIFO Control Register
BASEADDR + 0x001B	USARTn_TFCR	Transmission FIFO Control Register
BASEADDR + 0x001C	USARTn_RFCSR	Reception FIFO Control Set Register
BASEADDR + 0x001D	USARTn_TFCSR	Transmission FIFO Control Set Register
BASEADDR + 0x001E	USARTn_RFCCR	Reception FIFO Control Clear Register
BASEADDR + 0x001F	USARTn_TFCCR	Transmission FIFO Control Clear Register
BASEADDR + 0x0020	USARTn_RFSR	Reception FIFO Status Register
BASEADDR + 0x0021	USARTn_TFSR	Transmission FIFO Status Register
BASEADDR + 0x0022	USARTn_CSCR	Checksum Status and Control Register
BASEADDR + 0x0023	USARTn_ESR	Extended Status Register
BASEADDR + 0x0024	USARTn_CSCSR	Checksum Status and Control Set Register
BASEADDR + 0x0025	Reserved	Do not modify
BASEADDR + 0x0026	USARTn_CSCCR	Checksum Status and Control Clear Register
BASEADDR + 0x0027	USARTn_ESCLR	Extended Status Clear Register
BASEADDR + 0x0028	USARTn_BGRLL	Baud Rate Generation Reload Register L
BASEADDR + 0x0029	USARTn_BGRLM	Baud Rate Generation Reload Register M
BASEADDR + 0x002A	USARTn_BGRLH	Baud Rate Generation Reload Register H
BASEADDR + 0x002B	Reserved	Do not modify
BASEADDR + 0x002C	USARTn_BGRL	Baud Rate Generation Register L
BASEADDR + 0x002D	USARTn_BGRM	Baud Rate Generation Register M
BASEADDR + 0x002E	USARTn_BGRH	Baud Rate Generation Register H

Table 6-19: Registers Overview (Continued)

Base Address(es)	**Instance no 0: BASEADDR="00096000"**	
Absolute Address	**Register Name**	**Register Description**
BASEADDR + 0x002F	Reserved	Do not modify
BASEADDR + 0x0030	USARTn_STXDR	Serial Transmit DMA Configuration Register
BASEADDR + 0x0031	USARTn_SRXDR	Serial Receive DMA Configuration Register
BASEADDR + 0x0032	USARTn_STXDSR	Serial Transmit DMA Configuration Set Register
BASEADDR + 0x0033	USARTn_SRXDSR	Serial Receive DMA Configuration Set Register
BASEADDR + 0x0034	USARTn_STXDCR	Serial Transmit DMA Configuration Clear Register
BASEADDR + 0x0035	USARTn_SRXDCR	Serial Receive DMA Configuration Clear Register
BASEADDR + 0x0036	USARTn_SFTRL	Sync Field Timeout Register L
BASEADDR + 0x0037	USARTn_SFTRM	Sync Field Timeout Register M
BASEADDR + 0x0038	USARTn_SFTRH	Sync Field Timeout Register H
BASEADDR + 0x0039	Reserved	Do not modify
BASEADDR + 0x003A	USARTn_FIDR	Frame-ID Register
BASEADDR + 0x003B	Reserved	Do not modify
BASEADDR + 0x003C	Reserved	Do not modify
BASEADDR + 0x003D	Reserved	Do not modify

6.6.7 LIN-USART Additional Register Information

6.6.7.1 Transmission Data Register (TDRn)

Transmission

When transmission data is written to this register, the transmission data empty flag bit (SSRn:TDRE) is cleared to ‘0’.

When transfer to the Transmission Shift Register is complete and starts, the bit is set to ‘1’. When the TDRE bit is ‘1’, the next part of transmission data can be written. If output transmission interrupt requests have been enabled, a transmission interrupt is generated. Write the next part of transmission data when a transmission interrupt is generated or the TDRE bit is ‘1’.

If the TX FIFO is enabled (TFCRn:TXFE = 1), SSRn:TDRE is set when the number of data in the TX FIFO is less or equal to programmed trigger level in the TFCRn:TXFLC[4:0].

TDRn and TX-FIFO-content is reset to 11111111_b at reset.

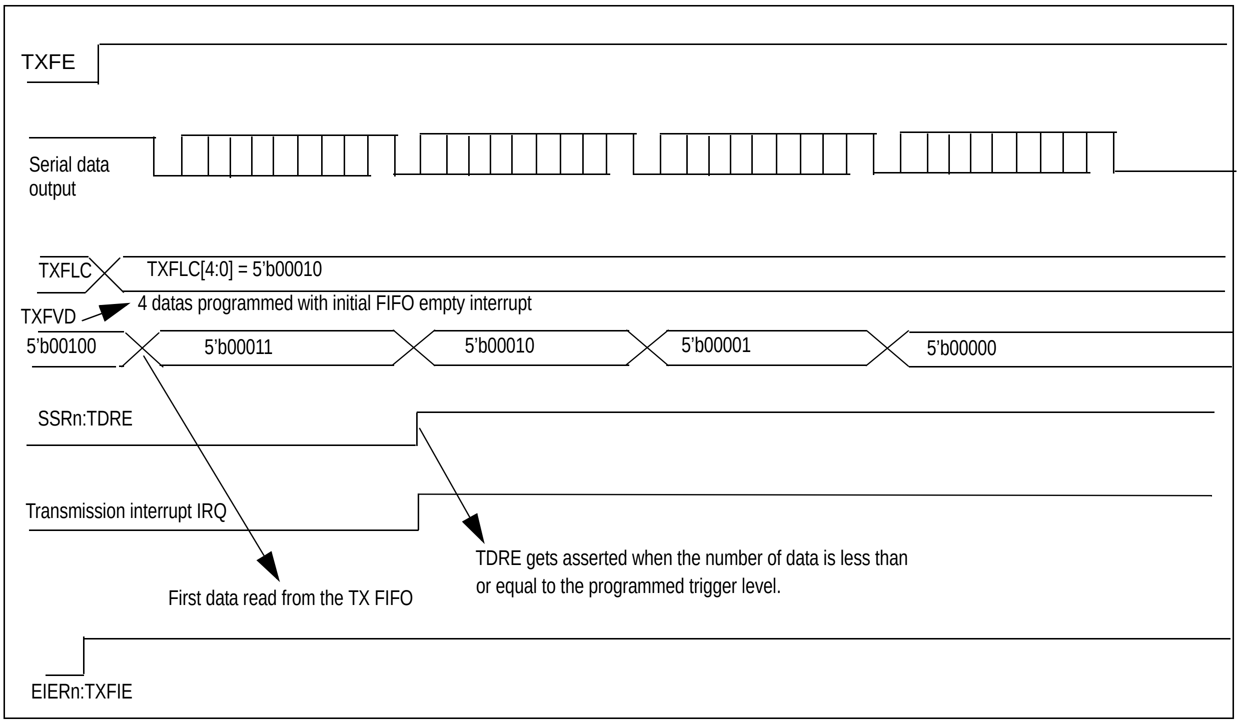

TXFE
Serial data output
TXFLC
TXFLC[4:0] = 5’b00010
TXFVD
4 datas programmed with initial FIFO empty interrupt
5’b00100
5’b00011
5’b00010
5’b00001
5’b00000
SSRn:TDRE
Transmission interrupt IRQ
TDRE gets asserted when the number of data is less than or equal to the programmed trigger level.
First data read from the TX FIFO
EIERn:TXFIE

Figure 6-60: Transmission of LIN-USART with FIFO

6.6.7.2 Reception Data Register (RDRn)

Reception

RDRn is the register that contains reception data. The serial data signal transmitted to the USART_DI pin is converted in the Shift register and stored there. When the data length is 7-bits, the uppermost bit (D[7]) contains 0. When reception is complete the data is stored in this register and the Reception Data Full Flag bit (SSRn:RDRF) is set to ‘1’. If a receive interrupt request is enabled at this point, a receive interrupt occurs.

Read RDRn when SSRn:RDRF bit is ‘1’. SSRn:RDRF bit is cleared automatically when RDRn is read. Also the receive interrupt is cleared if it is enabled and no error has occurred.

Data in RDRn is invalid when a reception error occurs (SSRn: PE or SSRn:ORE or SSRn:FRE is 1).

If the RX FIFO is enabled (RFCRn:RXFE = 1), RDRn contains the next value of the RX FIFO to be read. SSRn:RDRF is set when the number of data in the FIFO is greater or equal to the programmed trigger level in the RFCRn:RXFLC[4:0].

RDRn and RX-FIFO-content is reset to 00000000_b at reset and SMRn:UPCL is ‘1’.

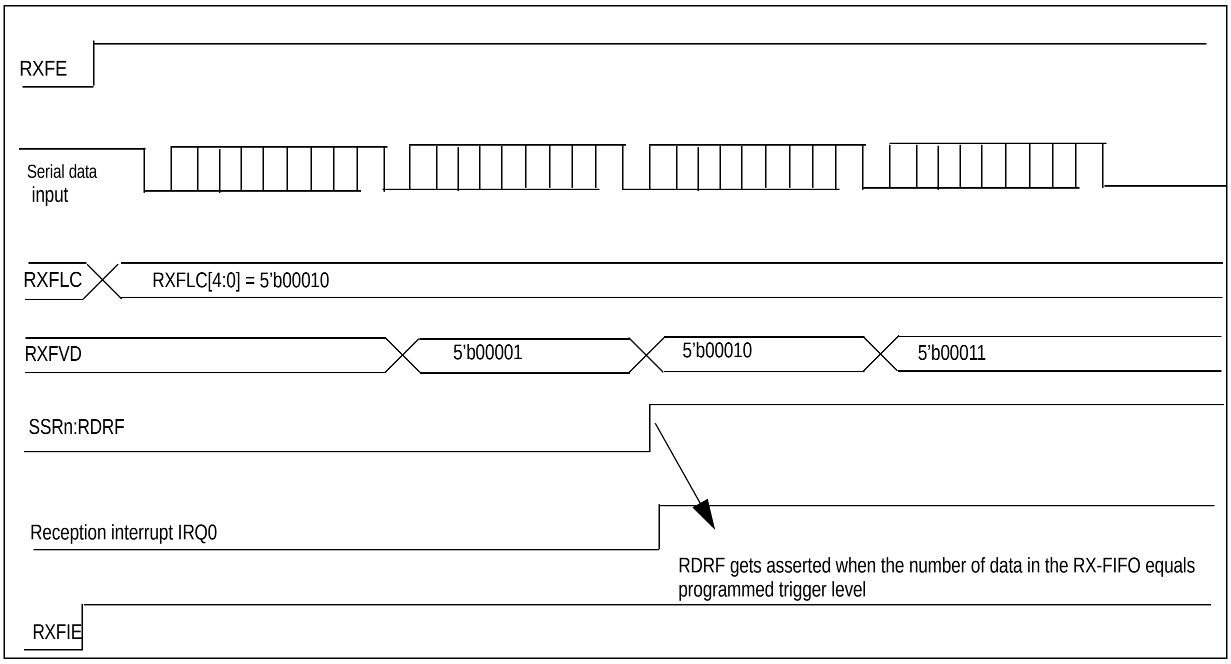


Figure 6-61: Reception of LIN-USART with FIFO

NOTE TDRn is a write-only register and RDRn is a read-only register.

6.6.7.3 Checksum Status and Control Register (CSCRn)

- Checksum Status and Control Register

The checksum calculation is done only in LIN mode. The checksum contains the inverted 8-bit sum with carry over all data bytes or all data bytes and the protected identifier (Frame-ID). The checksum calculation over the data bytes only is called classic checksum. Checksum calculation over both data bytes and the protected identifier (Frame-ID) is called enhanced checksum. Checksum generation and checksum verification are done according to the LIN specification 2.1.

- Checksum enabling

CRCGEN	CRCHECK	Description
0	0	No CRC generation /CRC verification
0	1	CRC check on received data
1	0	CRC generation for transmitting CRC byte
1	1	CRC generated is sent and CRC check on data received back

6.6.7.4 Sync Field Timeout Register - H (SFTRHn)

SFTRn contains 19 bits of timeout value for sync field detection. At the rising edge of USART_DI after LIN break, the Sync field timeout counter starts incrementing. When the Sync field timeout counter value is less than the value programmed in the SFTRn and the fifth falling edge of Sync field is detected, ESRn:SYNFE (timeout error flag) will not be asserted. If the Sync field timeout counter value reaches the Sync field time out value before detecting the fifth falling edge of the Sync field, the ESRn:SYNFE (timeout error flag) is set. This will result in error interrupt if the EIERn:SYNFEIE is set to "1".

SFTRn register can be read only in 32 bit mode. When this register is set to 0x00000, sync field timeout detection is disabled (default).

6.6.7.5 Frame-ID Register (FIDRn)

FIDRn register contains the Frame-ID used for header transmission or reception depending on enabled automatic header transmission or reception.

If LIN-USART is used as LIN-Master and Frame-ID register is enabled by EFERn:FIDE, Frame-ID is sent from this register. The value written to the Frame-ID register must be an 8-bit value including the parity bits of the Frame-ID.

If LIN-USART is used as LIN-Slave and Frame-ID register is enabled by EFERn:FIDE, Frame-ID is stored in this register including the parity bits.

When Frame-ID register is enabled along with FIFO, the Sync field data has to be set in the FIFO. In this case the Sync field value will be transmitted from the transmission FIFO and the Frame-ID data from Frame-ID register. From the start to end of Frame-ID transmission the read to TX FIFO will be halted.

6.7 High-speed SPI Interface (HS_SPI)

The HS_SPI unit provides various operating modes for interfacing to serial peripheral devices that use the de-facto standard SPI protocol. The HS_SPI unit serves up to four SPI targets, but can only serve one at a time (i.e. simultaneous communication with all four devices is not possible). A round-robin mechanism can be used to service all four external targets sequentially. In addition to the legacy SPI mode, the interface can also operate as one dual-bit or one quad-bit SPI interface (whereby the communication bandwidth underlies the restriction of only being able to serve one target at one time!). SPI peripherals or an external Flash can be connected to this interface.

6.7.1 Features of the High Speed SPI Interface

- Supports legacy as-well-as the dual-bit and quad-bit modes of SPI operation
- Supports up to 4 slave devices in master mode
- Programmable transfer rate, active-level of slave-select signal, polarity and phase of the serial clock per Slave Select
- External serial flash and serial SRAM devices can be memory-mapped to the address-space of the MB88F33x ‘Indigo2(-x)’, in “Command Sequencer” Mode
- In “Command Sequencer” Mode, memory accesses initiated by the AHB masters are automatically converted to the serial memory read/write commands
- “Direct” mode allows HS_SPI to be used as a standard SPI through FIFO interface
- Supports SPI clock frequencies up to HCLK/2 or peri_clk/2

6.7.2 Block Diagram

Figure 6-62 shows the block diagram of HS_SPI module

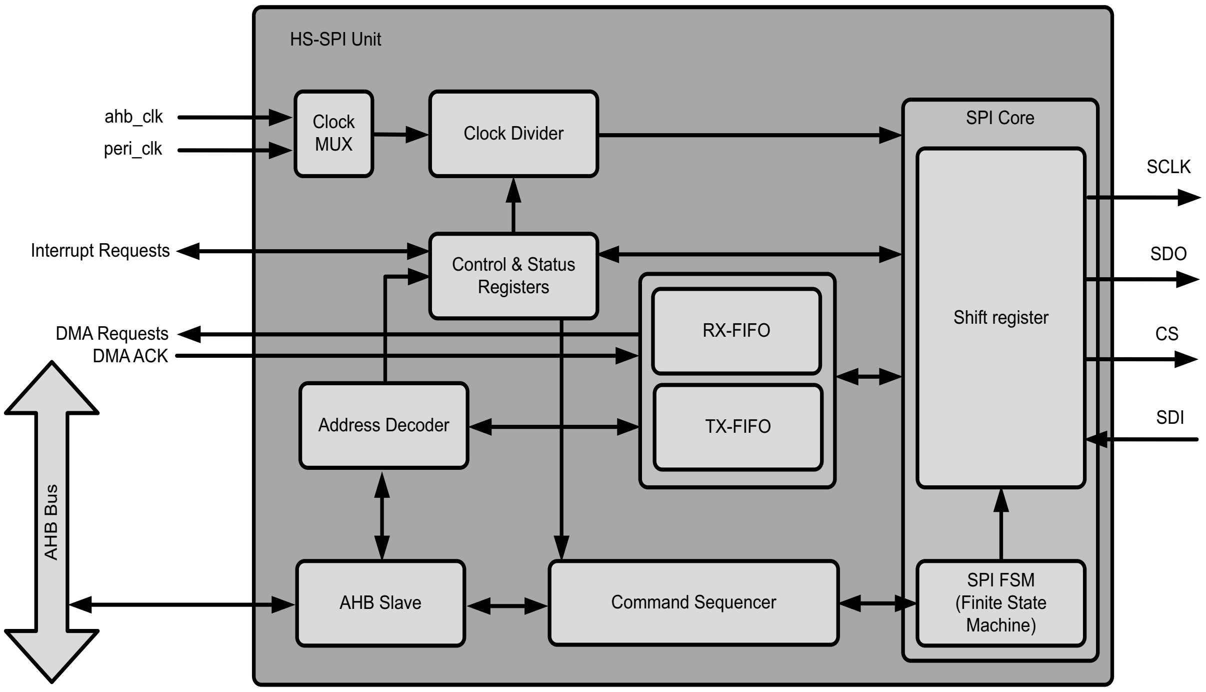


Figure 6-62: HS_SPI Block Diagram

6.7.3 Operation of High-speed SPI Interface

HS_SPI can be configured in one of the two operating modes: ‘Direct Mode’ and ‘Command Sequencer Mode’.

In ‘Direct Mode’ of operation, the software can directly write the data to be transmitted into the TX-FIFO. Similarly, the software can directly read the data received over the serial interface from the RX-FIFO and from the shift Register. The SPI core transfers the data to/from the FIFOs over the serial interface. Based on the configuration in CSR. The ‘Direct Mode’ is described in section “6.7.4 Direct Mode”.

In ‘Command Sequencer Mode’, HS_SPI maps the external serial Flash or serial SRAM devices onto the address-space of the MB88F33x ‘Indigo2(-x)’. Up to 4 serial memory devices can be mapped in this way, one on each of the four Slave-Select outputs. If any AHB master initiates an AHB transfer to access any of the mapped serial-memory device, the HS_SPI initiates serial transfer for the corresponding memory read or write operation. Until the time HS_SPI accesses the external device, the AHB transfer is stalled. The ‘Command Sequencer Mode’ is described in section “6.7.5 Command Sequencer Mode”.

6.7.3.1 Clocking Modes

Based on the programmed values of the HSSPIn_PCC0~3:CPOL, HSSPIn_PCC0~3:CPHA, and HSSPIn_PCC0~3:ACES bits, each peripheral can have up to 8 clocking modes. These bits, along with the HSSPIn_PCC0~3:RTM bits together, decide the serial data input and output timings of HS_SPI, with respect to the serial SPI clock. This is explained in Table 6-20 .

Table 6-20: Clocking Modes

MODE	ACES (Active Clock Edges are Same)	CPOL (Clock Polarity)	CPHA (Clock Phase)	Description
Mode 0	0	0	0	Output data is driven one half-cycle before the first positive edge of the serial clock and on subsequent **negative** edges.
				Input data is sampled on **positive** edges of the serial clock.
Mode 1		0	1	Output data is driven on **positive** edge of the serial clock.
				Input data is sampled on the **negative** edges of the serial clock.
Mode 2		1	0	Output data is driven one half-cycle before the first negative edge of the serial clock and on subsequent **positive** edges.
				Input data is sampled on the **negative** edges of the serial clock.
Mode 3		1	1	Output data is driven on the **negative** edge of the serial clock.
				Input data is sampled on the **positive** edges.

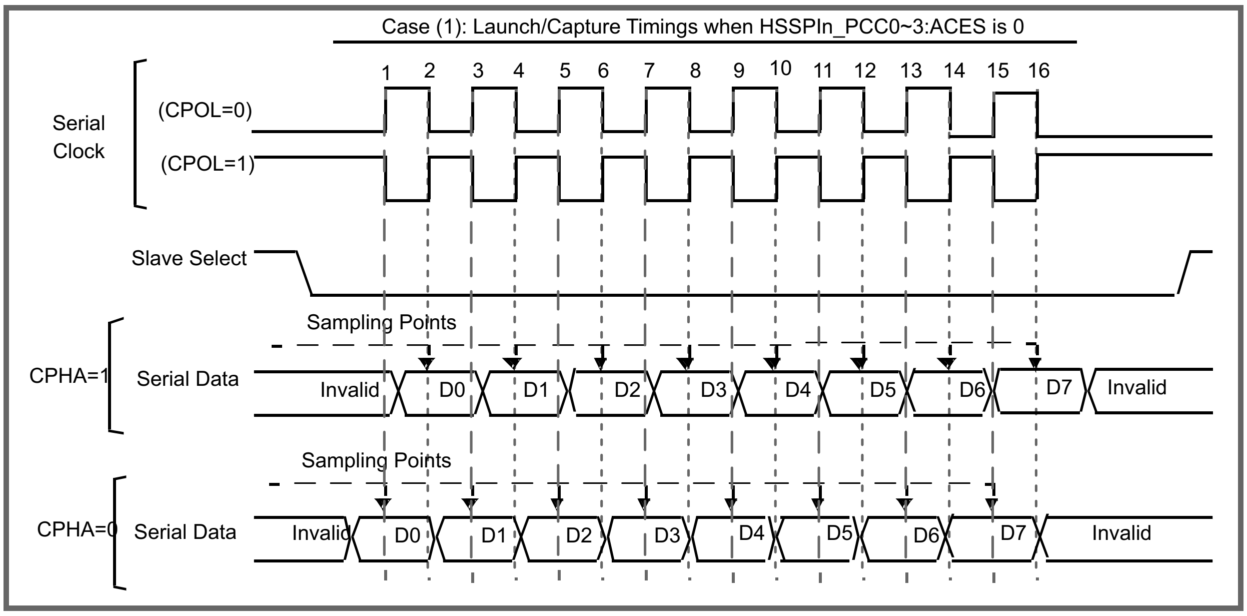
Table 6-20: Clocking Modes (Continued)

MODE	ACES (Active Clock Edges are Same)	CPOL (Clock Polarity)	CPHA (Clock Phase)	Description
Mode 4	1	0	0	Output data is driven one half-cycle before the first positive edge of the serial clock and on subsequent **negative** edges.
				Input data is sampled on **negative** edges of the serial clock.
Mode 5		0	1	Output data is driven on **positive** edge of the serial clock.
				Input data is sampled one half-cycle after the first negative edge of the serial clock and on the subsequent **positive** edges of the serial clock.
Mode 6		1	0	Output data is driven one half-cycle before the first negative edge of the serial clock and on subsequent **positive** edges.
				Input data is sampled on the **positive** edges of the serial clock.
Mode 7		1	1	Output data is driven on the **negative** edge of the serial clock.
				Input data is sampled one half-cycle after the first positive edge of the serial clock and on the subsequent **negative** edges of the serial clock.

Timing waveforms, indicating the serial data and the serial clock, along with the different combinations of ACES, CPOL, and CPHA bits are depicted in Figure 6-63 on page 105.

As indicated in this figure, when HSSPIn_PCC0~3:ACES bit is set, the data driving and sampling points are separated by one complete clock period (as against the case in traditional HSSPIn_PCC0~3:ACES=0 configuration, where the data driving and sampling points are separated only with a half clock period). Thus, when HSSPIn_PCC0~3:ACES is set, the transfer runs for one extra clock cycle. On start of a transfer in receive mode, HS_SPI skips the sampling of data on the 1st sampling point, and actually starts sampling the data from the next sampling point. This skipping of the data on first sampling point is done in order to capture the correct serial data in re-timed mode.

The users shall note here, that when HSSPIn_PCC0~3:ACES is set, if transmission and reception are both enabled simultaneously through the CSRs, then the additional extra clock cycle, inserted at the start of the reception, has a side-effect of also transmitting some data for an extra clock cycle to the SPI Slave that is interfaced with HS_SPI. Therefore, to avoid this extra data transmission when ACES is configured, the users shall disable the transmission while reception is enabled.



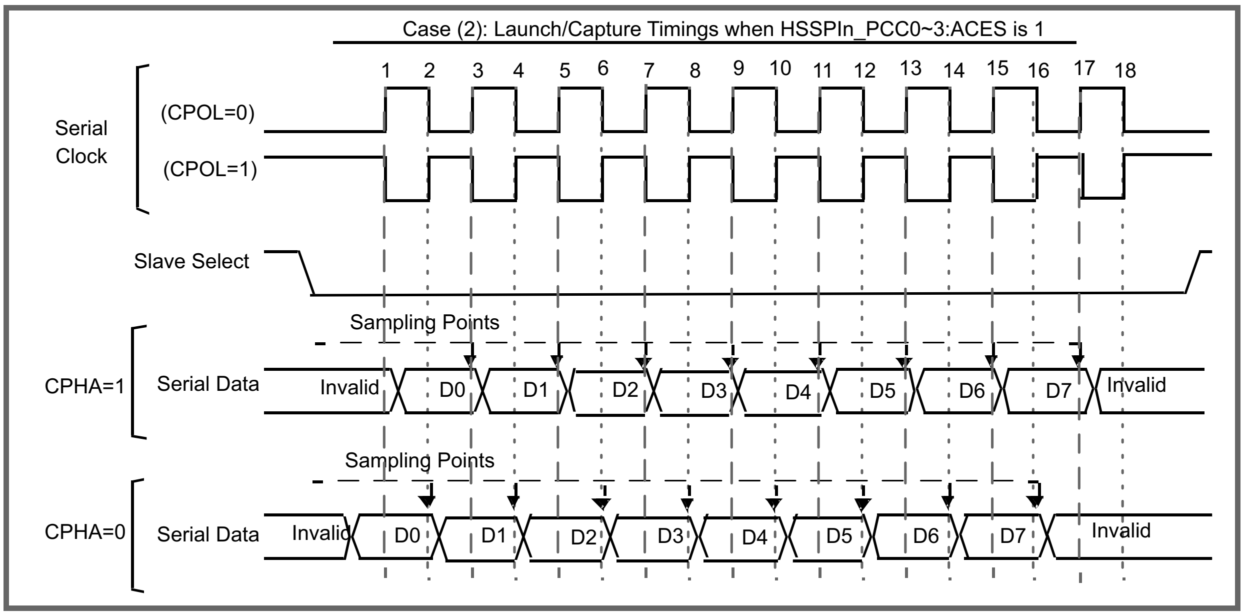


Figure 6-63: Clocking modes of the serial interface clock

As shown in the figure above, when ACES is set to 1, one extra clock cycle is required for the serial data to be correctly captured by the remote device.

6.7.3.2 Re-timed Clock

Some of the Serial Flash devices use SCLK frequencies of 80 MHz (and above), and they leave very tight setup margins, for the serial data to be captured by HS_SPI. When HS_SPI is interfaced with such memory devices, then data setup violations might occur at the registers that are used to capture the serial data input. To capture the valid data read from such serial memories, the re-timed clock mode shall be used. Re-timed clock mode can be set using the HSSPIn_PCC0~3:RTM bit.

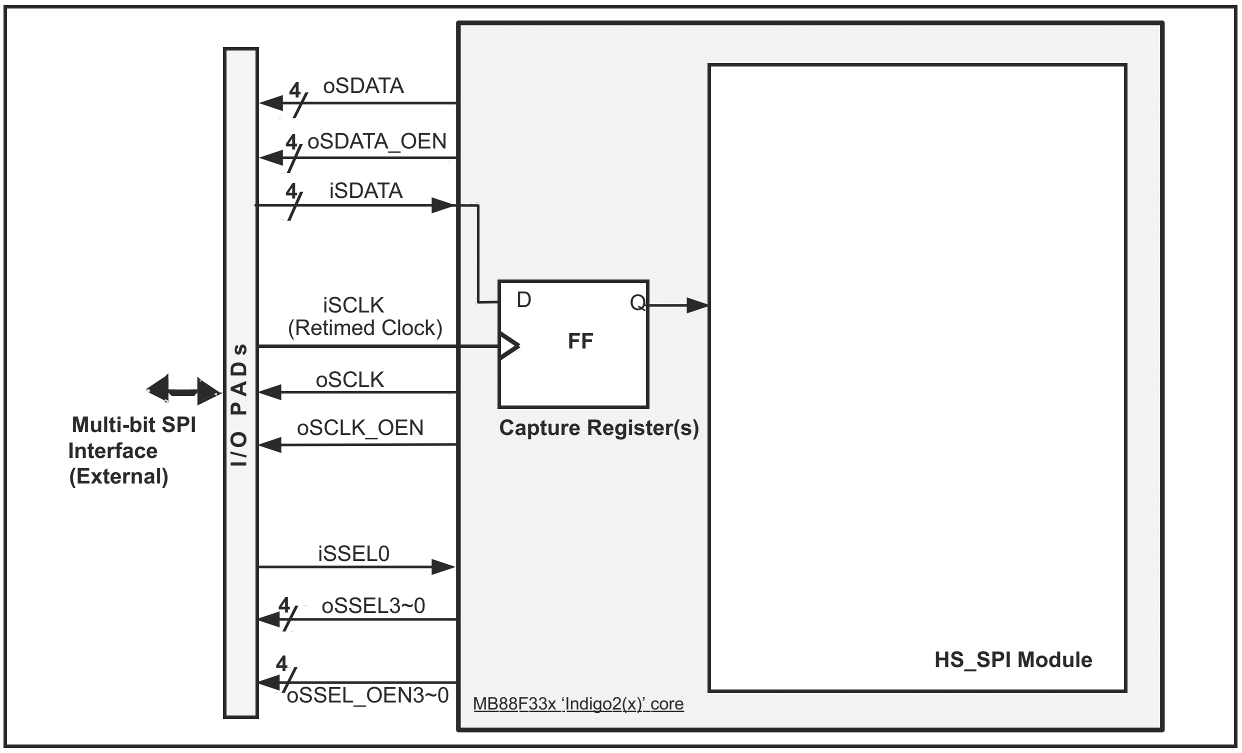


Figure 6-64: Re-timed Clock in HS_SPI

Figure 6-64 shows how the re-timed clock is generated in the MB88F33x ‘Indigo2(-x)’. The registers that are used for capturing the serial data input are placed physically close to the I/O pads, at the MB88F33x ‘Indigo2(-x)’ boundary. In re-timed clock mode (i.e. when HSSPIn_PCC0~3:RTM=1), these registers are sourced from the clock input, which is looped-back to the HS_SPI (from oSCLK to the I/O pad at the peripheral of the MB88F33x ‘Indigo2(-x)’, and back to the HS_SPI) i.e. iSCLK input is used for capturing the input data. This is the re-timed clock. This re-timing technique is a cycle stealing technique, which allows late arriving serial data signals (i.e. iSDATA[3:0]) to be sampled at a later point in time, by intentionally introducing a skew on the clock.

NOTE The alternative SPI interface can not be used together with the retimed mode of the external SPI interface.

6.7.3.3 Serial Clock Frequency

The SCLK clock is internally generated by dividing either the AHB clock (HCLK) or the peripheral clock (peri_clk). The HSSPIn_MCTRL:CDSS register bit controls the selection of the source clock for each of the four Slave Selects. The clock division ratio of the resulting internal clock-divider can also be programmed in the HSSPIn_PCC0~3 registers.

Figure 6-65 shows how the serial clock is generated. The HSSPIn_PCC0~3:CDRS field decides the clock division ratio. The frequency “F_o” of the clock generated by the clock divider is given by the equation:

$F_o = F_i / (2 \times \text{HSSPIn\_PCC0\~3:CDRS})$

Where F_i is the frequency of the source clock selected by the HSSPIn_MCTRL:CDSS field.

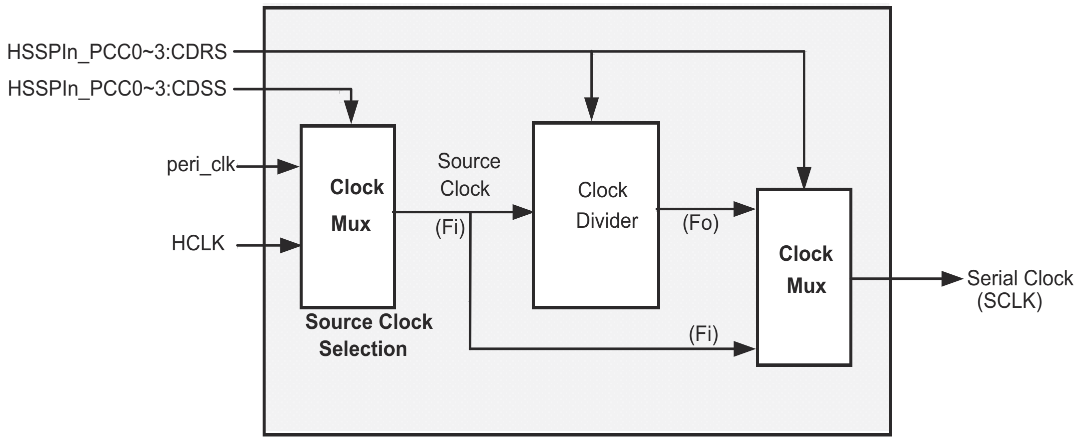


Figure 6-65: Serial Clock Generation

6.7.3.4 SPI Protocol

HS_SPI supports both: Legacy SPI, as well as the new dual-bit or quad-bit SPI protocol.

While in ‘Direct Mode’ of operation, the HSSPIn_DMTRP:TRP[1:0] bits decide whether HS_SPI uses legacy, the dual-bit, or the quad-bit protocol.

While in ‘Command Sequencer Mode’, the HSSPIn_CSCFG:MBM bits decide whether the HS_SPI uses legacy, the dual-bit, or the quad-bit protocol. The dual-bit and quad-bit SPI protocol is used for interfacing with the newer generation serial-flash memory devices.

6.7.3.5 Legacy SPI Protocol

The legacy SPI protocol is a full-duplex protocol. When the HS_SPI is configured with legacy protocol, the data can be received on a single wire (i.e. SDATA[1]) and simultaneously, the data can also be transmitted on a single wire (i.e. SDATA[0]). While legacy SPI protocol is being used, the unused data lines (i.e. SDATA[2] and SDATA[3]) are tri-stated by HS_SPI.

In ‘Direct Mode’, when HSSPIn_DMTRP:TRP is configured for “TX-and-RX in Legacy Mode”, ‘TX-Only in Legacy Mode’, or ‘RX-Only in Legacy Mode’, the full-duplex legacy SPI protocol is used by HS_SPI.

6.7.3.6 Dual Bit Protocol

In a dual-bit SPI protocol, two serial data lines (i.e. SDATA[1:0]) are used, in a half-duplex manner. Data transmission and reception cannot happen simultaneously. While dual-bit SPI protocol is being used, the unused data lines (i.e. SDATA[2] and SDATA[3]) are tri-stated by HS_SPI.

In ‘Direct Mode’, when HSSPIn_DMTRP:TRP is configured for ‘TX-Only in Dual Mode’, or ‘RX-Only in Dual Mode’, the Dual-bit SPI protocol is used.

6.7.3.7 Quad Bit Protocol

In quad-bit SPI protocol, all four serial data lines (i.e. SDATA[3:0]) are used, in a half-duplex manner. data transmission and reception cannot happen simultaneously.

In ‘Direct Mode’, when HSSPIn_DMTRP:TRP is configured for ‘TX-Only in Quad Mode’ or ‘RX-Only in Quad Mode’, the Quad-bit SPI protocol is used.

6.7.3.8 Shift Direction

The HS_SPI Peripheral Communication Configuration (HSSPIn_PCC0~3) registers have a bit (i.e. SDIR), which decides the direction in which the Shift Register is shifted.

When HSSPIn_PCC0~3:SDIR is 0, Most Significant Bit in the Shift Register is transmitted first and the first received data is shifted into the Least Significant Bit in the Shift Register. i.e. the Shift Register is shifted left.

When HSSPIn_PCC0~3:SDIR is 1, Least Significant Bit in the Shift Register is transmitted first and the first received data is shifted into the Most Significant Bit in the Shift Register. i.e. the Shift Register is shifted right.

Irrespective of the value of the HSSPIn_PCC0~3:SDIR bit, the read/write accesses to the data-registers always have least significant bit of the data in bit 0.

Figure 6-66 and Figure 6-67 on page 109 depict the direction in which the data in the shift register is shifted to/from the serial data lines, when either Legacy, Dual-bit, or Quad-bit SPI protocol is used.

The waveforms assume HSSPIn_PCC0~3:CPOL=0, HSSPIn_PCC0~3:CPHA=0 and HSSPIn_FIFOCFG:FWIDTH=0.

The figures depict that the transmit data is loaded into the shift register from the TX-FIFO. However, it shall be noted that the source of transmit data could also be the other data registers, like the HSSPIn_RDCSDC0~7:RDCSDATA or the HSSPIn_WRCSDC0~7:WRCSDATA.

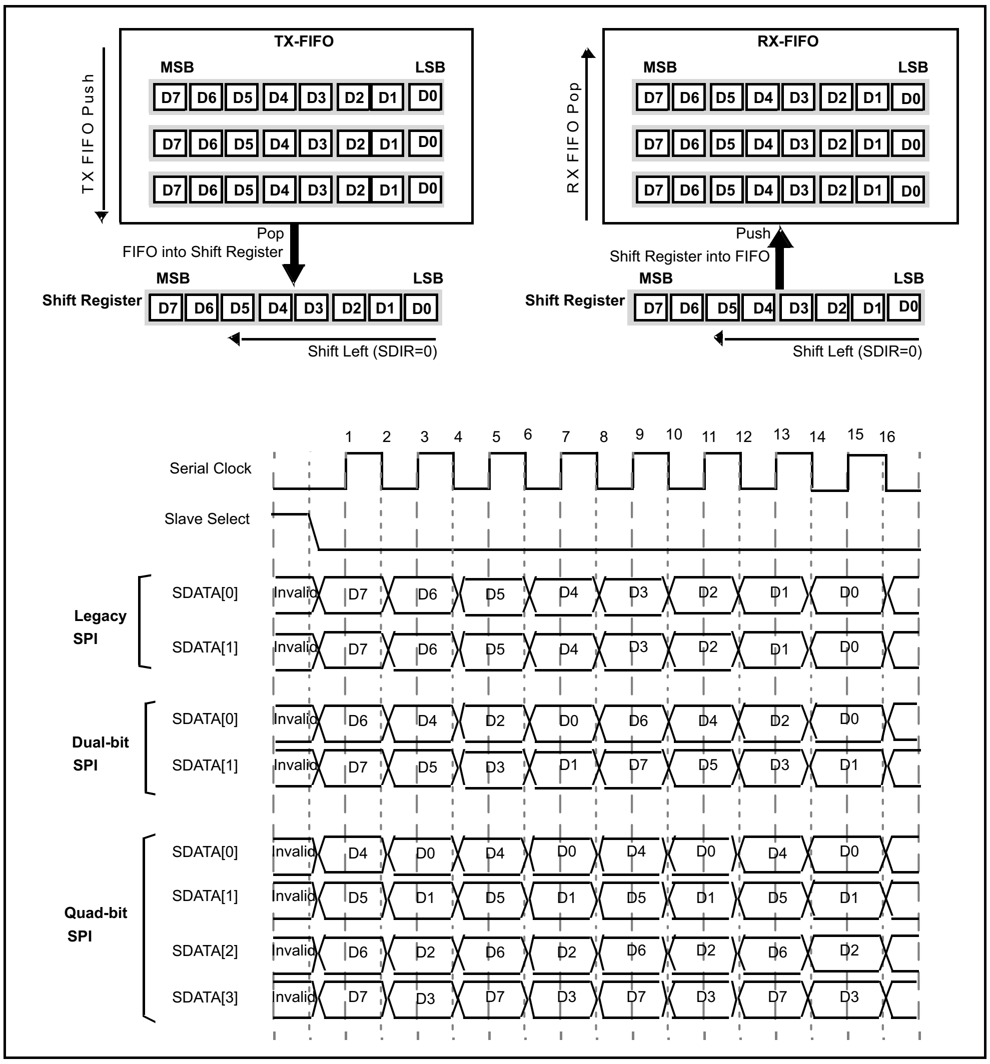


Figure 6-66: Shift Direction (Assumptions: CPOL=0, CPHA=0, SDIR=0, FWIDTH=0

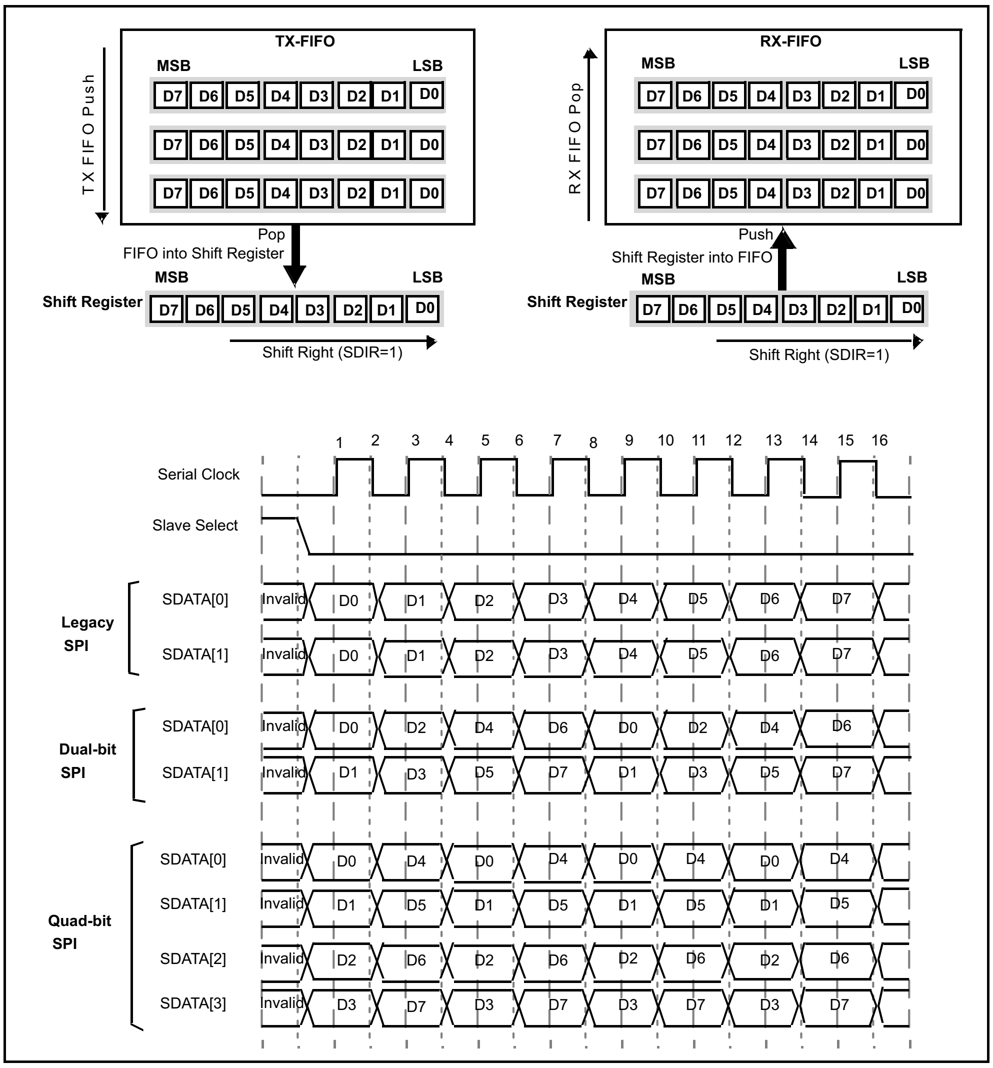


Figure 6-67: Shift Direction (Assumptions: CPOL=0, CPHA=0, SDIR=1, FWIDTH=0)

6.7.3.9 Safe Synchronisation of Internal Data

While a serial transfer is in progress, HS_SPI has to internally move the data across the two clock domains (AHB Clock domain and the Serial Clock domain), using synchronizers which have their inherent latency.

6.7.3.9.1 Synchronization

In certain cases, if the synchronizer latency becomes a bottleneck in the serial transfer, the data will not be synchronised properly. To avoid this situation, the software programmers must ensure that the HSSPIn_PCC0~3:SAFESYNC is set in such cases.

The exact conditions when the setting of the SAFESYNC bit is required, depend on the transfer protocol, the width of the shift register and the ratio between the AHB Clock Frequency (i.e. Fhclk) and the Serial Clock frequency (Fsclk).

When HSSPIn_PCC0~3:SAFESYNC bit is set, HS_SPI master halts the current serial transfer intermittently while it is internally synchronizing the data. The duration of this halt is 3 periods of the serial clock. The serial interface is halted for the safe synchronisation of internal data only when the following conditions are satisfied:

- HSSPIn_PCC0~3:SAFESYNC bit is set to 1 and
- “Width of shift register is 8-bits and serial interface is configured for Dual bit or Quad bit mode” OR “Width of shift register is 16-bits and serial interface is configured for Quad bit mode”.

Thus, when the HSSPIn_PCC0~3:SAFESYNC bit is set to 1, after the transfer of each data-chunk, the SPI-Core in HS_SPI decides on-the-fly whether to wait for safe-synchronisation or not, depending on the width of the shift register that is being used at that particular instant. Merely setting the HSSPIn_PCC0~3:SAFESYNC bit does not imply that the SPI-Core would insert extra wait states (for safe synchronisation) for every chunk of the data being transmitted. It inserts the extra wait states if-and-only-if the specific conditions related to the width of shift register and the SPI protocol (i.e. Legacy/ Dual/Quad) in use are satisfied. This ensures that the achievable bandwidth of the serial transfer is not severely hampered due to the time lost in safe-synchronisation.

The term "Width of Shift Register", used above, with reference to SAFESYNC bit is explained in the following sub-sections.

In ‘Direct Mode’, generally the width of the shift register is decided by the width of the FIFOs, configured in HSSPIn_FIFOCFG:FWIDTH. However, there are two special conditions that also need to be considered:

- If the TXCTRL bit in any of the TX-FIFO locations is set to 1, then the smallest width of the shift register used during the entire transfer is 1 byte. This smallest value of the shift register width shall be used by HS_SPI while deciding whether safe synchronisation is required for a transfer or not.
- Specifically while using the ‘Counter Mode’ (i.e. HSSPIn_DMBCC and HSSPIn_DMBCS registers) for stopping the serial transfer, if the number of bytes to be transferred (programmed in HSSPIn_DMBCC register) is not divisible by the number of bytes configured in the FIFO width (i.e. HSSPIn_FIFOCFG:FWIDTH field), then the remainder of the division decides the smallest width of the shift-register that is used during the transfer. As an example, if HSSPIn_DMBCC:BCC is programmed with a value of 10 bytes and the HSSPIn_FIFOCFG:FWIDTH is programmed with a value of 4 bytes, then the HS_SPI will perform the loading/unloading of the shift register in sets of 4 bytes, followed again by 4 bytes and the remaining chunk of 2 bytes is transferred before the transfer ends. Thus, in this case the smallest width of the shift register used during the entire transfer is of 2 bytes (assuming that the TXCTRL bit in all TX-FIFO locations is 0). This smallest value of the shift register width shall be used by HS_SPI while deciding whether safe synchronisation is required for a transfer or not.

Table 6-21 tabulates the conditions when the HSSPIn_PCC0~3:SAFESYNC bit shall be set by the CPU to “1” while HS_SPI is configured in ‘Direct Mode’ operation.

Table 6-21: Criteria for setting the SAFE SYNC bit in ‘Direct Mode’ Operation

Mode	Width of Shift Register	Protocol	SAFESYNC shall be set to “1”, if:	Maximum Supported SCLK Frequency
Direct Mode	8 bits	Legacy	SAFESYNC is not required	Fsclk = Fhclk
		Dual Bit	Fsclk > (1/2) Fhclk	
		Quad Bit	Fsclk > (1/5) Fhclk	
	16 bits	Legacy	SAFESYNC is not required	
		Dual Bit	SAFESYNC is not required	
		Quad Bit	Fsclk > (1/2) Fhclk	
	24 bits	Legacy	SAFESYNC is not required	
		Dual Bit		
		Quad Bit		
	32 bits	Legacy	SAFESYNC is not required	
		Dual Bit		
		Quad Bit		

In ‘Command Sequencer Mode’, while the Command Sequence is being transmitted, the width of the shift register is 8-bits; and while the data is being written/read (to/from the Serial Memory), the width of the shift register is equal to the AHB bus transfer size.

Table 6-22 precisely tabulates the conditions where setting the HSSPIn_PCC0~3:SAFESYNC bit to “1” is required while operating in ‘Command Sequencer Mode’.

Table 6-22: Criteria for setting the SAFESYNC bit in ‘Command Sequencer Mode’

Mode Of Operation	AHB Transfer Size of the Memory-mapped Transfer	Protocol	SAFESYNC shall be set to “1”, if:	Maximum Supported SCLK Frequency
Command Sequencer	8 bits	Legacy	SAFESYNC is not required	Fsclk = Fhclk
		Dual Bit	Fsclk > (1/2) Fhclk	
		Quad Bit	Fsclk > (1/6) Fhclk	Fsclk = (3/4) Fhclk
	16 bits	Legacy	SAFESYNC is not required	Fsclk = Fhclk
		Dual Bit	Fsclk > (1/2) Fhclk	
		Quad Bit	Fsclk > (1/5) Fhclk	
	32 bits	Legacy	SAFESYNC is not required	
		Dual Bit	Fsclk > (1/2) Fhclk	
		Quad Bit	Fsclk > (1/5) Fhclk	

6.7.4 Direct Mode

In 'Direct Mode', the software is responsible for the direct control of the serial transfer via the Serial Interface. 'Direct Mode' of transfer can be enabled using the HSSPIn_MCTRL:CSEN bit. In 'Direct Mode' of operation, HS_SPI uses its internal FIFOs, for temporary storage of the data to be transmitted and the data received over the serial interface.

This section describes the 'Direct Mode' of operation.

6.7.4.1 Internal FIFOs

HS_SPI internally has two FIFOs for temporary storage: One for the data to be transmitted and one for the data to be received.

Based on whether the serial transfers in HS_SPI are configured as TX-Only, RX-Only, or TX-and-RX in the HSSPIn_DMTRP:TRP field, only one or both FIFOs are used by HS_SPI. If HS_SPI is configured for TX-Only operation, the TX-FIFO is used. If HS_SPI is configured for RX-Only operation, the RX-FIFO is used. If HS_SPI is configured for TX-and-RX operation, then both FIFOs are used.

6.7.4.2 FIFO Size

Each FIFO is 16-locations deep and has a data-width of 32 bits. However, the software can configure the valid data-width of the TX-FIFO and the RX-FIFO in HSSPIn_FIFOCFG:FWIDTH.

The shift register in the SPI core is 32-bit wide. When the width of the FIFO is changed in the HSSPIn_FIFOCFG:FWIDTH field, the usable width of the Shift Register also changes accordingly.

Please refer to Figure 6-68 for more details.

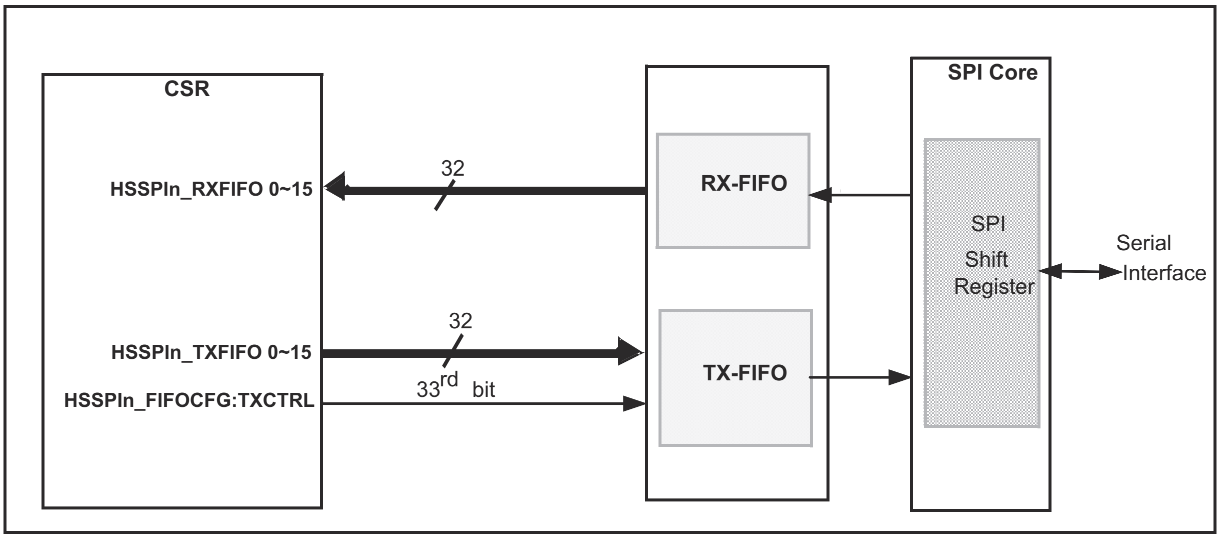


Figure 6-68: HS_SPI in 'Direct Mode'

In addition to the 32-bit of data-width, each location in TX-FIFO has a 33rd control bit (known as TXCTRL bit), which decides whether the data from the TX-FIFO is to be transmitted by the SPI core, or whether the serial data lines are to be tri-stated. If the TXCTRL bit is set to "1", HS_SPI further decodes the bit 0 of the data in the corresponding TX-FIFO location. All possibilities of the combinations of values of the TXCTRL bit and the bit 0 of TX-FIFO data are tabulated in Table 6-23 .

Table 6-23: Tri-stating the serial data output lines during transmission

TXCTRL	Bit 0 of TX-FIFO Data	Description
0	Don’t Care	Serial data output lines are not tri-stated while transmitting the corresponding data.
1	0	Serial data output lines are tri-stated for 1 byte-time. The data from the corre-sponding TX-FIFO location is not transmitted.
1	1	Irrespective of the shift direction configured in HSSPIn_PCC0~3:SDIR bits, the data transmission takes place in the following order: Data bits [7:4] from the corresponding TX-FIFO location are transmitted. Direction of transmission of this data depends on configuration of HSSPIn_PCC0~3:SDIR bit. SDATA output lines are tri-stated for 4 Bit-Times.

For all practical reasons (e.g. while using the HSSPIn_DMBCC:BCC feature or while referring to Table 6-21: "Criteria for setting the SAFE SYNC bit in ‘Direct Mode’ Operation"), the data in the TX-FIFO location for which the TXCTRL bit is set, is considered to be one byte wide. The TXCTRL bit associated with the data words in the TX-FIFO allows the software to generate the command sequences (using ‘Direct Mode’), for interfacing with some quad-bit SPI Memories available in the market, which need to tri-state the Serial Data (i.e. SDATA) lines during the “dummy” cycles or during the transmission of lower nibble of the “mode bits” of the command sequence.

6.7.4.3 FIFO Accesses

Irrespective of the configured width of the FIFOs, only 32-bit word accesses are allowed to the HSSPIn_RXFIFO0~15 and HSSPIn_TXFIFO0~15 registers.

A read access to any of the HSSPIn_RXFIFO0~15 registers in CSR directly pops out a word from the RX-FIFO. If the RX-FIFO width was set to 8-bit, then the most-significant 24-bits read out from HSSPIn_RXFIFO0~15 register are logic-0. Similarly, when FIFO-width is set to 16-bit or 24-bits, the unused bits read-out from HSSPIn_RXFIFO0~15 register are logic-0.

A write access to any of the HSSPIn_TXFIFO0~15 registers in CSR pushes a word of data and a TXCTRL bit (see HSSPIn_FIFOCFG:TXCTRL) into the TX-FIFO. However, when HS_SPI transmits the data on the serial lines, it uses only the least-significant-bits of the data read from the HSSPIn_TXFIFO0~15 registers. The number of these least-significant-bits that are transmitted depend on the configured width of the TX-FIFO. The unused most-significant bits are ignored by HS_SPI.

6.7.4.4 Accessing the RX-Data

The serial data received by HS_SPI on the SDATA lines is assembled in a Shift Register (in the SPI Core), before it is pushed into the RX-FIFO.

When a transfer completes (i.e. Slave-Select line is deasserted) or when a transfer halts, the HSSPIn_RXSHIFT register is updated with the assembled data and the HSSPIn_RXBITCNT:RXBITCNT field is updated with the number of bits valid in HSSPIn_RXSHIFT register. When the HSSPIn_TXF:TSSRS or the HSSPIn_RXF:RSSRS interrupt flag is set, the software can read the HSSPIn_RXSHIFT and HSSPIn_RXBITCNT:RXBITCNT field, to get the RX data which is not yet pushed into the RX-FIFO.

6.7.4.5 Service Requests

When operating in ‘Direct-Mode’, Interrupt Service Requests to the host software are triggered based on the current fill-levels of the TX-FIFO and the RX-FIFO; and their configured threshold values. Alternatively, the external DMA engine can be used for data transfers. HS_SPI has an interface with the DMA engine in the MB88F33x ‘Indigo2(-x)’, for block-transfers of data to/from its TX-FIFO and the RX-FIFO, when operating in ‘Direct Mode’. Interrupt flags are also set when the current SPI transfer finishes.

6.7.4.6 Assertion of Interrupt Service Requests Based on FIFO Levels

The fill-levels of both FIFOs are accessible to the system through the HSSPIn_DMSTATUS:TXFLEVEL and the HSSPIn_DMSTATUS:RXFLEVEL fields. The Interrupt service requests are generated by HS_SPI based on the FIFO fill levels and their configured threshold values.

The “TX-FIFO Fill-level Less Than or Equal to Threshold” (i.e. HSSPIn_TXF:TFLETS) interrupt flag is set, if the TX-FIFO fill level (i.e HSSPIn_DMSTATUS:TXFLEVEL) is less than or equal to the TX-FIFO threshold value configured in HSSPIn_FIFOCFG:TXFTH.

The “RX-FIFO Fill-level More than Threshold” (i.e. HSSPIn_RXF:RFMTS) interrupt flag is set, if the RX-FIFO fill level (i.e. HSSPIn_DMSTATUS:RXFLEVEL) is more than the RX-FIFO threshold value configured in HSSPIn_FIFOCFG:RXFTH.

If HS_SPI is configured for TX-Only operation, the RX-FIFO is not used. If HS_SPI is configured for RX-Only operation, the TX-FIFO is not used.

6.7.4.7 Assertion of DMA Service Requests Based on FIFO Levels

HS_SPI supports block transfer mechanism of DMA Engine. The DMA service requests are generated by HS_SPI, based on the FIFO fill levels and their configured threshold values. To keep track of the number of successful data-transfers to/from the TX FIFO and/or the RX FIFO, HS_SPI internally maintains two down-counters: HS_SPI RX Block Counter and HS_SPI TX Block Counter. Each of these counters is a 5-bit down counter, which is reloaded with the DMA Block size (for the respective channel) whenever the DMA service request for that channel is asserted. The counters are decremented with every successful read (or write) accesses to the RX FIFO (or TX FIFO). In case of the RX FIFO accesses, the RX Block Counter is decremented only if the access was from an AHB master other than the DAP Controller. The block counters do not underflow (i.e. the counter value remains 0 even if it is decremented while it is already 0).

Each DMA Channel (Read Channel and Write Channel) has a dedicated DMA Block Size Fault status flag in the HSSPIn_FAULTF register. A DMA Block Size Fault is triggered, if all of the following conditions are satisfied:

1. The DMA Block Counter is decremented (due to a valid AHB access) while it is already 0, AND
2. The DMA enable bit (HSSPIn_DMDMAEN:RXDMAEN or HSSPIn_DMDMAEN:TXDMAEN) for the corresponding DMA Channel is set to 1, AND
3. Module is enabled (i.e. HSSPIn_MCTRL:MES=1), AND
4. Module is operating in ‘Direct Mode’ of operation (i.e. HSSPIn_MCTRL:CSEN=0).

The DMA Read Channel must be setup to perform a block transfer of "HSSPIn_FIFOCFG:RXFTH + 1" transfers. The DMA Write Channel must be setup to perform a block transfer of "16 - HSSPIn_FIFOCFG:TXFTH" transfers. These values are reloaded into the HS_SPI module’s internal Block Counters, whenever the DMA read/write channel service request is asserted.

The DMA Block Counter is reset to "0" in either of the following conditions:

1. The corresponding DMA channel is disabled (in HSSPIn_DMDMAEN register), OR
2. Module is completely disabled (i.e. HSSPIn_MCTRL:MES=0), OR
3. Mode of operation is switched from ‘Direct Mode’ to ‘Command Sequencer Mode’.

The RX DMA Service Request (for DMA Read Channel) is asserted if all of the following conditions are satisfied:

1. RX FIFO fill level (i.e. HSSPIn_DMSTATUS:RXFLEVEL) is more than the RX-FIFO threshold value configured in HSSPIn_FIFOCFG:RXFTH. This condition is same as the HSSPIn_RXF:RFMTS bit, AND
2. HS_SPI RX Block Counter value is 0, AND
3. DMA Read Channel acknowledgement signal is deasserted by the DMA Engine, AND
4. Previous service request for DMA Read Channel is not pending, AND
5. DMA Read Channel Service request is enabled in the HSSPIn_DMDMAEN:RXDMAEN bit, AND
6. DMA Read Channel Block Size Fault interrupt flag is not set (i.e. HSSPIn_FAULTF:DRCBFES=0), AND
7. Module is enabled (i.e. HSSPIn_MCTRL:MES=1), AND
8. Module is in ‘Direct Mode’ (i.e. HSSPIn_MCTRL:CSEN=0), AND
9. The configured transfer protocol (in HSSPIn_DMTRP) is such that RX FIFO is used. e.g. If HS_SPI is configured for TX-Only operation, the RX-FIFO is not used and DMA Read Service Request is not asserted in such cases.

The RX DMA Service Request (for DMA Read Channel) is deasserted if any of the following condition is satisfied:

1. DMA Read Channel Service Request has been acknowledged by the DMA engine, OR
2. DMA Read Channel Service Requests have been disabled (i.e. HSSPIn_DMDMAEN:RXDMAEN=0), OR
3. Module is completely disabled (i.e. HSSPIn_MCTRL:MES=0), OR
4. Mode of operation is switched from ‘Direct Mode’ to ‘Command Sequencer Mode’.

The TX DMA Service Request (for DMA Write Channel) is asserted if all of the following conditions are satisfied:

1. TX FIFO fill level (i.e. HSSPIn_DMSTATUS:TXFLEVEL) is less than or equal to the threshold value configured in HSSPIn_FIFOCFG:TXFTH. This condition is same as the HSSPIn_TXF:TFLETS interrupt flag, AND
2. HS_SPI TX Block Counter value is 0, AND
3. DMA Write Channel acknowledgement signal is deasserted by the DMA Engine, AND
4. Previous service request for DMA Write Channel is not pending, AND
5. DMA Write Channel Service Request is enabled in the HSSPIn_DMDMAEN:TXDMAEN bit, AND

6. DMA Write Channel Block Size Fault interrupt flag is not set (i.e. HSSPIn_FAULTF:DWCBSFS=0), AND
7. Module is enabled (i.e. HSSPIn_MCTRL:MES=1), AND
8. Module is in ‘Direct Mode’ (i.e. HSSPIn_MCTRL:CSEN=0), AND
9. The configured transfer protocol (in HSSPIn_DMTRP) is such that TX FIFO is used. e.g. If HS_SPI is configured for RX-Only operation, the TX-FIFO is not used and DMA Write Service Request is not asserted in such cases.

The TX DMA Service Request (for DMA Write Channel) is deasserted if any of the following condition is satisfied:

1. DMA Write Channel Service Request has been acknowledged by the DMA engine, OR
2. DMA Write Channel Service Requests have been disabled (i.e. HSSPIn_DMDMAEN:TXDMAEN=0), OR
3. Module is completely disabled (i.e. HSSPIn_MCTRL:MES=0), OR
4. Mode of operation is switched from ‘Direct Mode’ to ‘Command Sequencer Mode’.

6.7.4.8 Assertion of Service Requests on End of Transfer

While operating in ‘Direct Mode’, the HS_SPI also triggers interrupts, when the Slave Select line is deasserted. The Slave Select Deassertion event is routed onto two Interrupt Flags: HSSPIn_TXF:TSSRS and HSSPIn_RXF:RSSRS, which have separate Interrupt-Clear and Interrupt-Enable bits. The interrupt flags are routed onto separate Interrupt Signals. This is indicated in .

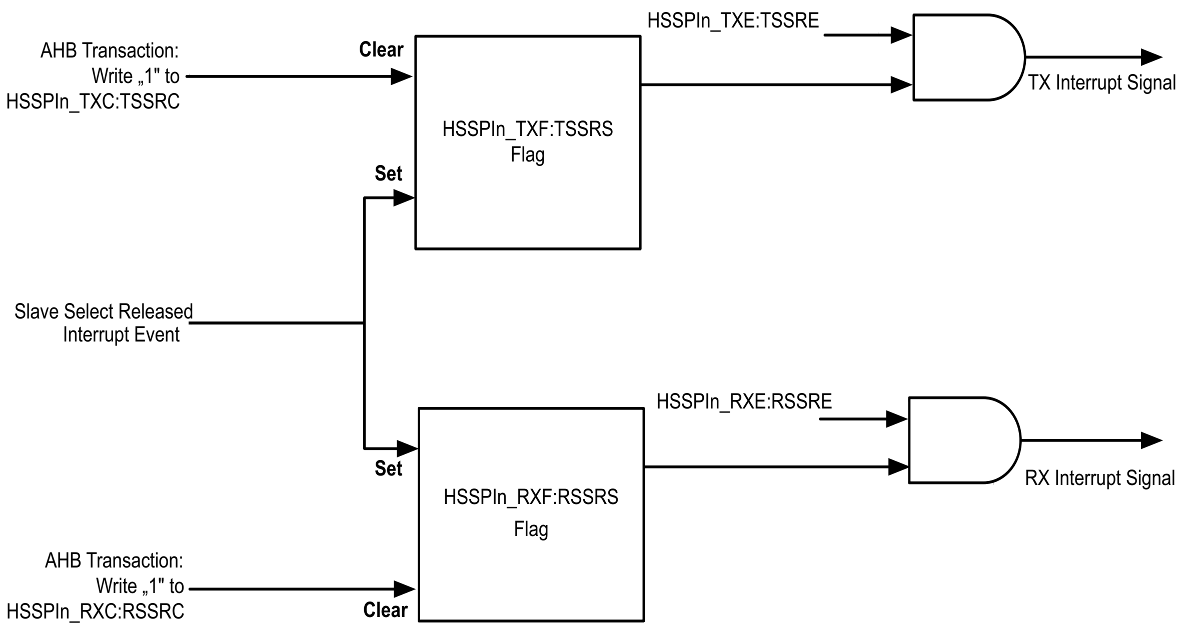


Figure 6-69: Routing of the “Slave Select Released” Interrupt Event

6.7.4.9 SPI Transfers

The HS_SPI initiates the transfers onto one of the four SPI slave-select lines, selected by the HSSPIn_DMPSEL:PSEL field.

6.7.4.10 Communication Attributes of HS_SPI

Communication over the serial interface has several attributes, like: Frequency, Polarity and Phase of the Serial Interface Clock, Polarity of the Slave Select line, etc. These communication attributes are different for different SPI devices. When HS_SPI is working in ‘Direct Mode’ of operation, it can be interfaced with upto 4 slaves, each with a different values of these attributes.

These device-specific communication attributes can be configured in the HSSPIn_PCC0~3 registers in CSR.

6.7.4.11 Initiating the Serial Transfers

When HS_SPI is enabled (i.e. HSSPIn_MCTRL:MEN=1) in ‘Direct Mode’ (i.e. HSSPIn_MCTRL:CSEN=0), serial transfers are initiated by HS_SPI when the HSSPIn_DMSTART:START bit is set to “1”.

If HSSPIn_DMTRP:TRP is programmed such that transmission is enabled, and if the TX-FIFO is empty when the HSSPIn_DMSTART:START bit is set to "1"; then HS_SPI delays the initiation of the serial transfer until the TX-FIFO is written by the software.

While HS_SPI has delayed the initiation of the serial transfer (until TX-FIFO is written by software):

1. If HSSPIn_MCTRL:MEN bit is reset to "0" by software, then the disabling of the module will take precedence over starting the next transfer.
2. If ‘Counter Mode’ is used for controlling the transfer length, then the HSSPIn_DMBCS register will be loaded with the value in HSSPIn_DMBCC register only when the serial transfer is initiated by HS_SPI. Until then, the HSSPIn_DMBCS register will maintain it’s 0 value.

Please note, that once the HSSPIn_DMSTART:START bit is set to “1”, it cannot be reset by the software. The HS_SPI module resets the bit after it has started the Serial Transfer. Writing a “1” to the START bit while it is already set to “1” has no effect on the bit. Writing a “1” to the START bit while it is “0” and a serial transaction is already in progress does not affect the ongoing transfer. A new serial transfer is initiated after the current transfer completes.

6.7.4.12 Halting a Transfer Due To Lack Of TX-DATA or Due To Lack of RX-FIFO Space

As-per the standard SPI protocol, an ongoing transfer can be halted by keeping the Slave Select asserted and by cutting the Serial Clock. HS_SPI automatically cuts the serial clock while it is waiting for the TX-FIFO to be written or while it is waiting for the RX-FIFO to be read. Depending on whether the HS_SPI Master is operating as TX-Only, RX-Only or TX-and-RX, there are three scenarios in which HS_SPI cuts the serial clock and halts a transfer:

- **TX-Only Mode**: The serial clock is cut when the TX-FIFO is empty and the Shift Register is empty.
- **RX-Only Mode**: The serial clock is cut when the RX-FIFO is full and when the Shift Register is full.
- **TX-and-RX Mode**: The serial clock is cut when: (a) the TX-FIFO and the Shift Register is empty OR (b) RX-FIFO and the Shift Register is full.

When an ongoing transfer is halted (by cutting the serial clock) by HS_SPI due to unavailability of FIFO resources, the corresponding slave-select line is kept asserted, indicating to the slave, that the transfer has not finished. The halted transfers are automatically resumed by HS_SPI (by starting the toggling of the serial clock) when the FIFO resources become available.

6.7.4.13 Controlling the Transfer Length

The transfer length (i.e the de-assertion of the slave-select lines) can be controlled in two ways:

- ‘Counter mode’ and
- ‘Software Flow Control’ mode

These modes can be selected in HSSPIn_DMCFG:SSDC bit.

In ‘Counter Mode’, the CPU is supposed to initialize the HSSPIn_DMBCC:BCC field with the number of bytes to be transferred over the serial interface, before the Slave-Select (i.e. SSEL) output is deasserted. When the HS_SPI transfers are initiated, the HS_SPI counts the number of bytes that are transferred and releases the slave-select signal after the number of bytes indicated in HSSPIn_DMBCC:BCC have been transferred.

In ‘Software Flow Control Mode’, the CPU controls the transfer length by using the HSSPIn_DMSTOP:STOP bit. Depending on whether the HS_SPI is operating as TX-Only, RX-Only or TX-and-RX, the deassertion of Slave-Select output is controlled in the following ways:

- **TX-Only Mode**: The transfer is completed when the HSSPIn_DMSTOP:STOP bit is set and all contents of TX-FIFO are transmitted.
- **RX-Only Mode**: The transfer is completed, when the HSSPIn_DMSTOP:STOP bit is set and all bits in the Shift Register (used in SPI-Core for assembling the received serial data) are shifted in.
- **TX-and-RX Mode**:The transfer is completed when the HSSPIn_DMSTOP:STOP bit is set and all contents of TX-FIFO are transmitted.

6.7.4.14 Lack Of FIFO Resources while a Serial Transfer is Ongoing

While HS_SPI is configured for transmission (i.e. TX-Only or TX-and-RX configuration in HSSPIn_DMTRP:TRP field), if the TX-FIFO becomes empty, HS_SPI sets the HSSPIn_TXF:TFES interrupt flag. Even after the TX-FIFO becomes empty, it keeps reading the data from the TX-FIFO (causing a TX-FIFO Underrun) and transmitting the data it gets from the TX-FIFO on the serial data lines, as long as the slave select is asserted and the clock is toggling.

While HS_SPI is configured for reception (i.e. RX-Only or TX-and-RX configuration in HSSPIn_DMTRP:TRP field), if the RX-FIFO becomes full, HS_SPI sets the HSSPIn_RXF:RFFS interrupt flag and keeps overwriting the serial data received on the SDATA lines in to the RX-FIFO.

6.7.5 Command Sequencer Mode

In Command Sequencer mode, HS_SPI interfaces with the external serial memory devices. Each of the 4 slave select lines can be used for mapping uniform type of "Serial Flash" or "Serial SRAM" devices. Memory accesses initiated by the CPU and the other AHB masters on the AHB bus are automatically converted to the serial memory read/write commands by HS_SPI.

This section describes the Command Sequencer mode of operation of HS_SPI.

6.7.5.1 Memory Mapping

The Command Sequencer mode can be used for memory mapping of up to 4 Serial Flash or Serial SRAM memory devices on the four Slave Select outputs of HS_SPI. All memory mapped devices shall be of the same family.

In Command Sequencer mode, HS_SPI is allocated a memory space of 256MBytes, for mapping up to 4 external memory devices. Each Slave Select can theoretically address a memory of up to 4GBytes (i.e. 32-bit address bus) by using the Address Extension mechanism in Command Sequencer. The Address Extension mechanism allows concatenation of the most significant bits from a 19-bit Address Extension Register (i.e. HSSPIn_CSAEXT register) with the few bits from the AHB address bus, to form a 32-bit address to be accessed on each slave-select. This feature is explained in detail in the subsequent sub-sections of this chapter.

Thus, the 256MByte of the MB88F33x 'Indigo2(-x)' address-space is virtually mapped to 16GBytes of external serial memory, as shown in Figure 6-70.

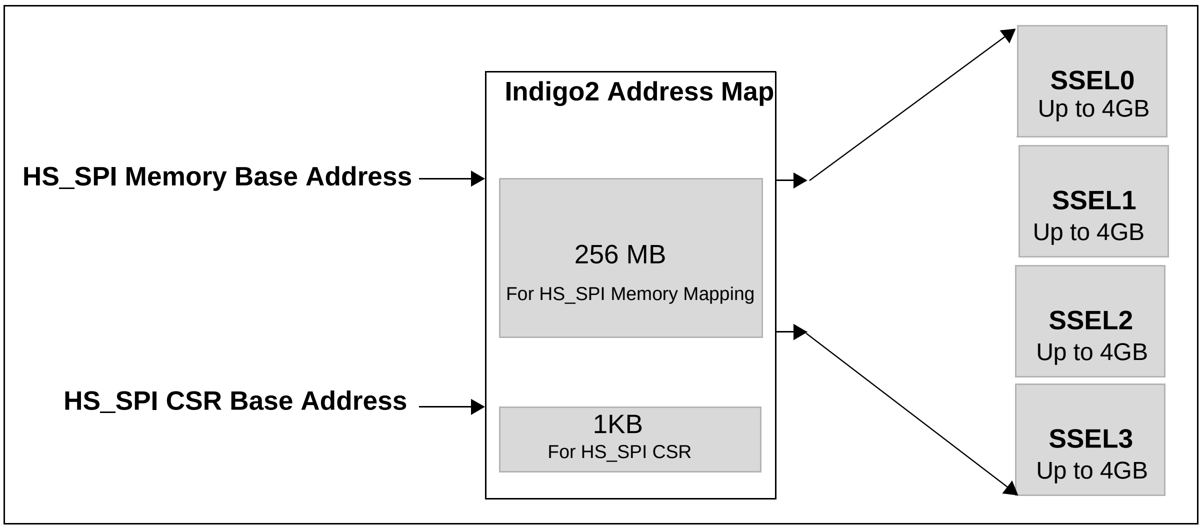


Figure 6-70: Mapping of Memory Devices on the Slave Select lines

6.7.5.2 Selection of Slaves

The HSSPIn_CSCFG:MSEL field indicates the size of the AHB address space associated with each Slave Select line. Based on the value of HSSPIn_CSCFG:MSEL field and the address placed by the CPU (or the other AHB Master, like the DMA Controller) on the AHB Address Bus, HS_SPI Command Sequencer decides which of the 4 Slave Select lines is asserted. Please refer to Table 6-25 for details.

As an example, let us assume that the HSSPIn_CSCFG:MSEL indicates that the AHB address space associated with each Slave Select is of 8KB. If the AHB address is between "HS_SPI Memory Base Address" and "HS_SPI Memory Base Address + 8KB", then Slave Select 0 is asserted. If the AHB address is between "HS_SPI Memory Base Address + 8KB" and "HS_SPI Memory Base Address + 16KB", then Slave Select 1 is asserted, and so on. If the AHB Address is beyond "HS_SPI Memory Base Address + 32KB", then the address is out of range and HSSPIn_FAULTF:UMAFS interrupt flag is set.

6.7.5.3 Generation of 32-bit Memory Address

The mapping of 256MB of the MB88F33x ‘Indigo2(-x)’ address space to 4GB of address space on a Slave Select line is possible due to Address Extension Mechanism. Every memory device can be visualized to be consisting of several memory banks. The size of each bank can be programmed in the HSSPIn_CSCFG:MSEL field. Each bank can be selected by changing the value in the HSSPIn_CSAEXT register. By reprogramming the HSSPIn_CSAEXT register each time a new bank in the selected Slave is to be accessed, a memory device of upto 4GB size can be addressed through different banks.

Please refer to Figure 6-71. It shows how each 4GB device consists of 524288 banks when the HSSPIn_CSCFG:MSEL field is programmed to “0000”.

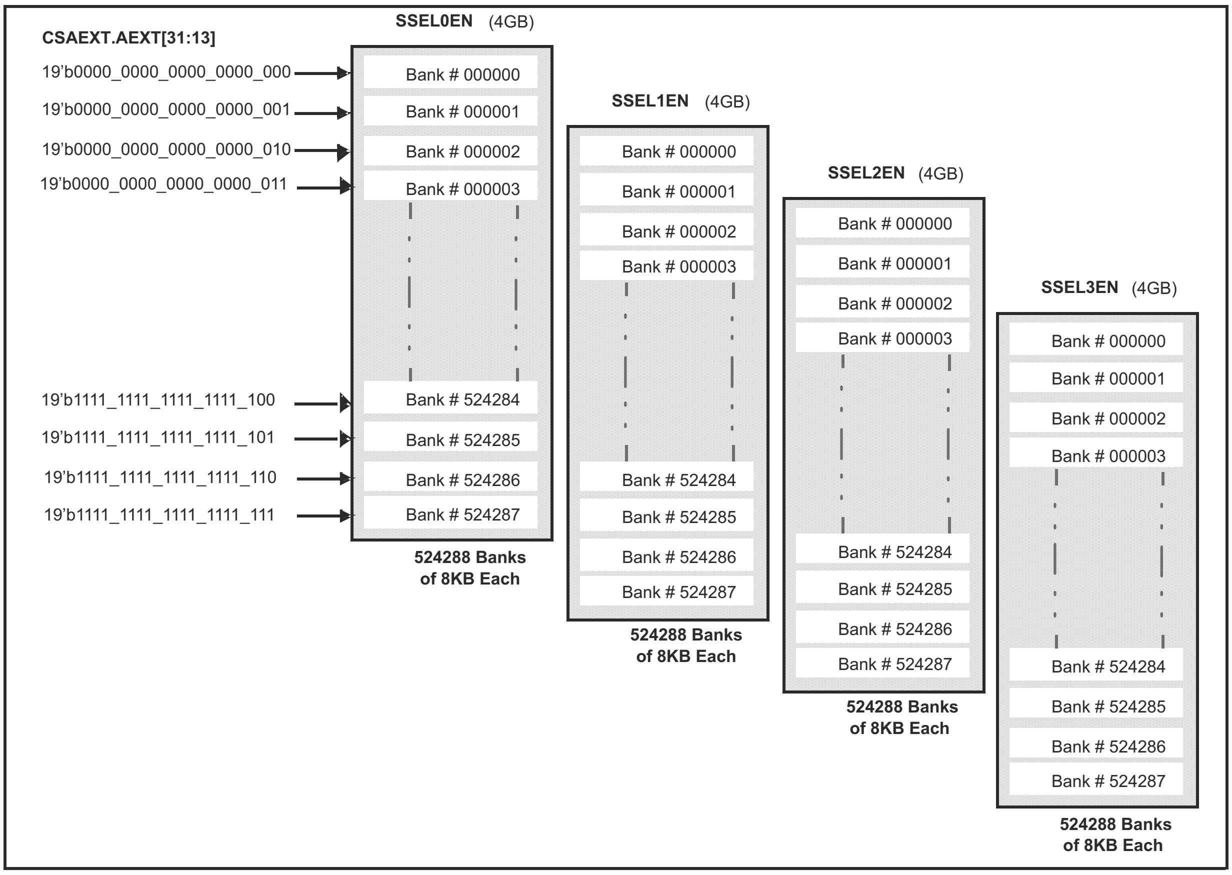

CSAEXT.AEXT[31:13]
19’b0000_0000_0000_0000_000
19’b0000_0000_0000_0000_001
19’b0000_0000_0000_0000_010
19’b0000_0000_0000_0000_011
19’b1111_1111_1111_1111_100
19’b1111_1111_1111_1111_101
19’b1111_1111_1111_1111_110
19’b1111_1111_1111_1111_111
SSEL0EN (4GB)
SSEL1EN (4GB)
SSEL2EN (4GB)
SSEL3EN (4GB)
Bank # 000000
Bank # 000001
Bank # 000002
Bank # 000003
Bank # 524284
Bank # 524285
Bank # 524286
Bank # 524287
524288 Banks of 8KB Each

Figure 6-71: FromSubjectReceivedSizeCategories (HSSPIn_CSCFG:MSEL=0000)

The least significant bits of the AHB Address received by HS_SPI on the AHB Bus are used as the offset within the bank selected by the Address Extension bits.

The concatenation of the appropriate number of bits from the Address Extension Register with the appropriate number of bits from the AHB address bus give the 32-bit address of the memory to be accessed on the serial interface. Please refer to Table 6-24 on page 121.

.

Table 6-24: MB88F33x ‘Indigo2(-x)’’ Address Space to Memory Address Mapping

HSSPIn_CSCFG: MSEL Field	Size of a Memory Bank on each Slave Select / Size of the AHB Address Range associated with each Slave Select	Number of Slave Select lines that can be activated	Number of Bits Used from HSSPIn_CSAEXT Register, for Selection of the Memory Bank on a Slave Select	Number of Bits Used from AHB Address Bus for addressing the memory location within a Bank
0000	8K Bytes	SSEL0, SSEL1, SSEL2 and SSEL3	AEXT[31:13]	HADDR[12:0]
0001	16K Bytes		AEXT[31:14]	HADDR[13:0]
0010	32K Bytes		AEXT[31:15]	HADDR[14:0]
0011	64K Bytes		AEXT[31:16]	HADDR[15:0]
0100	128K Bytes		AEXT[31:17]	HADDR[16:0]
0101	256K Bytes		AEXT[31:18]	HADDR[17:0]
0110	512K Bytes		AEXT[31:19]	HADDR[18:0]
0111	1M Bytes		AEXT[31:20]	HADDR[19:0]
1000	2M Bytes		AEXT[31:21]	HADDR[20:0]
1001	4M Bytes		AEXT[31:22]	HADDR[21:0]
1010	8M Bytes		AEXT[31:23]	HADDR[22:0]
1011	16M Bytes		AEXT[31:24]	HADDR[23:0]
1100	32M Bytes		AEXT[31:25]	HADDR[24:0]
1101	64M Bytes		AEXT[31:26]	HADDR[25:0]
1110	128M Bytes	SSEL0 and SSEL1 Only	AEXT[31:27]	HADDR[26:0]
1111	256M Bytes	SSEL0 Only	AEXT[31:28]	HADDR[27:0]

Last two columns in Table 6-24 indicate which bits from HSSPIn_CSAEXT:AEXT and the AHB address bus (i.e. HADDR) are concatenated, to get the final 32-bit address of the serial memory.

Although the final memory address generated in this way is a 32-bit address, it must be noted, that the software can choose the number of bytes (from this 32-bit address) to be sent to the serial memory device during the address phase of a memory read/write command sequence.

6.7.5.4 Initiation of Command Sequence

Whenever the Command Sequencer receives a AHB read access for the memory mapped serial device, it initiates a corresponding memory read command on one of the four Slave Select lines, assembles the data it has received, and responds with the memory read-data.

Similarly, if it receives an AHB write access for the memory mapped serial device, it initiates a corresponding memory write command on one of the four Slave Select lines and transmits the data to be written, serially on the SDATA lines.

While the HS_SPI initiates a memory-read command and receives the read-data from the serial memory device, the AHB Slave port of HS_SPI inserts WAIT states on the AHB Bus. Similarly, WAIT states are also inserted for serial memory write sequences.

HS_SPI keeps track of the previous address and the AHB transfer type issued by the AHB master. If the new transaction address is not contiguous or if there is switch in the command (from read to write or vice-versa), then a new command is issued on the serial interface.

6.7.5.5 AHB Idle Timeout

After a serial device is accessed in the Command Sequencer mode, HS_SPI keeps asserting the slave-select line even if the serial transaction is over. Within the time-period defined by HSSPIn_CSITIME:ITIME field, if a new AHB transaction is detected on AHB which has an address contiguous with the previous transaction and which has the same command (i.e. read / write), then the HS_SPI continues with the same serial transfer instead of initiating a new command sequence phase on the serial device, thus reducing the access time. If there are subsequent AHB accesses to a non-consecutive memory address during the idle time or if the command (i.e. read/write) changes, then even before the idle-timer expires, HS_SPI de-asserts the Slave Select (indicating the termination of the current transfer) and initiates the fresh transfer. If there are no subsequent AHB accesses to the consecutive memory address during the idle time, then after the idle-timer expires, HS_SPI de-asserts the Slave Select, indicating the termination of the transfer.

Thus, the HSSPIn_CSITIME:ITIME field is used to enhance the overall performance of the memory-mapped accesses by continuing the previous serial transaction for a configurable period.

The unit of the time-period defined by HSSPIn_CSITIME:ITIME field is the time-period of AHB Clock input.

6.7.5.6 Configuration of Command Sequence in CSR

Command Sequencer supports memory read accesses. Optionally, if Serial SRAM devices are memory mapped, the write Accesses by Command Sequencer can be enabled by setting HSSPIn_CSCFG:SRAM=1.

The sequence of command phases (i.e. instruction-phase, address-phase and data-phase) generated by HS_SPI in Command Sequencer Mode, on the SDATA lines is configured by the software (during initialization of HS_SPI) in the CSR.

6.7.5.6.1 Generation of Memory Read Command Sequence

For memory read transactions, the sequence of command phases can be configured in the list of 8 registers: HSSPIn_RDCSDC0, HSSPIn_RDCSDC1, HSSPIn_RDCSDC2, HSSPIn_RDCSDC3, HSSPIn_RDCSDC4, HSSPIn_RDCSDC5, HSSPIn_RDCSDC6 and HSSPIn_RDCSDC7. Each of the 8 registers in the list is parsed, starting from the HSSPIn_RDCSDC0, up to the HSSPIn_RDCSDC7 Please refer to Figure 6-72.

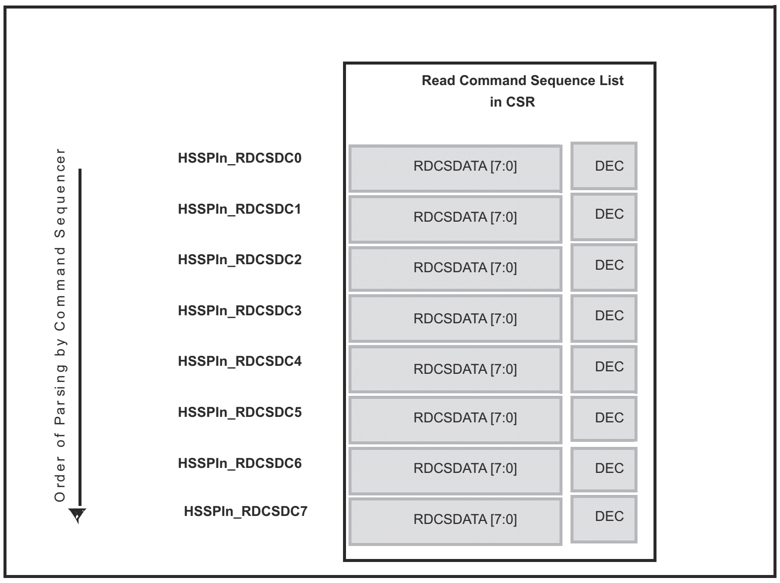


Figure 6-72: Memory Read Command Sequence List

The DEC bit in each of these registers indicates whether the data byte (i.e. RDCSDATA[2:0]) must be decoded (as shown in Table 6-25:), or whether the data byte in RDCSDATA[7:0] must be transmitted as-it-is.

.

Table 6-25: Decoding of the Read Command Sequence List

DEC	RDCSDATA [2:0]	Description
0	Don't Care	Transmit RDCSDATA[07:00] as-it-is.
1	000	Transmit address bits [07:00] of the serial memory to be accessed
1	001	Transmit address bits [15:08] of the serial memory to be accessed
1	010	Transmit address bits [23:16] of the serial memory to be accessed
1	011	Transmit address bits [31:24] of the serial memory to be accessed
1	100	Tri-state the SDATA output lines, for one Byte-Time
1	101	Irrespective of the shift-direction configured in HSSPIn_PCC0~3:SDIR bit, the transmission takes place in the following order: Transmit RDCSDATA[07:04] as-it-is. Transmit direction of this data will depend on the value of HSSPIn_PCC0~3:SDIR bit. Tri-state the SDATA output lines for 4 Bit-Times.
1	111	End of List

The Command Sequencer switches to data-read cycles if it gets "End of List" or after the HSSPIn_RDCSDC7 register, whichever occurs earlier. During data-read cycles, the serial data on SDATA lines is sampled and the assembled data is returned to the AHB master, in response to it's AHB Read transaction.

6.7.5.6.2 Generation of Memory Write Command Sequence

Memory write command sequences can be initiated by the Command Sequencer only if they are enabled, in HSSPIn_CSCFG:SRAM bit.

For memory write transactions, the sequence of command phases can be configured in the list of 8 registers: HSSPIn_WRCSDC0, HSSPIn_WRCSDC1, HSSPIn_WRCSDC2, HSSPIn_WRCSDC3, HSSPIn_WRCSDC4, HSSPIn_WRCSDC5, HSSPIn_WRCSDC6 and HSSPIn_WRCSDC7. Each of the 8 registers in the list is parsed, starting from the HSSPIn_WRCSDC0, up to the HSPWRCSDC7. Please refer to Figure 6-73.

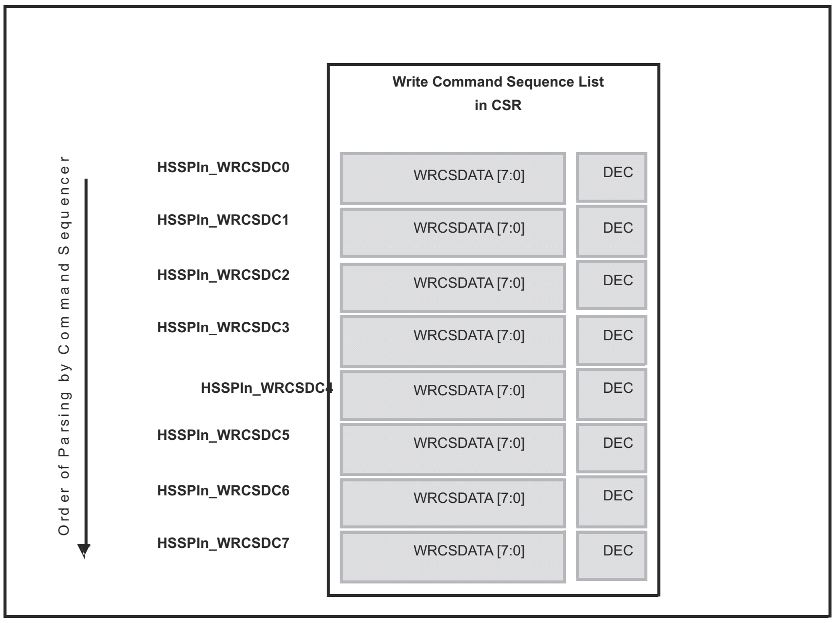


Figure 6-73: Memory Write Command SequenceList

The DEC bit in each of these registers indicates whether the data byte (i.e. WRCSDATA[2:0]) must be decoded (as shown in Table 6-26), or whether the data byte in WRCSDATA[7:0] must be transmitted as-it-is.

Table 6-26: Decoding of the Write Command Sequence List

DEC	WRCSDATA [2:0]	Description
0	Don't Care	Transmit WRCSDATA[07:00] as-it-is.
1	000	Transmit address bits [07:00] of the serial memory to be accessed
1	001	Transmit address bits [15:08] of the serial memory to be accessed
1	010	Transmit address bits [23:16] of the serial memory to be accessed
1	011	Transmit address bits [31:24] of the serial memory to be accessed
1	100	Tri-state the SDATA output lines, for one Byte-Time
1	101	Irrespective of the shift-direction configured in HSSPIn_PCC0~3:SDIR bit, the transmission takes place in the following order: Transmit RDCSDATA[07:04] as-it-is. Transmit direction of this data will depend on the value of HSSPIn_PCC0~3:SDIR bit. Tri-state the SDATA output lines for 4 Bit-Times.
1	111	End of List

The Command Sequencer switches to data-write cycles if it gets "End of List" or after the HSSPIn_WRCSDC7 register, whichever occurs earlier. During data-write cycles, the parallel data from the AHB Write Data Bus (i.e. HWDATA) is serially transmitted over the SDATA lines, as-per the configured SPI protocol.

6.7.6 Alternative SPI Interface

The output of the SPI module can be used in several configurations. Either with on clock output and a different CS signal for every SPI bus. Or without a CS signal and with a gated clock. For this, the Indigo2 internally masks the SDO and SCLK output signal with the dedicated CS signal and internally multiplexes the SDI input with the CS signals. Which kind of interface (or a combination of both interface kinds) can be selected with the pinmux (see section Pin Multiplexing). Together with the setting in the pinmux, an additional register has to be set that enables the correct input of the SPI interface (see DMSTART register).

NOTE An alternative SPI interface for the external Flash SPI is not possible.

NOTE The alternative SPI interface can not be used together with the retimed mode of the external SPI interface.

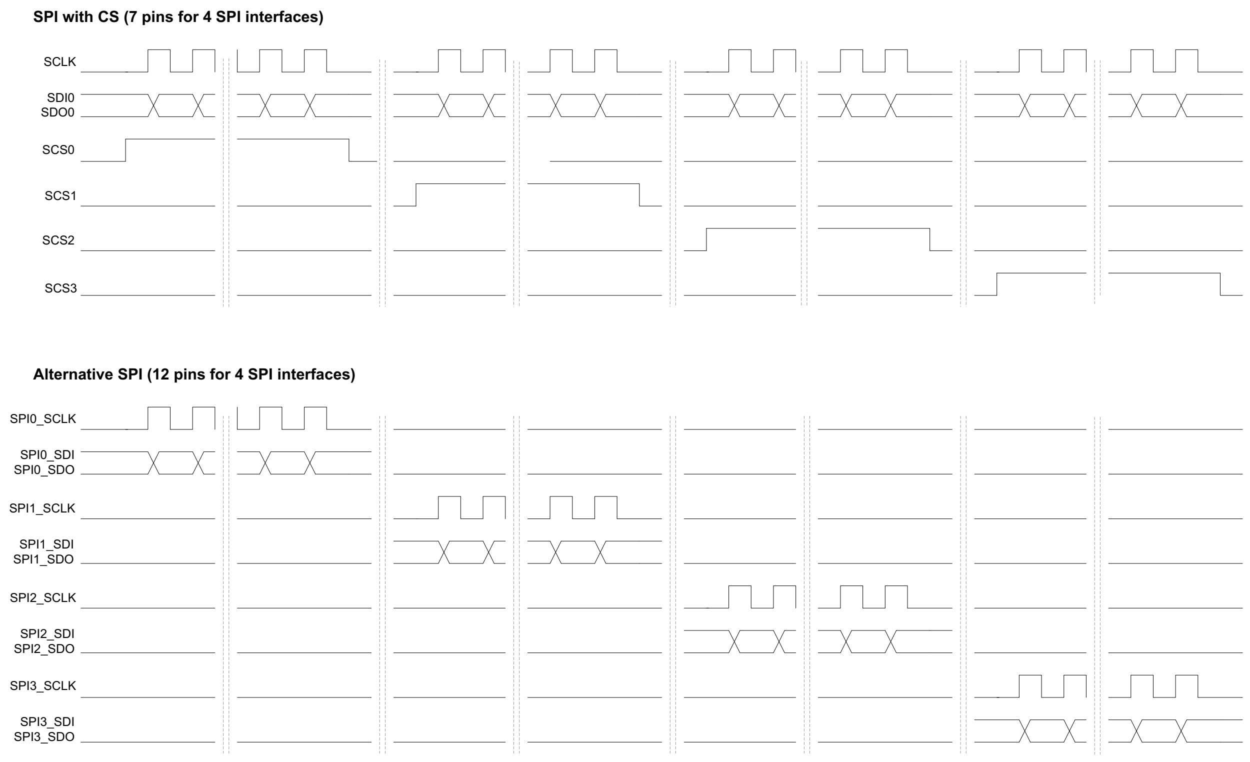


Figure 6-74: SPI interface (polarity of signals is programmable)

6.7.7 Notes on Using High-speed SPI Interface

This section is the “Programmer’s Guide”, which lists the usage notes for programming the HS_SPI module. Programmers are supposed to read these guidelines before programming the HS_SPI module.

6.7.7.1 General Usage Notes

- Any serial-transaction related parameters and control bits (e.g. selection of the attributes in the HSSPIn_PCC0~3 registers, switching between ‘Direct Mode’ and the ‘Command Sequencer Mode’ of operation) shall not be changed while a transaction is in progress. Any such changes shall be performed only after HS_SPI module is disabled (HSSPIn_MCTRL:MEN=0) and the current serial-transfer has ended (i.e. HSSPIn_TXF:TSSRS=1 or HSSPIn_RXF:RSSRS=1). To ensure that the HS_SPI module has finished all its transfers, the software can read the HSSPIn_DMSTATUS:TXACTIVE and the HSSPIn_DMSTATUS:RXACTIVE bits.
- For mimicking the transfer protocols of certain Serial Flash Memory devices (like the devices from Winbond) in ‘Direct Mode’ of Operation, it would be necessary to transfer initial set of bytes in legacy mode and then transfer the remaining sets of data bytes using the dual-bit or quad-bit mode. Thus, in such cases, it might be necessary to change the HSSPIn_DMTRP register while a serial transfer is in progress (i.e. while one of the Slave Select lines: SSEL0~3 is asserted). When the software has to re-program the HSSPIn_DMTRP register while a serial transfer has already started, it can do so after halting the current serial transfer (as explained in Section “Halting a Transfer Due To Lack Of TX-DATA or Due To Lack of RX-FIFO Space”).
- However, it is recommended that while the serial transfer has halted, the HSSPIn_DMTRP shall not be reprogrammed for switching from "TX-Only Legacy" to "RX-Only Legacy" mode and vice-versa. Instead of halting the serial transfer for switching the transfer protocol from "TX Only Legacy" to "RX Only Legacy, the software can program the HSSPIn_DMTRP for "TX and RX Legacy" mode before the start of transfer and then ignore the bytes in the data received while reception is not applicable, or transmit dummy data when transmission is not applicable.
- When ‘Direct Mode’ of operation is used, the software shall be responsible to take care that the internal FIFOs of HS_SPI do not get overrun or underrun. In case the FIFOs get overrun or underrun, the FIFO fill-levels (i.e. HSSPIn_DMSTATUS:TXFLEVEL and HSSPIn_DMSTATUS:RXFLEVEL) are no longer pertinent and the software would have to flush the FIFOs.
- The HSSPIn_FIFOCFG:RXFLSH and HSSPIn_FIFOCFG:TXFLSH bits shall be used by the software to flush the corresponding FIFOs before using them for any serial transfer. Flushing a FIFO ensures that they are not pre-loaded with any garbage data (possibly from the previous transfer).
- In ‘Direct Mode’, whenever the transfer ends, the data from the RX shift register is pushed into the RX-FIFO; provided that the RX-FIFO is not full.
  - If the RX FIFO is already full while a serial transfer is terminating, the serial transfer halts and the remainder data remains in the RX shift register as long as HS_SPI is halted. As soon as the RX FIFO is not full anymore, HS_SPI comes out of the halt state and the remainder data from the RX shift register is pushed into the RX-FIFO before HS_SPI releases the slave-select line. The remainder data is pushed into the RX-FIFO irrespective of whether the RX Shift Register is filled completely or partially.
  - Thus, in ‘Direct Mode’, remainder data never remains in the RX Shift Register. The bit-count from the HSSPIn_RXBITCNT register is always 0 and the received data is always pushed into the RX-FIFO.

6.7.7.2 Steps in Programming the HS_SPI Module

Figure 6-75 gives the general steps a programmer shall follow while using the HS_SPI module.

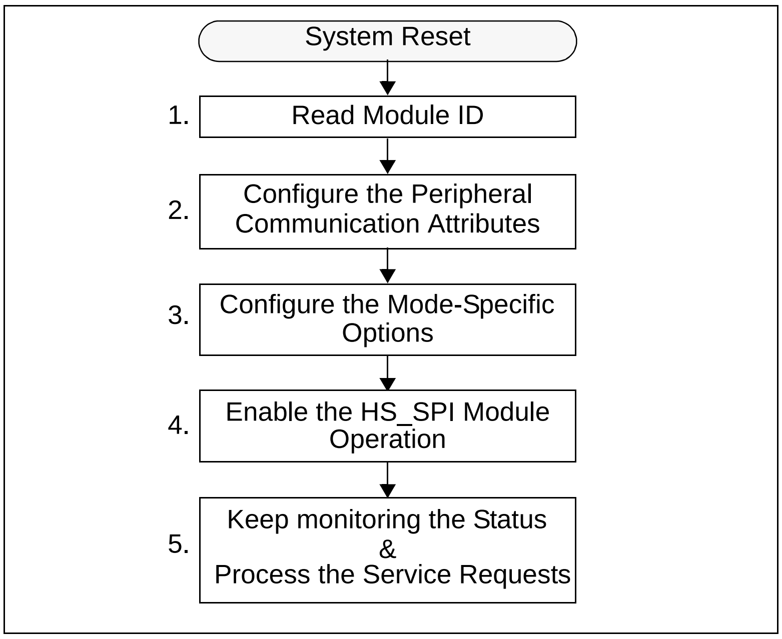

System Reset
1. Read Module ID
2. Configure the Peripheral Communication Attributes
3. Configure the Mode-Specific Options
4. Enable the HS_SPI Module Operation
5. Keep monitoring the Status & Process the Service Requests

Figure 6-75: Programmer’s Flowchart: General Steps

1. After the system reset, the software shall detect the Module ID number of HS_SPI, by reading the HSSPIn_MID register. This would help it in identifying the attributes and capabilities supported by the HS_SPI module.
2. The next step is to configure the attributes related to the peripheral communication with the serial device(s) connected with HS_SPI. HS_SPI can be interfaced with up to 4 serial devices. Serial communication related attributes like Clock Polarity, Clock Phase, Transfer Frequency (i.e. Clock Division Ratio and Clock Source Selection bits), Slave Select Polarity, etc. shall be configured in the registers: HSSPIn_PCC0, HSSPIn_PCC1, HSSPIn_PCC2 and HSSPIn_PCC3. It is very important that these attributes shall be the same as being used by the remote serial device with which HS_SPI is serially interfaced. These configurations shall not be modified while the HS_SPI module is active. In case the software has to re-program the values, the software shall first disable the HS_SPI module and wait until the current serial transfer is finished.
3. HS_SPI can be configured either in ‘Direct Mode’, or in ‘Command Sequencer Mode’ through the HSSPIn_MCTRL:CSEN bit. Depending on which mode is to be used, the software shall configure the mode-specific registers. The registers specific to the ‘Direct Mode’ are: (HSSPIn_TXF, HSSPIn_TXE, HSSPIn_TXC, HSSPIn_RXE, HSSPIn_RXF, HSSPIn_RXE, HSSPIn_RXC, HSSPIn_DMCFG, HSSPIn_DMSTATUS, HSSPIn_RXSHIFT, HSSPIn_TXFIFO0~15, HSSPIn_RXFIFO0~15 and HSSPIn_FIFOCFG) and the registers specific to the ‘Command Sequencer Mode’ are: (HSSPIn_CSCFG, HSSPIn_CSITIME, HSSPIn_CSAEXT, HSSPIn_RDCSDC0 - HSSPIn_RDCSDC7 and HSSPIn_WRCSDC0 - HSSPIn_WRCSDC7).
4. Only after all module-specific configurations are programmed, the HS_SPI module shall be enabled (by setting the HSSPIn_MCTRL:MEN to 1).

5. Once the HS_SPI module is enabled, its normal working begins. The software shall keep monitoring the status of the HS_SPI module using the various status bits. If the HS_SPI module is configured for initiating the service requests, it would periodically trigger the service requests (i.e. Interrupts and/or DMA requests). The software would service those requests, in order to ensure the normal working of HS_SPI.

6.7.7.3 Using the HS_SPI in ‘Direct Mode’ of Operation

Figure 6-76 gives the general steps a programmer shall follow while using the HS_SPI module in ‘Direct Mode’ of operation.

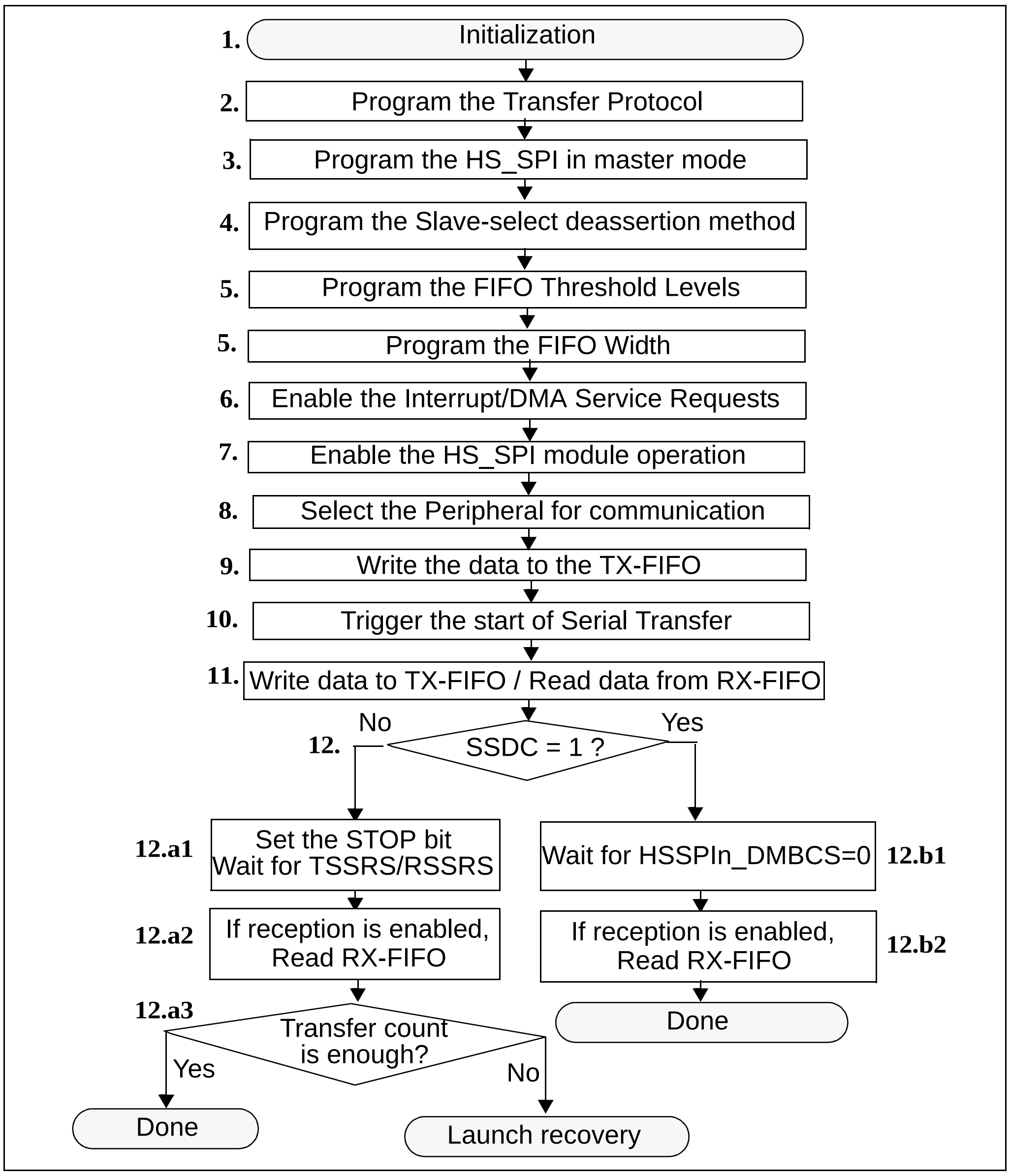

1. Initialization
2. Program the Transfer Protocol
3. Program the HS_SPI in master mode
4. Program the Slave-select deassertion method
5. Program the FIFO Threshold Levels
5. Program the FIFO Width
6. Enable the Interrupt/DMA Service Requests
7. Enable the HS_SPI module operation
8. Select the Peripheral for communication
9. Write the data to the TX-FIFO
10. Trigger the start of Serial Transfer
11. Write data to TX-FIFO / Read data from RX-FIFO
12. SSDC = 1 ?
No
Yes
12.a1 Set the STOP bit Wait for TSSRS/RSSRS
12.a2 If reception is enabled, Read RX-FIFO
12.a3 Transfer count is enough?
Yes
No
Done
Launch recovery
12.b1 Wait for HSSPIn_DMBCS=0
12.b2 If reception is enabled, Read RX-FIFO
Done

Figure 6-76: Programmer’s Flowchart: HS_SPI in ‘Direct Mode’ of operation

1. After the System Reset, the software shall initialize the HS_SPI module by reading the HSSPIn_MID register and setting the peripheral communication related attributes in the HSSPIn_PCC0, HSSPIn_PCC1n, HSSPIn_PCC2n and HSSPIn_PCC3 registers. It is very important that these attributes shall be the same as being used by the remote serial device with which HS_SPI is serially interfaced. Ensure that the HSSPIn_MCTRL:CSEN bit is reset to “0”.
2. The next step is to configure the transfer protocol (i.e. whether the HS_SPI serial transfers use the Legacy, dual-bit or the quad-bit SPI protocol and whether the HS_SPI would be used only for transmission, or only for reception or for both: transmission and reception) in the HSSPIn_DMTRP:TRP field.
3. Configure its HSSPIn_DMCFG:SSDC field. This selects how the Slave Select output is to be deasserted. If Byte-Counter mode is used, load the HSSPIn_DMBCC:BCC field with the number of bytes to be serially transferred. If the Software Flow Control is used, the software is responsible to set the HSSPIn_DMSTOP:STOP bit after it has finished transmission/reception of the desired data.
4. Configure the HSSPIn_FIFOCFG register, to set the FIFO threshold levels. By programming these levels, the assertion of the service requests can be controlled. Also configure the HSSPIn_FIFOCFG:FWIDTH field, to select the width of the FIFOs.
5. Configure the service requests. HS_SPI supports both: Interrupt and DMA service requests, for the normal data read/write operations from/to the internal FIFOs. For normal operation, either the interrupt requests or the DMA requests shall be enabled by the software.
   - To enable the interrupt service requests for TX-FIFO write requests, please program the bits in the HSSPIn_TXE register.
   - To enable the interrupt service requests for RX-FIFO read requests, please program the bits in the HSSPIn_RXE register.
   - To enable the DMA read and/or DMA write service requests, please program either/both of the HSSPIn_DMDMAEN:TXDMAEN and the HSSPIn_DMDMAEN:RXDMAEN bits. The DMA Read Channel must be setup to perform a block transfer of "HSSPIn_FIFOCFG:RXFTH + 1" transfers. The DMA Write Channel must be setup to perform a block transfer of "16 - HSSPIn_FIFOCFG:TXFTH" transfers.
6. This finishes the steps in initialization of HS_SPI for ‘Direct Mode’ of operation. Set the HSSPIn_MCTRL:MEN bit, to enable the module.
7. Select the peripheral (in HSSPIn_DMPSEL:PSEL field) on which HS_SPI shall initiate the transfer.
8. If HS_SPI is configured for TX-Only or TX-and-RX mode of operation (in HSSPIn_DMTRP:TRP field), then write the data to be transmitted in the TX-FIFO by performing write accesses to the HSSPIn_TXFIFO0~15 register address. Before writing to the HSSPIn_TXFIFO0~15 register, modify the value of the HSSPIn_FIFOCFG:TXCTRL field appropriately. Generally (i.e when the data being written to the TX-FIFO is to be transmitted as-it-is), the HSSPIn_FIFOCFG:TXCTRL bit shall be reset to “0”. Only when HS_SPI is to be instructed to tri-state the serial data lines for a byte-time or for 4 bit-times, the HSSPIn_FIFOCFG:TXCTRL bit shall be set to “1” and a HSSPIn_TXFIFO0~15 write access shall be performed.
9. Setting the HSSPIn_DMSTART:START bit triggers the start of the serial transaction.
10. Once the serial transaction starts, if transmission is enabled in the HSSPIn_DMTRP:TRP field, the HS_SPI reads the TX-FIFO and loads the shift-register. The shift-register is shifted either left or right (based on configuration in HSSPIn_PCC0~3:SDIR field and the transmit data is shifted-out onto the serial line(s). If HS_SPI is enabled for Receive operation (in HSSPIn_DMTRP:TRP field), the HS_SPI assembles the received data by serially shifting the received data into the shift register. The received data assembled in the Shift Register is shifted into the RX-FIFO.

11. Service requests are asserted by HS_SPI whenever the TX-FIFO level is below the threshold or whenever the HS_SPI RX-FIFO level is above the threshold. The software shall read/write the FIFOs, to ensure normal operation of HS_SPI. Once processed, the interrupt service requests shall be cleared by the software in the HSSPIn_TXC or the HSSPIn_RXC registers. DMA service requests are automatically cleared by HS_SPI.
12. For stopping the serial transfers, the software can use either of the two modes (configured in HSSPIn_DMCFG:SSDC bit) - ‘Software Flow Control Mode’ or the ‘Byte Counter Mode’.

In ‘Software Flow Control Mode’ (13 a):

- In ‘Software Flow Control Mode’, software waits till either of the HSSPIn_TXF:TSSRS or the HSSPIn_RXF:RSSRS flag is set, indicating that the slave select is released.
- If reception is enabled in HSSPIn_DMTRP register, then the software fetches the received data from the RX-FIFO.
- If the number of byte of data transferred using the ‘Software Flow Control Mode’ is not enough, then the software launches its own recovery.

In ‘Byte Counter Mode’ (12 b):

- In ‘Byte Counter Mode’, software waits till the HSSPIn_DMBCS register value becomes 0.
- If reception is enabled in HSSPIn_DMTRP register, then the software fetches the received data from the RX-FIFO.

13. In the normal course of action, the software usually keeps repeating steps from 9 to 12, or it can loop back to the initialization step.
14. To switch between the ‘Direct Mode’ to the ‘Command Sequencer Mode’ of operation or to re-program any of the parameters that directly affect the serial transfer, the software shall first stop the current transfer and disable the HS_SPI module. Only after it is ensured that the HS_SPI module has finished all its transfers (by reading the HSSPIn_DMSTATUS:TXACTIVE and the HSSPIn_DMSTATUS:RXACTIVE bits), the software can reconfigure the module.

6.7.7.4 Using the Memory Mapped Memories

Following usage rules shall be followed, when interfacing serial memories, for memory-mapped accesses in ‘Command Sequencer Mode’.

6.7.7.4.1 Usage Rules and Notes

- In ‘Command Sequencer Mode’, all serial memory devices interfaced with HS_SPI shall be of same family. Do not mix serial memory devices from different vendor families.
- If memory devices of same family, but with different memory sizes are to be interfaced, then while deciding the suitable value of the HSSPIn_CSCFG:MSEL field, the memory device with maximum size must be considered. However, it shall be noted here, that the number of bytes from the final memory address that will be transmitted by HS_SPI to the serial memory-mapped device is programmed in the Command Sequence lists (in HSSPIn_RDCSDC0~7 and HSSPIn_WRCSDC0~7 registers). Interfacing of one memory device which requires 32-bit addressing and other memory device (also of same family, but) which requires only 24-bit addressing in ‘Command Sequencer Mode’ is not possible. This is because a memory device which has 24-bit addressing cannot be used with a bit-stuffed 32-bit address - its address phase is of 3 cycles only.
- If a memory device only needs 21 bit addressing, and if the Command Sequence in HS_SPI is configured to transmit a 24-bit address to the memory device, then the three most significant bits [23:21] shall be bit-stuffed with 0s (using the HSSPIn_CSAEXT register) by the software. If the unused most-significant bits are not reset to 0s, then the address pointers in the serial Memory Devices interfaced with HS_SPI might wrap, causing unwanted results.
- Serial SRAM devices support burst-operation only when they are configured to work in burst-mode. However, the Command Sequencer always assumes that the SRAM device is in burst mode. Before enabling the ‘Command Sequencer Mode’ of HS_SPI, the software shall configure the serial SRAM device (using ‘Direct Mode’ of operation), for burst mode of operation.
- In ‘Command Sequencer Mode’, it is not possible to change between the legacy, dual-bit or quad-bit modes when a transfer has started. For this reason, for some of the newer Serial Flash devices - like the memory devices from Winbond, the Command Sequencer can be enabled only after the device has been initialized to work in the ‘Continuous Read Mode’. The memory device can be programmed in the ‘Continuous Read Mode’, using the ‘Direct Mode’ of operation of HS_SPI.

6.7.7.4.2 Programmer’s Flowchart

Figure 6-77 gives the general steps a programmer shall follow while using the HS_SPI for memory-mapping the serial memory devices onto the address space of the MB88F33x ‘Indigo2(-x)’.

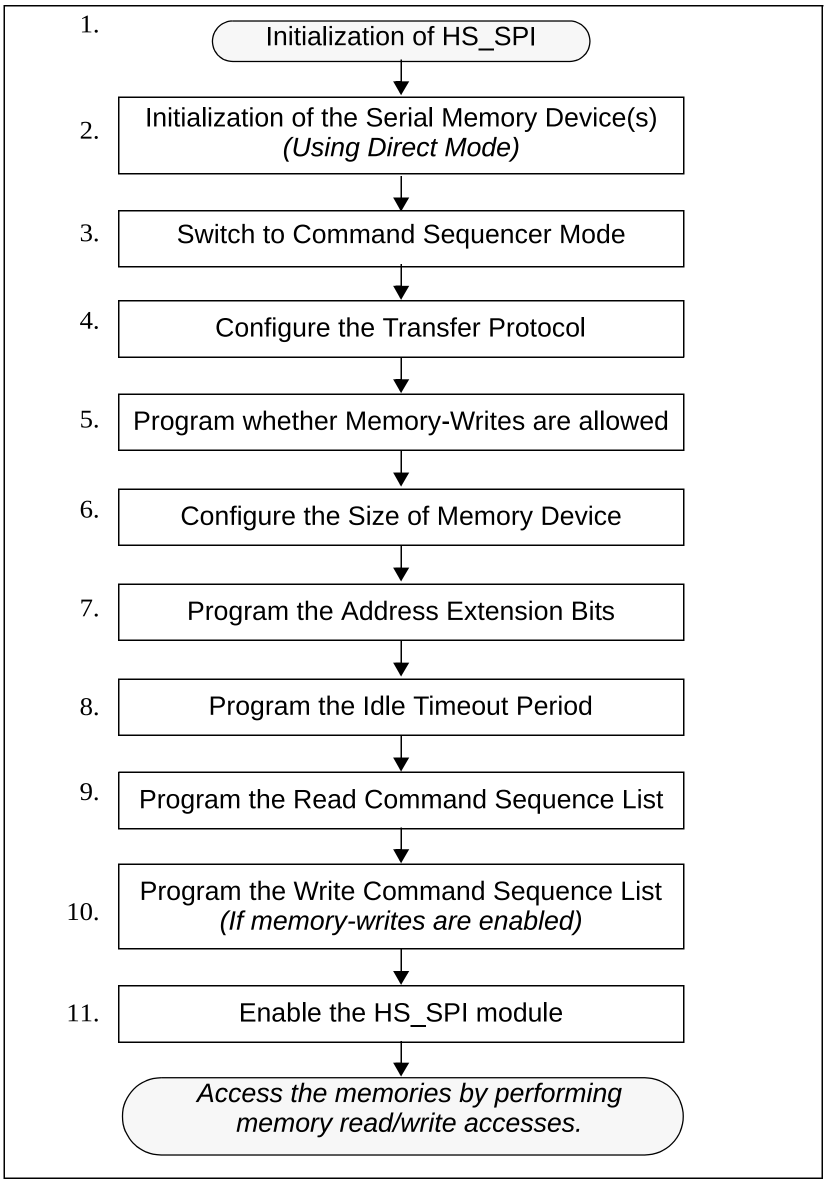

1.
Initialization of HS_SPI
2.
Initialization of the Serial Memory Device(s)
(Using Direct Mode)
3.
Switch to Command Sequencer Mode
4.
Configure the Transfer Protocol
5.
Program whether Memory-Writes are allowed
6.
Configure the Size of Memory Device
7.
Program the Address Extension Bits
8.
Program the Idle Timeout Period
9.
Program the Read Command Sequence List
10.
Program the Write Command Sequence List
(If memory-writes are enabled)
11.
Enable the HS_SPI module
Access the memories by performing memory read/write accesses.

Figure 6-77: Programmer’s Flowchart: Memory Mapping of Serial Memory Devices

1. After the system reset, the software shall initialize the HS_SPI module by setting the peripheral communication related attributes in the HSSPIn_PCC0, HSSPIn_PCC1n, HSSPIn_PCC2n and HSSPIn_PCC3 registers. It is very important that these attributes shall be the same as being used by the remote serial device with which HS_SPI is serially interfaced. When serial memory devices are to be memory-mapped using ‘Command-Sequencer Mode’, all memory devices shall be of same family. Therefore, all 4 peripheral communication configuration registers (i.e HSSPIn_PCC0~3) shall have same configuration values.

2. The next step is to initialize the serial device that is to be memory mapped. The initialization is device specific and it may include setting of some control/status bits in its register-set. e.g. To use a Quad-SPI serial memory device, you might want to set the device in a high-performance (i.e. Quad mode). Please consult the data sheet of the serial memory device to be interfaced. This initialization of the serial memory device shall be performed using ‘Direct Mode’ of HS_SPI.
3. After initializing the serial device (using ‘Direct Mode’ of operation), re-program the HS_SPI module in the ‘Command Sequencer Mode’. To switch from the ‘Direct Mode’ of operation to the ‘Command Sequencer Mode’, the software shall first stop the current transfer and disable the HS_SPI module. Only after it is ensured that the HS_SPI module has finished all its transfers (by reading the HSSPIn_DMSTATUS:TXACTIVE and the HSSPIn_DMSTATUS:RXACTIVE bits), the software can reconfigure the module to ‘Command Sequencer Mode’.
4. The next step is to configure the transfer protocol (i.e. whether the HS_SPI serial transfers use the Legacy, dual-bit or the quad-bit SPI protocol in the HSSPIn_CSCFG:MBM field.
5. If serial SRAM devices are connected, you might want to enable the write-accesses to these memory-mapped devices, using the HSSPIn_CSCFG:SRAM bit. If serial flash devices are connected, do not enable the write-accesses.
6. Program the HSSPIn_CSCFG:MSEL field, with the size of the AHB address space which must be used in selection of the Memory Device on which the serial transfer must be initiated. Please refer to Section “Command Sequencer Mode” for details of the Slave Selection logic.
7. If the addresses generated for the memory-mapped accesses are to be virtually extended to cover a memory range of virtually 16GB, you might have to program the HSSPIn_CSAEXT register. Please refer to Section “Command Sequencer Mode” for details of address generation.
8. The HSSPIn_CSITIME:ITIME field is used to enhance the overall performance of the memory-mapped accesses by continuing the previous serial transaction for a configurable period. If there is an access (of same type: i.e. read/write) to the consecutive memory address, HS_SPI proceeds with the same serial transfer (without having to issue a new command/address cycles), thus reducing the access time. Program the HSSPIn_CSITIME:ITIME with appropriate idle time-out value.
9. Program the list of Read Command Sequence registers (i.e. HSSPIn_RDCSDC0, HSSPIn_RDCSDC1, HSSPIn_RDCSDC2, HSSPIn_RDCSDC3, HSSPIn_RDCSDC4, HSSPIn_RDCSDC5, HSSPIn_RDCSDC6 and HSSPIn_RDCSDC7) with the sequence of the memory read command for the memory device you have interfaced. Please consult with the device-specific data sheet for details of the read command sequence.
10. If memory-write accesses are also enabled in HSSPIn_CSCFG:SRAM bit, then you also need to program the list of Write Command Sequence registers (i.e. HSSPIn_WRCSDC0, HSSPIn_WRCSDC1, HSSPIn_WRCSDC2, HSSPIn_WRDCSDC3n, HSSPIn_WRCSDC4, HSSPIn_WRCSDC5, HSSPIn_WRCSDC6 and HSSPIn_WRCSDC7) with the sequence of the memory write command for the memory device you have interfaced. Please consult with the device-specific data sheet for details of the write command sequence.
11. With this, you have configured the HS_SPI module for accessing the memory-mapped devices. Enable the HS_SPI module (in HSSPIn_MCTRL:MEN bit), so that it starts generating the read/write sequences on the serial interface, by mapping the AHB accesses to the memory-mapped locations.

6.7.7.4.3 Timing Diagram for Command Sequencer

Figure 6-78 illustrates with an example, how the Command Sequencer generates the serial memory read command sequence. Let us assume, that the Read Command Sequence list is programmed in HSSPIn_RDCSDC0 through HSSPIn_RDCS5 registers, as shown in the figure. The Command Sequencer parses the list, starting from HSSPIn_RDCS0 register, and executes the commands as explained in Section “Command Sequencer Mode”.

Figure 6-78 shows the corresponding timing diagram for a Read Command Sequence, for a Clocking Mode of “0”.

Read Command Sequence List in CSR

Order of Parsing by Command Sequencer

	RDCSDATA [7:0]	DEC	
HSSPIn_RDCSDC0	(0000 1011) i.e. 0x0B	0	Instructio: High-Speed Read
HSSPIn_RDCSDC1	xxxx x010	1	Memory Address [24:16]
HSSPIn_RDCSDC2	xxxx x001	1	Memory Address [15:08]
HSSPIn_RDCSDC3	xxxx x000	1	Memory Address [07:00]
HSSPIn_RDCSDC4	xxxx x100	1	High-Z Byte
HSSPIn_RDCSDC5	xxxx x111	1	End of List
HSSPIn_RDCSDC6	xxxx xxxx	x	Unused
HSSPIn_RDCSDC7	xxxx xxxx	x	Unused

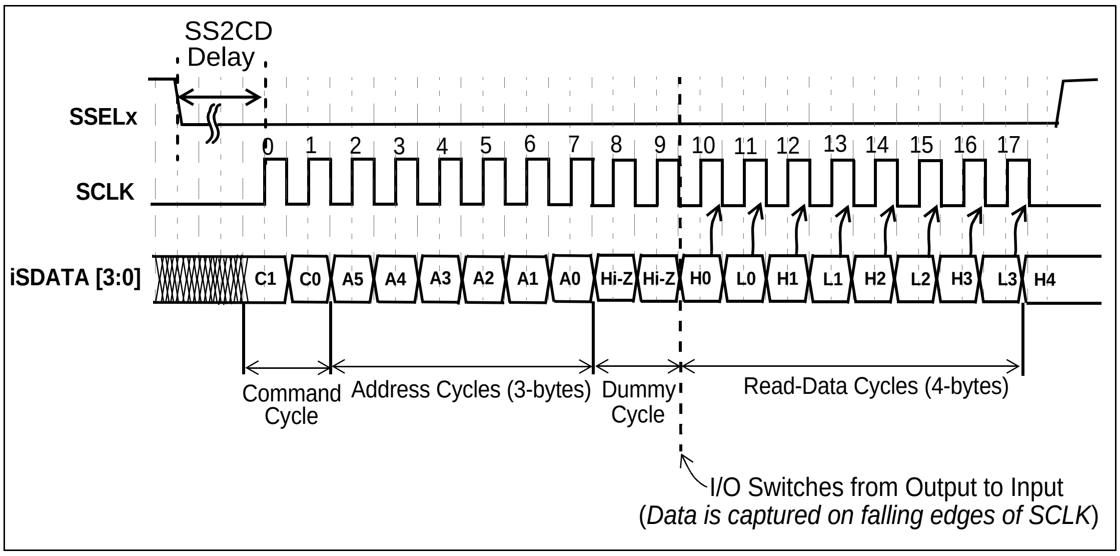

SS2CD Delay
SSELx
SCLK
iSDATA [3:0]
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17
C1 C0 A5 A4 A3 A2 A1 A0 Hi-Z Hi-Z H0 L0 H1 L1 H2 L2 H3 L3 H4
Command Cycle
Address Cycles (3-bytes)
Dummy Cycle
Read-Data Cycles (4-bytes)
I/O Switches from Output to Input
(Data is captured on falling edges of SCLK)

Figure 6-78: Read Command Sequence Illustration With Timing Diagram (*Mode-0*)

6.7.8 High-speed SPI Interface Register Overview

Table 6-27: Registers Overview

Base Address(es)	**Instance no 0: BASEADDR0="00026000" Instance no 1: BASEADDR1="000B1000"**	
Absolute Address	**Register Name**	**Register Description**
BASEADDRx + 0x0000	MCTRL	HS_SPI Module Control Register
BASEADDRx + 0x0004	PCC0	HS_SPI Peripheral Communication Configuration Register 0
BASEADDRx + 0x0008	PCC1	HS_SPI Peripheral Communication Configuration Register 1
BASEADDRx + 0x000C	PCC2	HS_SPI Peripheral Communication Configuration Register 2
BASEADDRx + 0x0010	PCC3	HS_SPI Peripheral Communication Configuration Register 3
BASEADDRx + 0x0014	TXF	HS_SPI TX Interrupt Flag Register
BASEADDRx + 0x0018	TXE	HS_SPI TX Interrupt Enable Register
BASEADDRx + 0x001C	TXC	HS_SPI TX Interrupt Clear Register
BASEADDRx + 0x0020	RXF	HS_SPI RX Interrupt Flag Register
BASEADDRx + 0x0024	RXE	HS_SPI RX Interrupt Enable Register
BASEADDRx + 0x0028	RXC	HS_SPI RX Interrupt Clear Register
BASEADDRx + 0x002C	FAULTF	HS_SPI Fault Interrupt Flag Register
BASEADDRx + 0x0030	FAULTC	HS_SPI Fault Interrupt Clear Register
BASEADDRx + 0x0034	DMCFG	HS_SPI Direct Mode Configuration Register
BASEADDRx + 0x0035	DMDMAEN	HS_SPI Direct Mode DMA Enable Register
BASEADDRx + 0x0036	SVCFG0	Supervision Config Register 0
BASEADDRx + 0x0037	SVCFG1	Supervision Config Register 1
BASEADDRx + 0x0038	DMSTART	HS_SPI Direct Mode Start Register
BASEADDRx + 0x0039	DMSTOP	HS_SPI Direct Mode Stop Register
BASEADDRx + 0x003A	DMPSEL	HS_SPI Direct Mode Peripheral Select Register
BASEADDRx + 0x003B	DMTRP	HS_SPI Direct Mode Transfer Protocol Register
BASEADDRx + 0x003C	DMBCC	HS_SPI Direct Mode Byte Count Control Register
BASEADDRx + 0x003E	DMBCS	HS_SPI Direct Mode Byte Count Status Register
BASEADDRx + 0x0040	DMSTATUS	HS_SPI Direct Mode Status Register
BASEADDRx + 0x0044	TXBITCNT	HS_SPI Transmit Bit Count Register
BASEADDRx + 0x0045	RXBITCNT	HS_SPI Receive Bit Count Register
BASEADDRx + 0x0046	RXSHIFT	HS_SPI RX Shift Register
BASEADDRx + 0x004A	FIFOCFG	HS_SPI FIFO Configuration Register
BASEADDRx + 0x004E	TXFIFO0	HS_SPI TX-FIFO Registers 0
BASEADDRx + 0x0052	TXFIFO1	HS_SPI TX-FIFO Registers 1
BASEADDRx + 0x0056	TXFIFO2	HS_SPI TX-FIFO Registers 2
BASEADDRx + 0x005A	TXFIFO3	HS_SPI TX-FIFO Registers 3
BASEADDRx + 0x005E	TXFIFO4	HS_SPI TX-FIFO Registers 4
BASEADDRx + 0x0062	TXFIFO5	HS_SPI TX-FIFO Registers 5
BASEADDRx + 0x0066	TXFIFO6	HS_SPI TX-FIFO Registers 6
BASEADDRx + 0x006A	TXFIFO7	HS_SPI TX-FIFO Registers 7
BASEADDRx + 0x006E	TXFIFO8	HS_SPI TX-FIFO Registers 8
BASEADDRx + 0x0072	TXFIFO9	HS_SPI TX-FIFO Registers 9
BASEADDRx + 0x0076	TXFIFO10	HS_SPI TX-FIFO Registers 10
BASEADDRx + 0x007A	TXFIFO11	HS_SPI TX-FIFO Registers 11
BASEADDRx + 0x007E	TXFIFO12	HS_SPI TX-FIFO Registers 12
BASEADDRx + 0x0082	TXFIFO13	HS_SPI TX-FIFO Registers 13
BASEADDRx + 0x0086	TXFIFO14	HS_SPI TX-FIFO Registers 14

Table 6-27: Registers Overview

Base Address(es)	**Instance no 0: BASEADDR0="00026000" Instance no 1: BASEADDR1="000B1000"**	
Absolute Address	**Register Name**	**Register Description**
BASEADDRx + 0x008A	TXFIFO15	HS_SPI TX-FIFO Registers 15
BASEADDRx + 0x008E	RXFIFO0	HS_SPI RX-FIFO Registers 0
BASEADDRx + 0x0092	RXFIFO1	HS_SPI RX-FIFO Registers 1
BASEADDRx + 0x0096	RXFIFO2	HS_SPI RX-FIFO Registers 2
BASEADDRx + 0x009A	RXFIFO3	HS_SPI RX-FIFO Registers 3
BASEADDRx + 0x009E	RXFIFO4	HS_SPI RX-FIFO Registers 4
BASEADDRx + 0x00A2	RXFIFO5	HS_SPI RX-FIFO Registers 5
BASEADDRx + 0x00A6	RXFIFO6	HS_SPI RX-FIFO Registers 6
BASEADDRx + 0x00AA	RXFIFO7	HS_SPI RX-FIFO Registers 7
BASEADDRx + 0x00AE	RXFIFO8	HS_SPI RX-FIFO Registers 8
BASEADDRx + 0x00B2	RXFIFO9	HS_SPI RX-FIFO Registers 9
BASEADDRx + 0x00B6	RXFIFO10	HS_SPI RX-FIFO Registers 10
BASEADDRx + 0x00BA	RXFIFO11	HS_SPI RX-FIFO Registers 11
BASEADDRx + 0x00BE	RXFIFO12	HS_SPI RX-FIFO Registers 12
BASEADDRx + 0x00C2	RXFIFO13	HS_SPI RX-FIFO Registers 13
BASEADDRx + 0x00C6	RXFIFO14	HS_SPI RX-FIFO Registers 14
BASEADDRx + 0x00CA	RXFIFO15	HS_SPI RX-FIFO Registers 15
BASEADDRx + 0x00CE	CSCFG	HS_SPI Command Sequencer Configuration Register
BASEADDRx + 0x00D2	CSITIME	HS_SPI Command Sequencer Idle Time Register
BASEADDRx + 0x00D6	CSAEXT	HS_SPI Command Sequencer Address Extension Register
BASEADDRx + 0x00DA	RDCSDC0	HS_SPI Read Command Sequence Data/Control Register 0
BASEADDRx + 0x00DC	RDCSDC1	HS_SPI Read Command Sequence Data/Control Register 1
BASEADDRx + 0x00DE	RDCSDC2	HS_SPI Read Command Sequence Data/Control Register 2
BASEADDRx + 0x00E0	RDCSDC3	HS_SPI Read Command Sequence Data/Control Register 3
BASEADDRx + 0x00E2	RDCSDC4	HS_SPI Read Command Sequence Data/Control Register 4
BASEADDRx + 0x00E4	RDCSDC5	HS_SPI Read Command Sequence Data/Control Register 5
BASEADDRx + 0x00E6	RDCSDC6	HS_SPI Read Command Sequence Data/Control Register 6
BASEADDRx + 0x00E8	RDCSDC7	HS_SPI Read Command Sequence Data/Control Register 7
BASEADDRx + 0x00EA	WRCSDC0	HS_SPI Write Command Sequence Data/Control Register 0
BASEADDRx + 0x00EC	WRCSDC1	HS_SPI Write Command Sequence Data/Control Register 1
BASEADDRx + 0x00EE	WRCSDC2	HS_SPI Write Command Sequence Data/Control Register 2
BASEADDRx + 0x00F0	WRCSDC3	HS_SPI Write Command Sequence Data/Control Register 3
BASEADDRx + 0x00F2	WRCSDC4	HS_SPI Write Command Sequence Data/Control Register 4
BASEADDRx + 0x00F4	WRCSDC5	HS_SPI Write Command Sequence Data/Control Register 5
BASEADDRx + 0x00F6	WRCSDC6	HS_SPI Write Command Sequence Data/Control Register 6

Table 6-27: Registers Overview

Base Address(es)	Instance no 0: BASEADDR0="00026000" Instance no 1: BASEADDR1="000B1000"	
Absolute Address	**Register Name**	**Register Description**
BASEADDRx + 0x00F8	WRCSDC7	HS_SPI Write Command Sequence Data/Control Register 7
BASEADDRx + 0x00FA	MID	HS_SPI Module ID Register

6.8 Programmable Pulse Generators (PPG)

Programmable Pulse Generators (PPGs) are used to obtain one-shot (rectangular wave) output or pulse width modulation (PWM) output. With their software-programmable cycle and duty capability and possibility to postpone the start of PWM output signal generation by software the PPGs comfortably fit into a broad range of applications. To increase flexibility, the PPGs can be configured as a 16-bit resolution PWM channel or two independent 8-bit resolution PWM outputs. Moreover, the PPGs can operate in a ramp mode, changing the output signal duty between defined start duty and end duty values.

The MB88F33x ‘Indigo2(-x)’ has in total 16 PPGs with 16-bit resolution each. Each of the 16-bit PPG can be split into 2 PPGs with a 8-bit resolution. Thus, a maximum of 32 PPGs can be supported. For more information please refer to section “1.5 Pinning”.

6.8.1 Features of the PPG / PWM Signals

- PWM waveform output
- One-shot waveform (Rectangular wave)
- Clamped output (high or low)
- Ramp mode operation
- 16bit or 8bit resolution
- 1, 1/4, 1/16, 1/64 of the rbus clock
- Interrupt generation: Choose from six choices
  - Software trigger or internal trigger
  - Counter borrow (cycle match)
  - Duty match
  - Counter borrow (cycle match) or duty match
  - Defined timing point match within the PPG cycle
  - End duty match during the ramp mode operation
- Activation trigger:
  - Individual software trigger for each PPG/PWM
  - Internal trigger (from Reload Timer or from IRIS frame generator)
  - Common internal software trigger, able to trigger all available PPG resources
  - Internal software trigger, able to trigger all available PPG resources of one group
- DMA request can be generated for two PPGs (PPGo and PPG1)

6.8.2 Block Diagram of the Programmable Pulse Generator

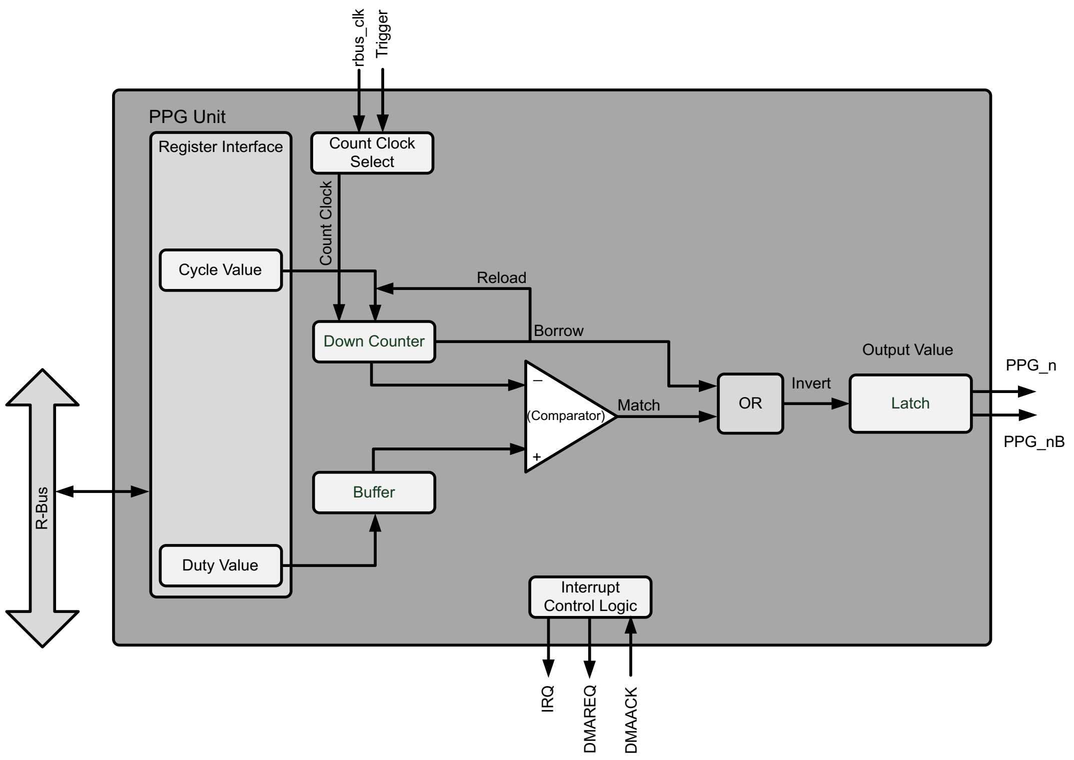


Figure 6-79: Simplified block diagram of one Programmable Pulse Generator

Indigo2 has 16 Pulse Generators in total, which are organized in 4 Groups. Each group has one Group Control Register (GCTRL). There is one common general control register (GCNR) for all PPG.

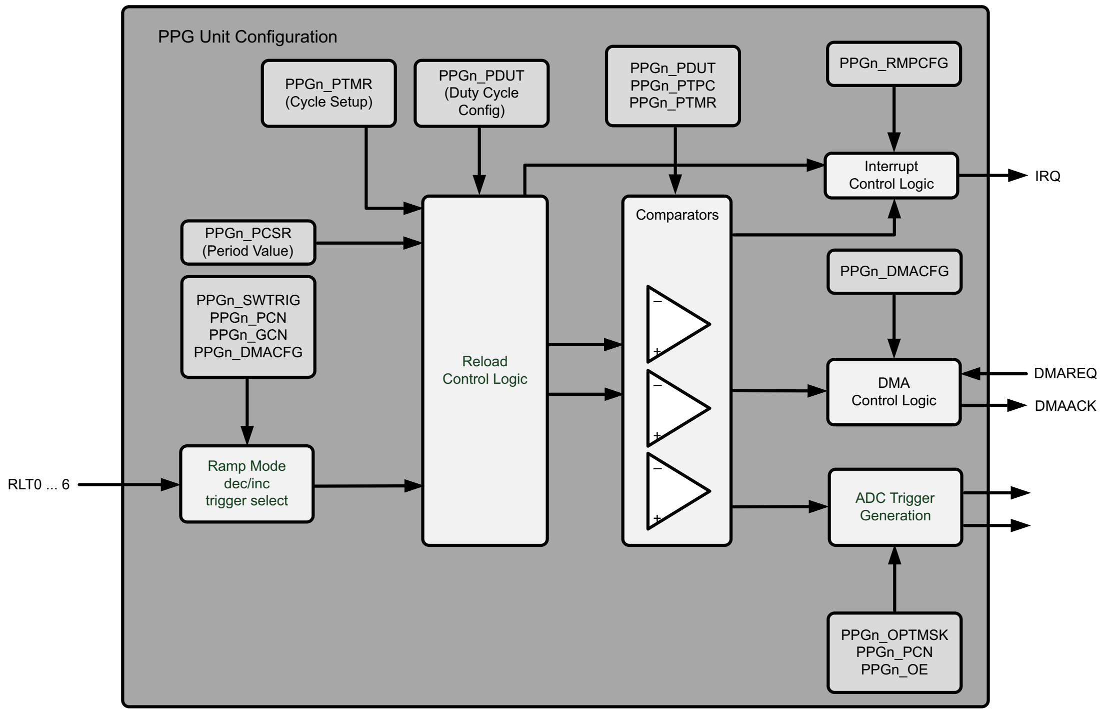


Figure 6-80: Configuration diagram of Programmable Pulse Generator

6.8.3 Operation of Programmable Pulse Generator

The Programmable Pulse Generators (PPGs) provide programmable pulse output independently or jointly. The individual modes of operation are described below.

6.8.3.1 PWM Operation

In PWM operation, variable-duty pulses are generated at the PPG pin.

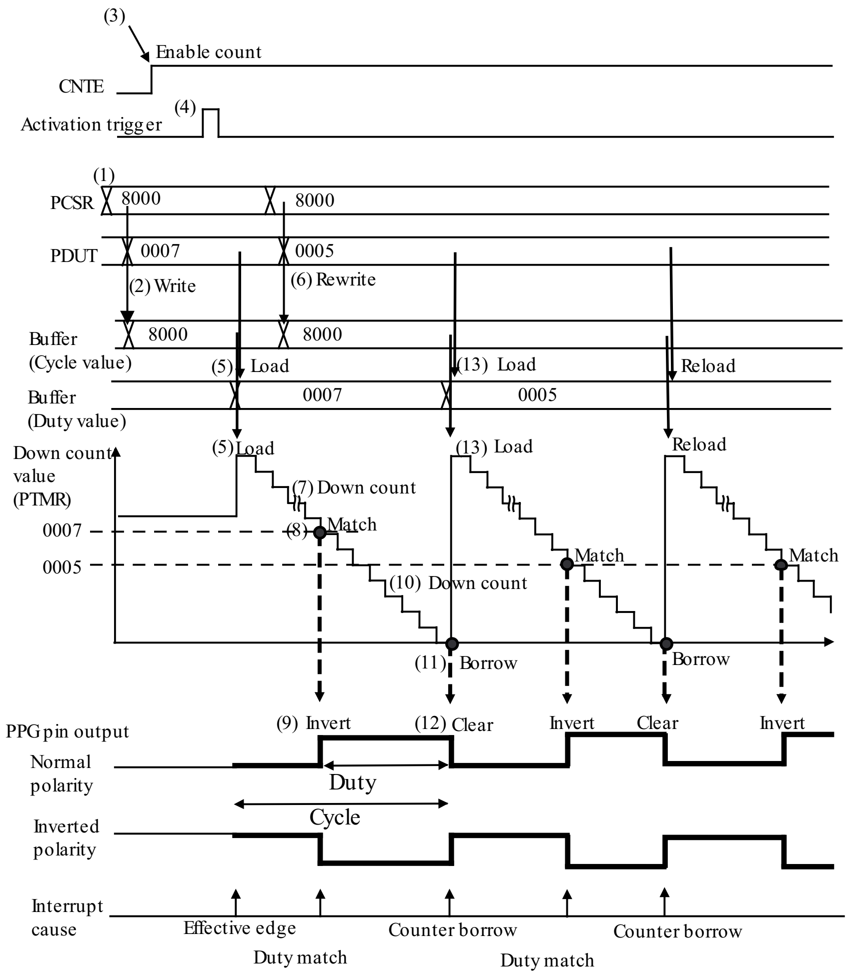


Figure 6-81: Variable-duty pulses

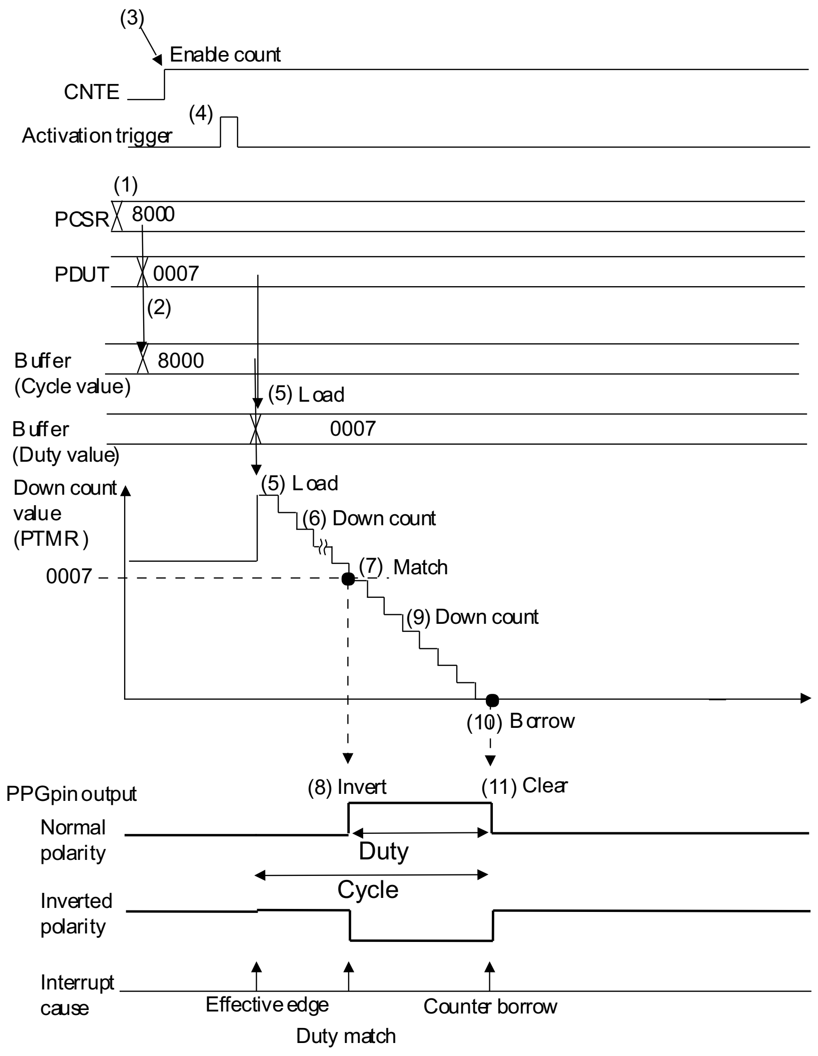
1. Write a cycle value.
2. Write a duty value and transfer the cycle value to buffers.
3. Enable PPG operation.
4. Generate an activation trigger.
5. Load the cycle and duty values.
6. Rewrite the duty value and transfer the cycle value to buffers.
7. Counter down count.
8. The down counter equals the duty value.

9. Inverses the PPG pin output level.
10. Counter down count.
11. Counter borrow.
12. Clear the PPG pin output level (return to normal).
13. Reload the cycle value.
14. Reload the duty value.
15. Steps from (7) to (14) are iterated.

- Equation:
  - Period = {Period value (PCSR) + 1} x Count clock
  - Duty = {Duty value (PDUT) + 1} x Count clock
  - Width up to pulse output = {Period value (PCSR) – Duty value (PDUT)} x Count clock

6.8.3.2 One-Shot Operation

In one-shot operation, one-shot pulses are generated at the PPG pin. One-shot operation cannot be used in 8-bit mode or together with start delay feature.

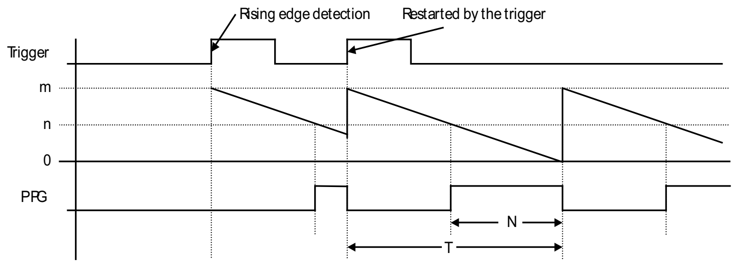


1. Write a cycle value.
2. Write a duty value and transfer the cycle value to buffers.
3. Enable PPG operation.
4. Generate an activation trigger.
5. Load the cycle and duty values.
6. Counter down count.
7. The down counter equals the duty value.
8. Inverses the PPG pin output level.
9. Counter down count.
10. Counter borrow.
11. Clear the PPG pin output level (return to normal).
12. The operating sequence is now completed

6.8.3.3 Restart Operation

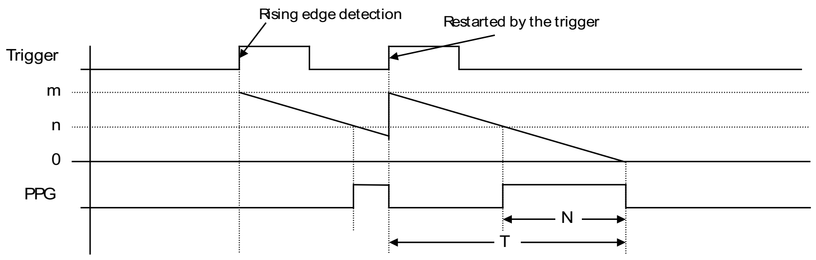
The restart operation is described below:

Restart available in PWM operation:



N = duty, T = cycle

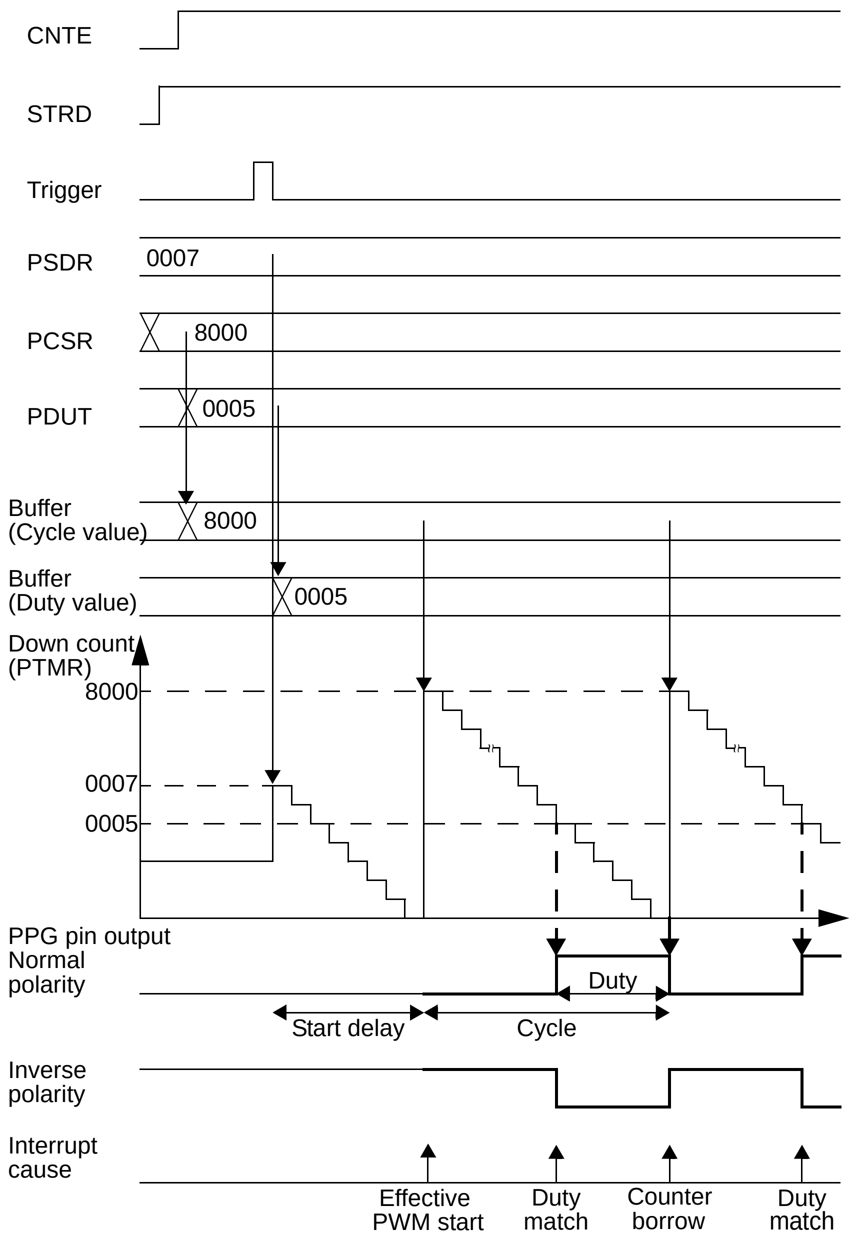
Restart available in one-shot operation:



If a restart is not available, the second and subsequent triggers have no effect in both PWM and one-shot operations. (The second and subsequent triggers following a shutdown of the down counter are functional).

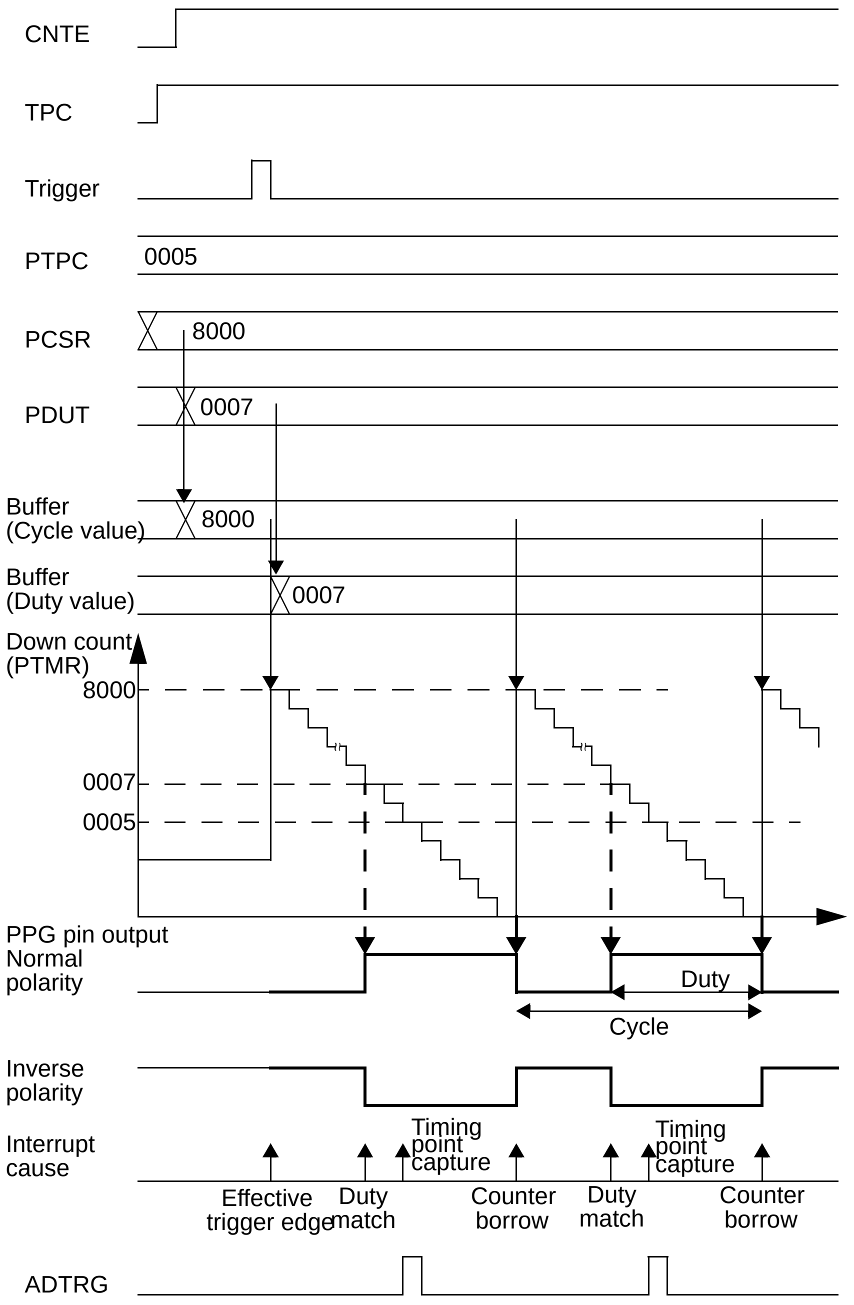
6.8.3.4 Start Delay Mode

In start delay mode, generation of the PWM output is postponed by PSDR PPG count cycles.



6.8.3.5 Timing Point Capture

In this operation mode, a timing point within PWM cycle can be defined to be used as possible trigger for a reload timer.



Timing point capturing is enabled for 4 PPG / PWMs and routed to the trigger input TIN_2 of 4 different Reload Timer (RLT).

PPG0(lower 8bit or 16bit) is routed to RLT13

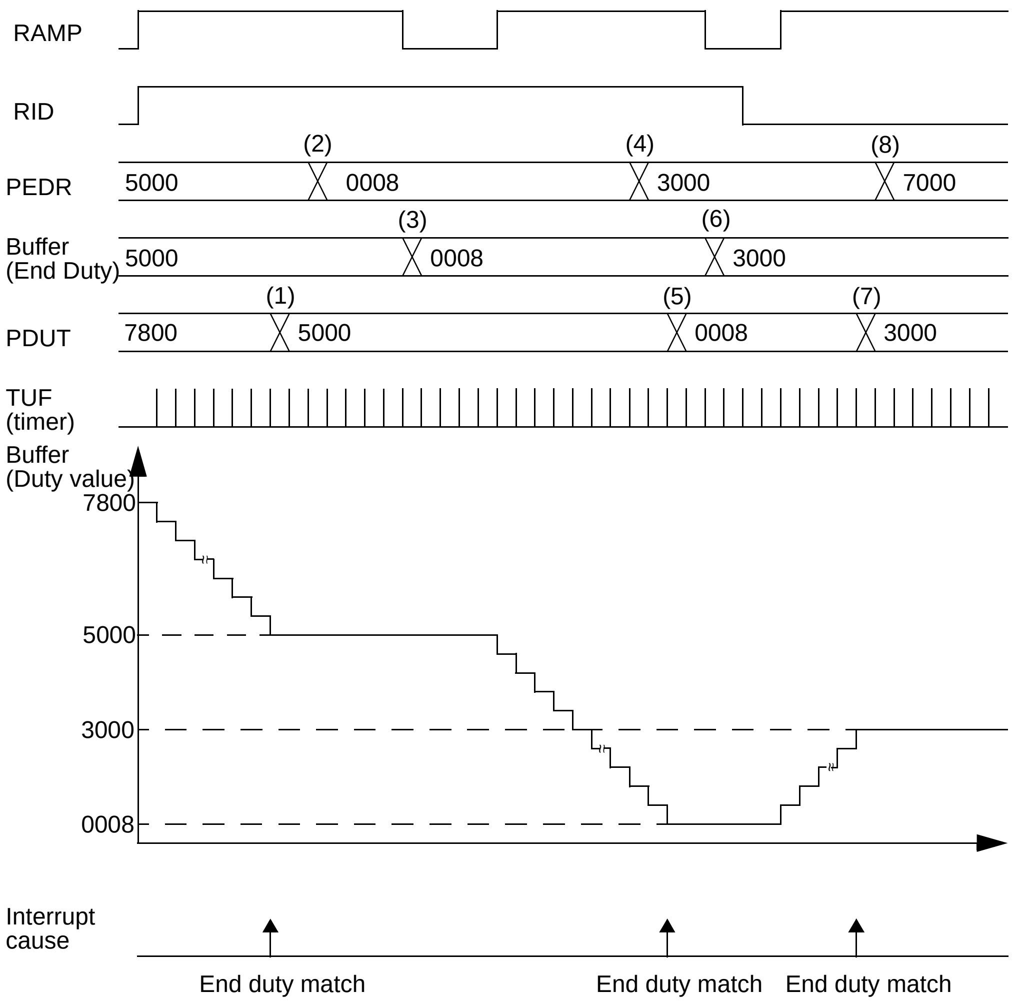
PPG4(lower 8bit or 16bit) is routed to RLT9

PPG8(lower 8bit or 16bit) is routed to RLT5

PPG12(lower 8bit or 16bit) is routed to RLT1

6.8.3.6 Ramp Mode

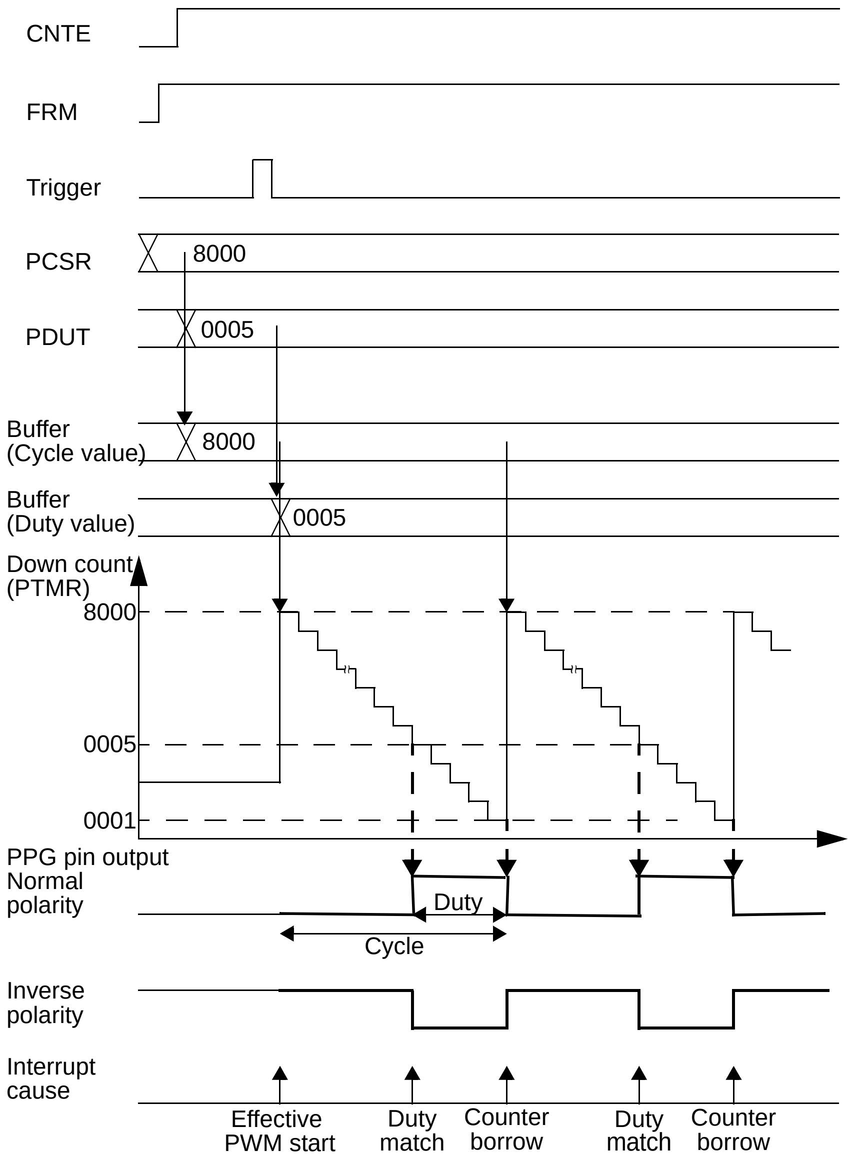
In the ramp mode PWM duty is incremented (PPGn_RMPCFG:RIDH/RIDL=“0") or decremented (PPGn_RMPCFG:RIDH/RIDL=“1") with every selected reload timer underflow pulse. PWM output waveform starts with the duty defined by PDUT register and the duty value is incremented/decremented until the end duty is reached. Reload timer 0..6 can be selected with register GCN3:RTG for ramp mode.



1. After the end duty is reached, PDUT register is updated to the end duty.
2. New PEDR value is set by software.
3. Since the ramp mode is disabled, PEDR value is transferred to the actual end duty.
4. New PEDR value is set by software.
5. End duty is reached, PDUT register is updated to the end duty.
6. Ramp mode is disabled, hence PEDR value is transferred to the actual end duty.
7. End duty is reached, PDUT register is updated to the end duty.
8. New PEDR value is set by software, but it will become active (transferred to end duty buffer register) after the ramp mode is disabled.

6.8.3.7 Full Range Mode

PPG counter borrow happens at PTMR=1. As a consequence, for the setting PDUT=0 the PPG output pin stays “0" all the time and accordingly, for the setting PDUT=PCSR PPG output stays on high level during the whole operation time. Please refer to Register PPGn_EPCN1 for detailed information on how to activate full range mode.



- Equation:
  - Period = Period value (PCSR) x Count clock
  - Duty = Duty value (PDUT) x Count clock
  - Width up to pulse output = {Period value (PCSR) – Duty value (PDUT)} x Count clock.

6.8.3.8 Count Clock Selection

The Count clock can be either the rbus_clk or the output from a Reload timer (GCN4:CKSEL). Reload timer 0..6 can be selected with PPGn_GCN4:RCK. The selected count clock can be divided with PCN:CKS.

6.8.3.9 6.8.3.9 Activation Trigger Selection

The activation trigger can be either a software trigger (SWTRIG:STGR) or an internal trigger (if PCN:EGS = 1).

The internal trigger can be selected with GCN1:TSEL.

TSEL = 0..3 internal software trigger individual for every group (GCTRL:EN)

TSEL = 4,5 trigger from Reload timer (individual for every group, see Table 6-28)

TSEL = 6,7 internal software trigger common for all PPG / PWM (GCNR:CTG)

TSEL = 8 Iris frame interrupt 0 (can be used to realize video synchronous PWM)

TSEL = 9 Reload timer 8

Table 6-28: Reload timer

	RTL input 1 (TSEL = 5)	**RLT 0 (TSEL=4)**
PPG Group A	Reload timer 4	Reload timer 0
PPG Group B	Reload timer 5	Reload timer 1
PPG Group C	Reload timer 6	Reload timer 2
PPG Group D	Reload timer 7	Reload timer 3

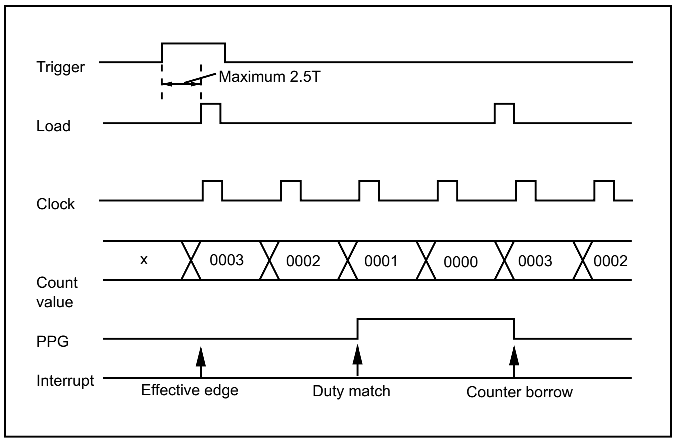
6.8.4 Important Notes

This section describes the cautions to be considered while using the programmable pulse generator:

6.8.4.1 Cautions

This section describes the cautions to be considered while using the programmable pulse generator:

- If the Interrupt Request flag (PPGn_PCN:IRQ.) equals “1” and the Interrupt Request flag is set to “0” at the same timing, the setting of the Interrupt Request flag to “1” overrides the flag clear request.
- The first load has a maximum delay of 2.5T after the activation trigger. (T: Count clock)
  If the down counter is loaded and counts at the same time, the load operation overrides.



- Be sure to write duty value PDUT after cycle PCSR has been initialized and rewritten.
  (Always write in the order of (1) PCSR and (2) PDUT).
  Only the PDUT can be written for rewriting the duty.
- The duty value PDUT must be equal or smaller than the cycle value PCSR. If any larger value has been set, disable the operation of the PPG before replacing the duty value with a smaller value.
- To activate a PPG, it is necessary to set the Timer Operation Enable bits (PPGn_CNTEN:CNTE) to “1” before or concurrently with the activation to enable the PPG operation.
- The values of mode (MDSE), restart enable (RTRG), count clock (CKS[1:0]), trigger input edge (EGS[1:0]), interrupt cause (IRS), internal trigger (TSEL), and output polarity specification (OSEL) may not be changed while the PPG is operating.
  If any of these values has been changed while the PPG was operating, disable the operation of the PPG before reloading the register.
- If Activation Trigger Specification bits (TSEL[3:0]) have been set to any value outside the specified range, disable the operation of the PPG and then write the specified value to let the register return to normal.
- If the Timer Operation Enable bit (PPGn_CNTEN:CNTE) is set to “0” to disable PPGn while it is operating, the PPG stops with its PPG counter being latched and resetting the PPG output level to default value (low level if bit OSEL=’0’ or high level if OSEL.=’1’).
  To activate PPG operation again the Timer Operation Enable bit (PPGn_CNTEN:CNTE) is set to “1” before or concurrently with providing one of the PPG triggers to enable the PPG. After that PPG counter is loaded with PPG cycle value and PPG pulses are generated.

One-shot pulse can be generated only in 16-bit operation mode and without start delay feature.

6.8.5 Programmable Pulse Generators Control Register Overview

A group of 4 PPGs has a set of group control registers.

All 16 PPGs have one common control register unit.

Table 6-29: Registers Overview

Base Address(es)	**Instance no 0: BASEADDR0="00089000"** **Instance no 1: BASEADDR1="0008B000"** **Instance no 2: BASEADDR2="0008D000"** **Instance no 3: BASEADDR3="0008F000"**	
Absolute Address	**Register Name**	**Register Description**
BASEADDRx + 0x0000	PPGGRPp_GCTRL	PPG Group Control Register
BASEADDRx + 0x0001	Reserved	Do not modify

Table 6-30: Registers Overview

Base Address(es)	**Instance no 0: BASEADDR="00090000"**	
Absolute Address	**Register Name**	**Register Description**
BASEADDR + 0x0000	PPGGLCg_GCNR	PPG General Control Register
BASEADDR + 0x0001	Reserved	Do not modify

6.8.6 Programmable Pulse Generator Core Register Overview

Every individual PPG has one set of core unit register.

Table 6-31: Registers Overview

Base Address(es)	**Instance no 0: BASEADDR0="00088000"** **Instance no 1: BASEADDR1="00088400"** **Instance no 2: BASEADDR2="00088800"** **Instance no 3: BASEADDR3="00088C00"** **Instance no 4: BASEADDR4="0008A000"** **Instance no 5: BASEADDR5="0008A400"** **Instance no 6: BASEADDR6="0008A800"** **Instance no 7: BASEADDR7="0008AC00"** **Instance no 8: BASEADDR8="0008C000"** **Instance no 9: BASEADDR9="0008C400"** **Instance no 10: BASEADDR10="0008C800"** **Instance no 11: BASEADDR11="0008CC00"** **Instance no 12: BASEADDR12="0008E000"** **Instance no 13: BASEADDR13="0008E400"** **Instance no 14: BASEADDR14="0008E800"** **Instance no 15: BASEADDR15="0008EC00"**	
Absolute Address	**Register Name**	**Register Description**
BASEADDRx + 0x0000	PPGn_PCN	PPG Control Status Register
BASEADDRx + 0x0002	PPGn_IRQCLR	Interrupt Flag Clear Register
BASEADDRx + 0x0003	PPGn_SWTRIG	Software Trigger Activation Register
BASEADDRx + 0x0004	PPGn_OE	Output Enable Register
BASEADDRx + 0x0005	PPGn_CNTEN	Timer Enable Operation Register
BASEADDRx + 0x0006	PPGn_OPTMSK	Output Mask and Polarity Selection Register
BASEADDRx + 0x0007	PPGn_RMPCFG	Ramp Configuration Register
BASEADDRx + 0x0008	PPGn_STRD	Start Delay Mode Register
BASEADDRx + 0x0009	PPGn_TRIGCLR	PPG Trigger Clear Flag Register
BASEADDRx + 0x000A	PPGn_EPCN1	Extended PPG Control Status Register 1
BASEADDRx + 0x000C	PPGn_EPCN2	Extended PPG Control Status Register 2
BASEADDRx + 0x000E	PPGn_GCN1	General Control Register 1
BASEADDRx + 0x000F	PPGn_GCN3	General Control Register 3
BASEADDRx + 0x0010	PPGn_GCN4	General Control Register 4
BASEADDRx + 0x0011	PPGn_GCN5	General Control Register 5
BASEADDRx + 0x0012	PPGn_PCSR	PPG Cycle Setting Register
BASEADDRx + 0x0014	PPGn_PDUT	PPG Duty Setting Register
BASEADDRx + 0x0016	PPGn_PTMR	PPG Timer Register
BASEADDRx + 0x0018	PPGn_PSDR	PPG Start Delay Register
BASEADDRx + 0x001A	PPGn_PTPC	PPG Timing Point Capture Register
BASEADDRx + 0x001C	PPGn_PEDR	PPG End Duty Register
BASEADDRx + 0x001E	PPGn_DMACFG	PPG DMA Configuration Register
BASEADDRx + 0x001F	Reserved	Do not modify

6.9 Reload Timer (RLT)

A reload timer consists of a 32-bit down-counter, a 32-bit reload register, one or two internal trigger input and one (device internal) trigger output, and control registers. In total, 16 reload timers are implemented in MB88F33x ‘Indigo2(-x)’ and are mostly used as triggers for synchronization purposes (e.g. ConfigFIFO, ADC).

6.9.1 Features of the Reload Timer

- Two internal clock/event sources.
- Trigger signal programmable as rising/falling edge or both.
- Gated count function.
- One-shot or reload counter mode.
- Prescaler with 6 different settings for the internal clock and 2 settings for the external clock.
- Several Reload Timers can be cascaded to form a longer Reload Timer.
- Generate DMA requests for RLT0 and RLT1.
- Generate interrupt in case of underflow.

6.9.2 DMA and Interrupts

RLT0 and RLT1 can generate DMA requests which can be used to start DMA transfers and all RLT units can generate an interrupt in the event of a count underflow.

6.9.3 Block Diagram

Figure 6-82 shows a detailed block diagram of 32-bit reload timer.

NOTE The suffix "n" denotes the instance of the Reload Timer.

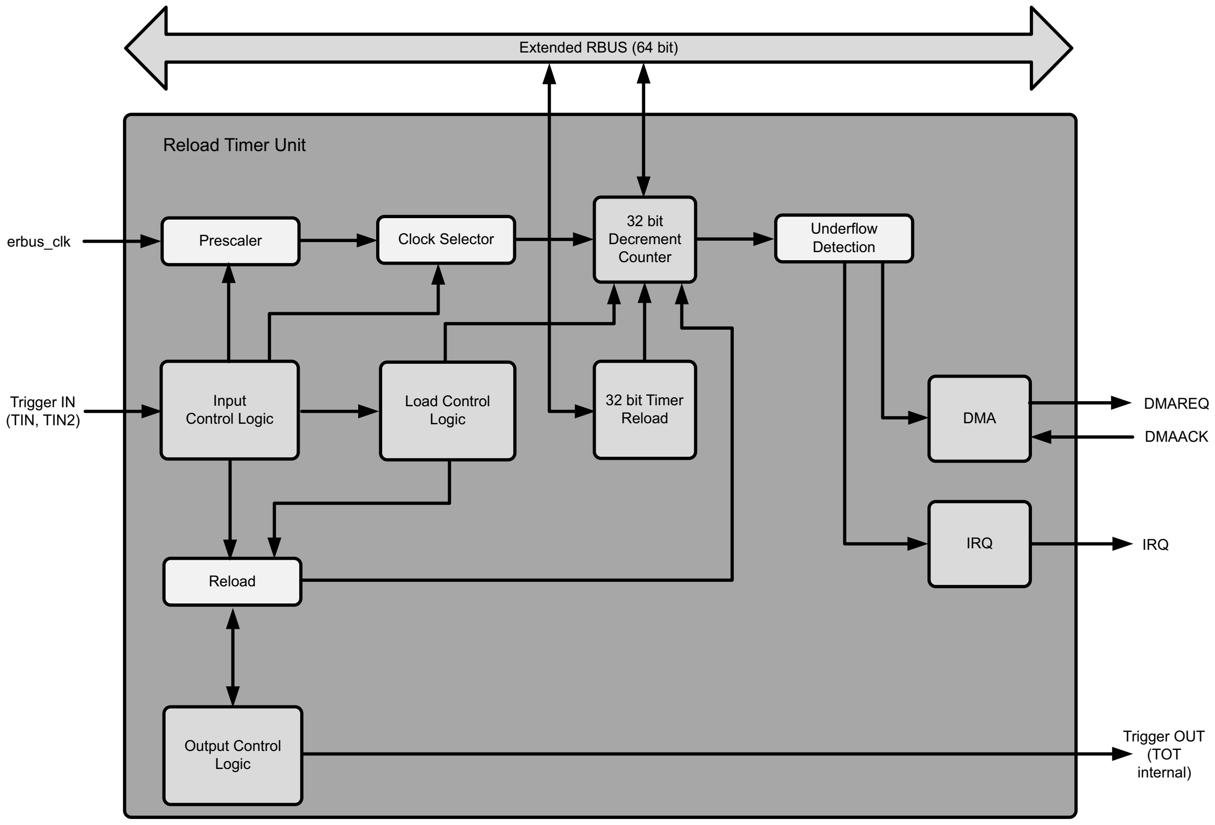


Figure 6-82: 32-bit reload timer block diagram

6.9.4 Operation of the 32-bit Reload Timer

A Reload Timer unit can be run in one of two modes: internal clock mode or event counter mode.

In internal clock mode, the peripheral clock erbus_clk is selected (different clock signal divider settings are possible) as the clock source for operating the Reload Timer. The trigger input TINs TIN, TIN2 (device internal signal) can be selected as either a trigger input or gate input via a register setting.

In event counter mode, TINn is used as an external event input signal. Every active edge detected on this input (configurable: rising, falling or both edges) decrements the counter.

When TMCSR1n:CSL0 = 1, then only every *second* event is counted.

6.9.4.1 Internal Clock Operation of 32-bit Reload Timer

Writing "1" to both the TMCSR3n:CNTE and TMCSR2n:TRG bits enables the unit and simultaneously starts counting. Use of the TMCSR2n:TRG bit as a trigger input is always possible when the timer has been enabled (TMCSR3n:CNTE = "1"), regardless of the operation mode.

Figure 6-83 illustrates the activation and operation of the counter.

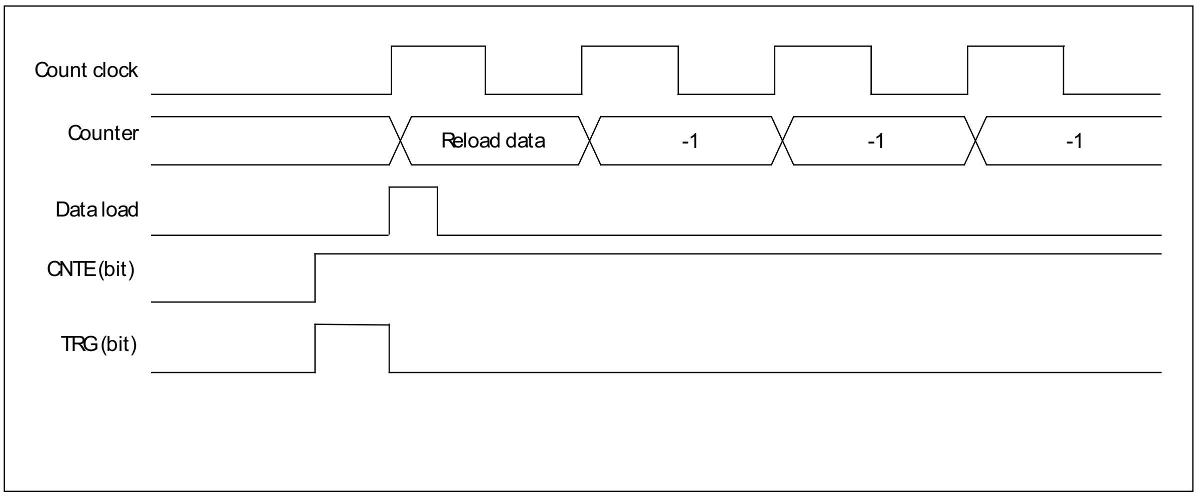


Figure 6-83: Activation and operation of 32-bit Reload Timer counter

6.9.4.2 Input Functions of 32-bit Reload Timer (in Internal Clock Mode)

The TINn input can be used as either a trigger input or a gate input when an internal clock is selected as the clock source. When used as a trigger input, an active edge causes the timer to load the reload register contents and resets the internal prescaler. Then counting starts.

Figure 6-84 on page 154 shows the operation of the trigger input.

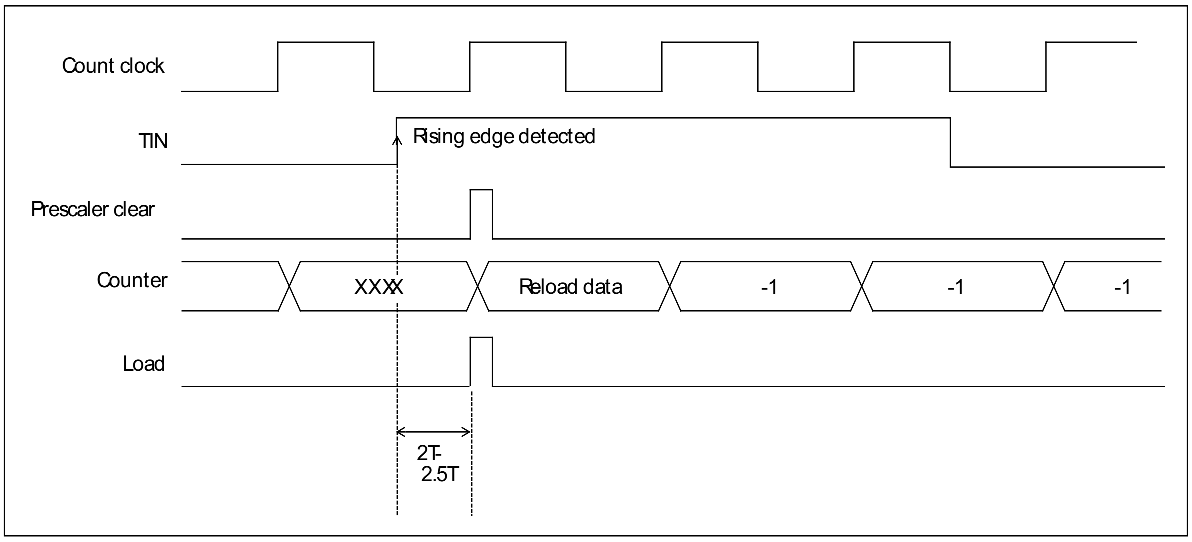


Figure 6-84: Trigger input operation of 32-bit Reload Timer

When used as a gate input, the counter only counts while the active level specified by the TMCSR1n:MOD0 bit is supplied to the TINn input. In this case, the count clock continues to operate until it is stopped. The software trigger can be used in gate mode, regardless of the gate level. Input a pulse width of at least 2T to the TINn. Figure 6-85: "Gate input operation of 32-bit Reload Timer" shows the operation of gate input.

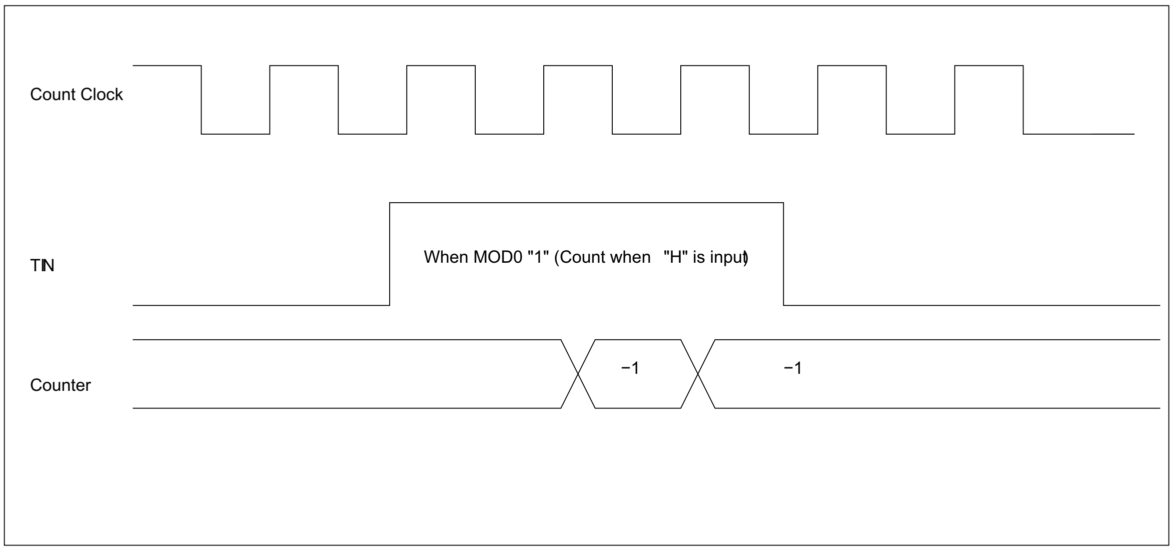


Figure 6-85: Gate input operation of 32-bit Reload Timer

6.9.4.3 Trigger Input/Output

The IRQ outputs of all Reload timers are connected to the interrupt controller. The trigger outputs (TOT) of all Reload timers can be used for the Config FIFO. Some other RLT outputs are wired to internal blocks (see Table 6-32 .

Table 6-32: RLT Connections

	Trigger Inputs		Outputs		
Name	**TIN**	**TIN_2 (Input 2)**	**TOT**	**UFFlag**	**DMA**
RLT0	RLT11TOT	n/c	PPG A RLT_TRG[0]	PPG UFSET[0]	DMA_8
RLT1	RLT12TOT	Timing Point Capture PPG12	PPG B RLT_TRG[0]	PPG UFSET[1]	DMA_9
RLT2	RLT13TOT	EXT_IRQ[3]	PPG C RLT_TRG[0]	PPG UFSET[2]	n/c
RLT3	RLT10TOT	n/c	PPG D RLT_TRG[0]	PPG UFSET[3]	n/c
RLT4	RLT11TOT	CMDSEQ WD	PPG A RLT_TRG[1]	PPG UFSET[4]	n/c
RLT5	RLT12TOT	Timing Point Capture PPG8	PPG B RLT_TRG[1]	PPG UFSET[5]	n/c
RLT6	RLT13TOT	EXT_IRQ[2]	PPG C RLT_TRG[1]	PPG UFSET[6]	n/c
RLT7	RLT10TOT	n/c	PPG D RLT_TRG[1]	n/c	n/c
RLT8	RLT11TOT	SYS Watchdog	PPG ETRG[1]	n/c	n/c
RLT9	RLT12TOT	Timing Point Capture PPG4	ADC TIMI	n/c	n/c
RLT10	RLT13TOT	EXT_IRQ[1]	RLT3 TIN, RLT7 TIN, RLT11 TIN, RLT15 TIN	n/c	n/c
RLT11	RLT10TOT	n/c	RLT0 TIN, RLT4 TIN, RLT8 TIN, RLT12 TIN	n/c	n/c
RLT12	RLT11TOT	PANIC_SWITCH	RLT1 TIN, RLT5 TIN, RLT9 TIN, RLT13 TIN	n/c	n/c
RLT13	RLT12TOT	Timing Point Capture PPG0	RLT2 TIN, RLT6 TIN, RLT10 TIN, RLT14 TIN	n/c	n/c
RLT14	RLT13TOT	EXT_IRQ[0]	n/c	n/c	n/c
RLT15	RLT10TOT	n/c	ALIVE SENDER	n/c	n/c

6.9.5 External Event Counter

When external event count mode is selected, TINn is used as an external event input. The counter counts on the active edge specified in the TMCSR1n.

6.9.6 Underflow Operation of 32-bit Reload Timer

A reload timer underflow (UF) is defined as the time when the counter value changes from 00000000_H to $FFFFFFFF_H$ or when a reload occurs (RLTn-TMCSR:RELD="1"). Therefore, an underflow occurs after a (reload register setting + 1) count.

6.9.6.1 Underflow Operation of 32-bit Reload Timer

If the RLTn-TMCSR:RELD bit is "1" and an underflow occurs, the contents of the reload register are loaded into the counter and counting continues.

If the RLTn-TMCSR:RELD bit is "0", counting stops when the counter reaches $FFFFFFFF_H$.

The TMCSR2n:UF bit is set when the underflow occurs. If the RLTn-TMCSR:INTE bit is "1" at this time, an interrupt request is generated.

Figure 6-86 shows operation when an underflow occurs with various values of RLTn-TMCSR:RELD. Figure 6-87: "Clearing of Underflow bit" shows a clear operation for the underflow flag.

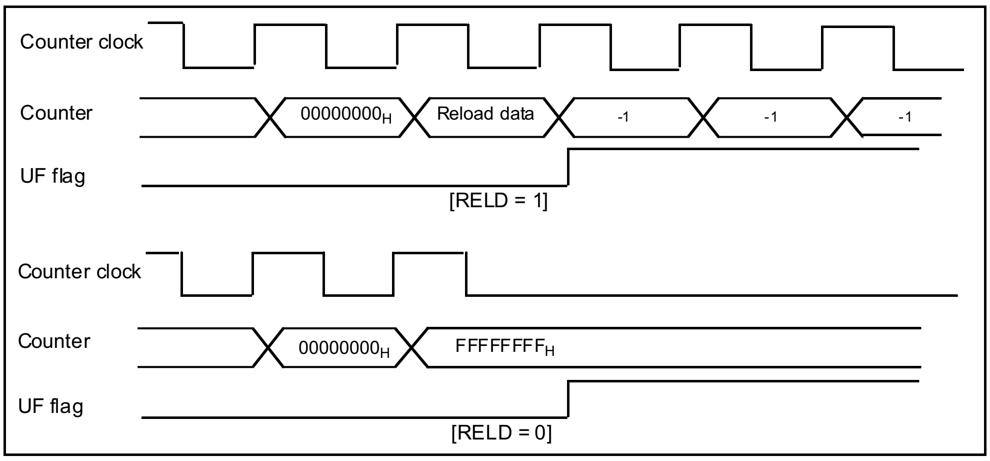


Figure 6-86: Underflow operation of 32-bit reload timer

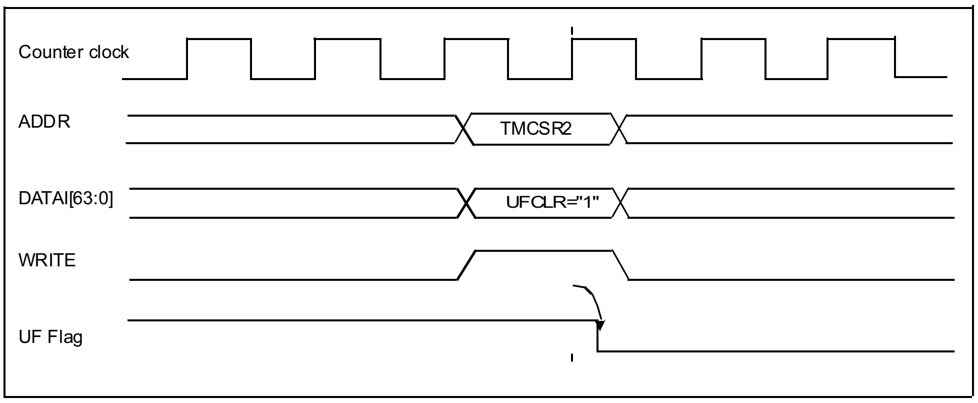


Figure 6-87: Clearing of Underflow bit

6.9.7 Output Functions of 32-bit Reload Timer

In reload mode, the TOTn output (device internal signal) toggles the output (inverts the signal at each underflow). In one-shot mode, the TOTn output is used as a pulse output that shows the configured level while counting is in progress.

6.9.7.1 Output Signal Functions of 32-bit Reload Timer

The RLTn-TMCSR:OUTL bit sets the output polarity.

If RLTn-TMCSR:OUTL = "0", the initial value or toggle output is "L" and the one-shot pulse output is "H" while the count is in progress.

If RLTn-TMCSR:OUTL = "1", the output waveforms are inverse to each other.

Figure 6-88: "Output signal function of 32-bit Reload Timer in reload mode" and Figure 6-89: "Output signal function of 32-bit Reload Timer in one-shot mode" show the output signal functions.

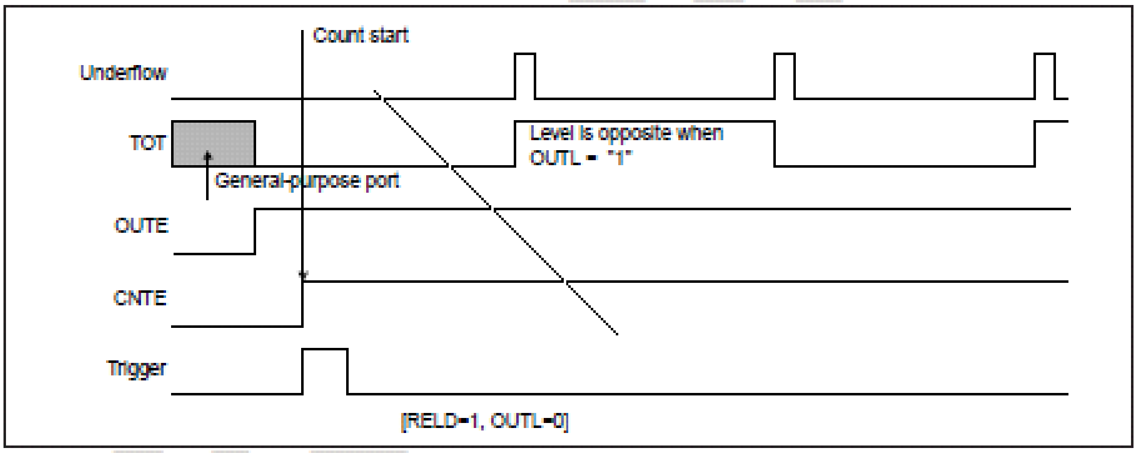


Figure 6-88: Output signal function of 32-bit Reload Timer in reload mode

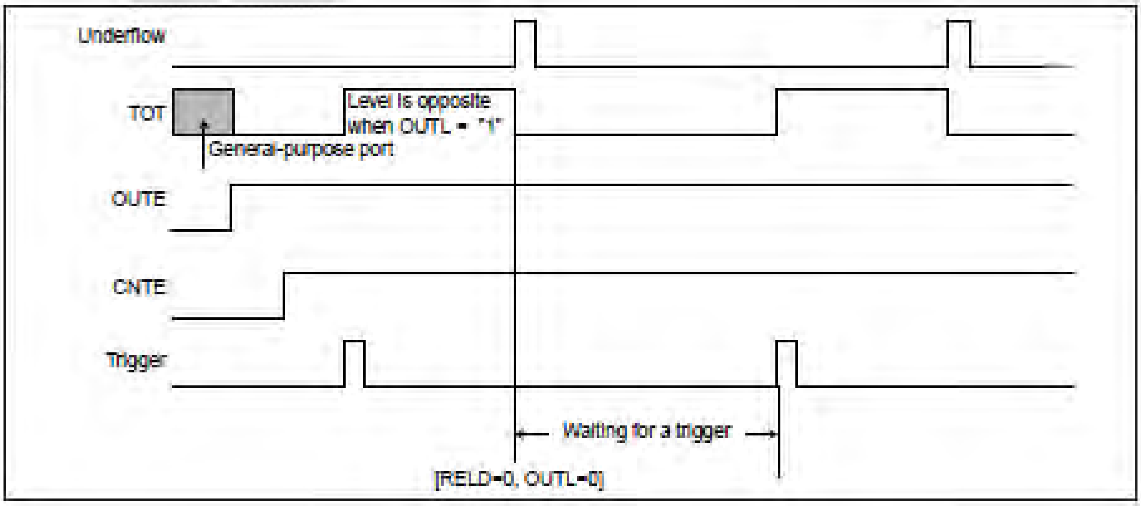


Figure 6-89: Output signal function of 32-bit Reload Timer in one-shot mode

6.9.8 Counter Operation State

The counter state is determined by the Debug Signal, RLTn-TMCSR:DBGE bit, TMCSR3n:CNTE bit in the control status register and the internal WAIT signal.

The following states exist for the reload timer:

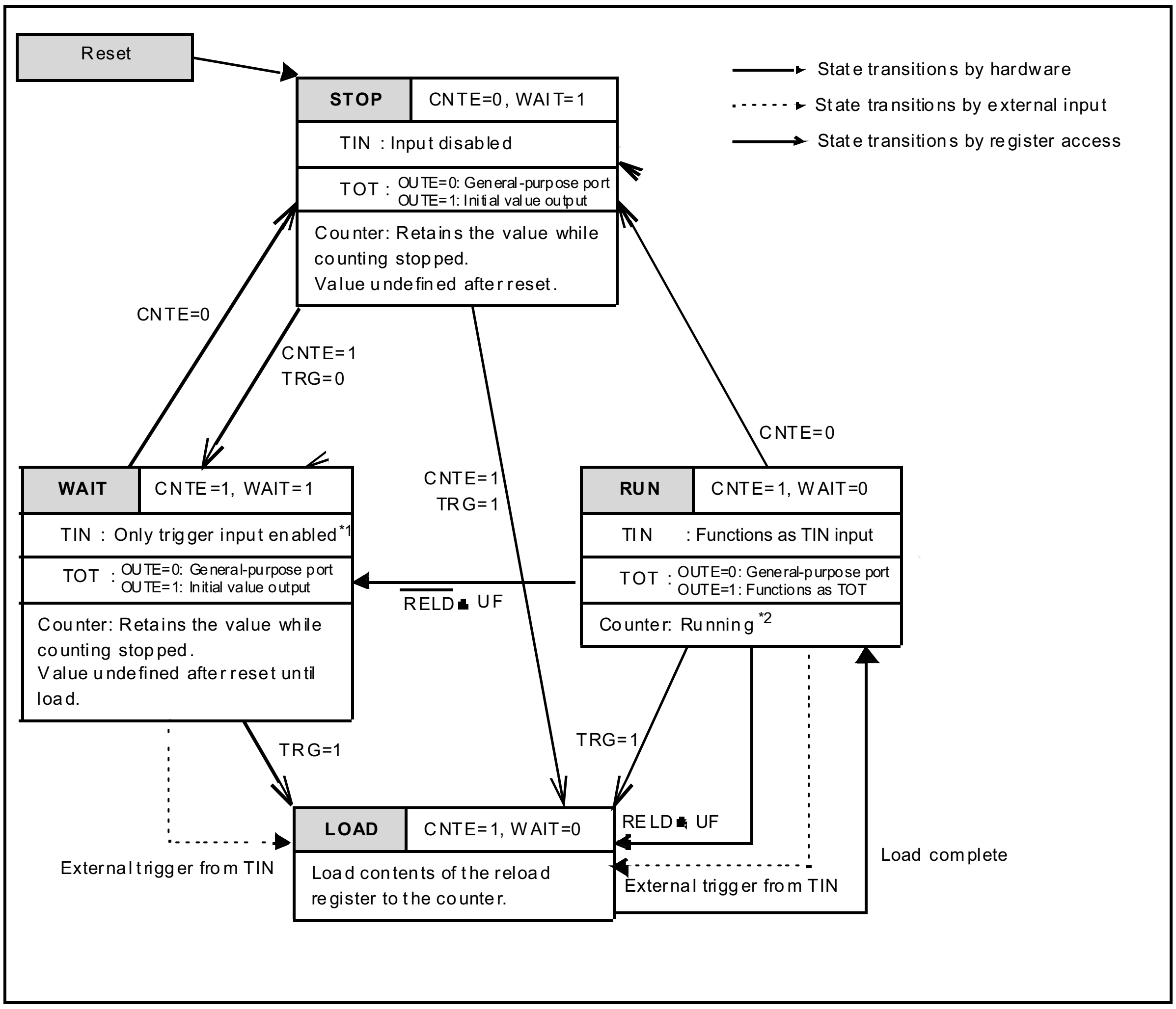
STOP State: (RLTn-TMCSR:DBGE & DEBUG) = "0", TMCSR3n:CNTE = "0" and WAIT = "1"

WAIT State: (RLTn-TMCSR:DBGE & DEBUG) = "0", TMCSR3n:CNTE = "1" and WAIT = "1"

RUN State: (RLTn-TMCSR:DBGE & DEBUG) = "0", TMCSR3n:CNTE = "1" and WAIT = "0"

6.9.9 Counter Operation States

Figure 6-90: "Counter state transitions" shows the transitions between each state.


Reset
State transitions by hardware
State transitions by external input
State transitions by register access
STOP
CNTE=0, WAIT=1
TIN : Input disabled
TOT : OUTE=0: General-purpose port OUTE=1: Initial value output
Counter: Retains the value while counting stopped. Value undefined after reset.
CNTE=0
CNTE=1 TRG=0
CNTE=1 TRG=1
CNTE=0
WAIT
CNTE=1, WAIT=1
TIN : Only trigger input enabled*1
TOT : OUTE=0: General-purpose port OUTE=1: Initial value output
Counter: Retains the value while counting stopped. Value undefined after reset until load.
RUN
CNTE=1, WAIT=0
TIN : Functions as TIN input
TOT : OUTE=0: General-purpose port OUTE=1: Functions as TOT
Counter: Running *2
RELD ■ UF
TRG=1
TRG=1
LOAD
CNTE=1, WAIT=0
Load contents of the reload register to the counter.
External trigger from TIN
RELD ■ UF
External trigger from TIN
Load complete

*1 : Before using TIN input, configure corresponding port pin control bits correctly

*2: In ’Gate input mode’. Counting can be influenced by TIN.

Figure 6-90: Counter state transitions

6.9.10 DMA Operation

DMA support is determined by the RLTn-DMACFG:EN_DMA_UF bit. Setting this bit enables the generation of DMA requests. The assertion of the DMA_REQ_ACK signal acknowledges a request and causes the DMA_REQ signal to be de-asserted.MB88F33x ‘Indigo2(-x)’ supports DMA for RLT0 and RLT1

6.9.10.1 Enabling DMA support

Writing "1" to RLTn-DMACFG:EN_DMA_UF enables the generation of DMA requests when an underflow occurs (when the RLTn-TMCSR:UF bit is set). Writing "0" to RLTn-DMACFG:EN_DMA_UF disables DMA request generation even if the TMCSR2n:UF bit is set.

When DMA_REQ_ACK is asserted, the DMA_REQ signal is de-asserted by acknowledging the DMA Request.

Figure 6-91 shows the consequence of an asserted DMA_REQ_ACK signal.

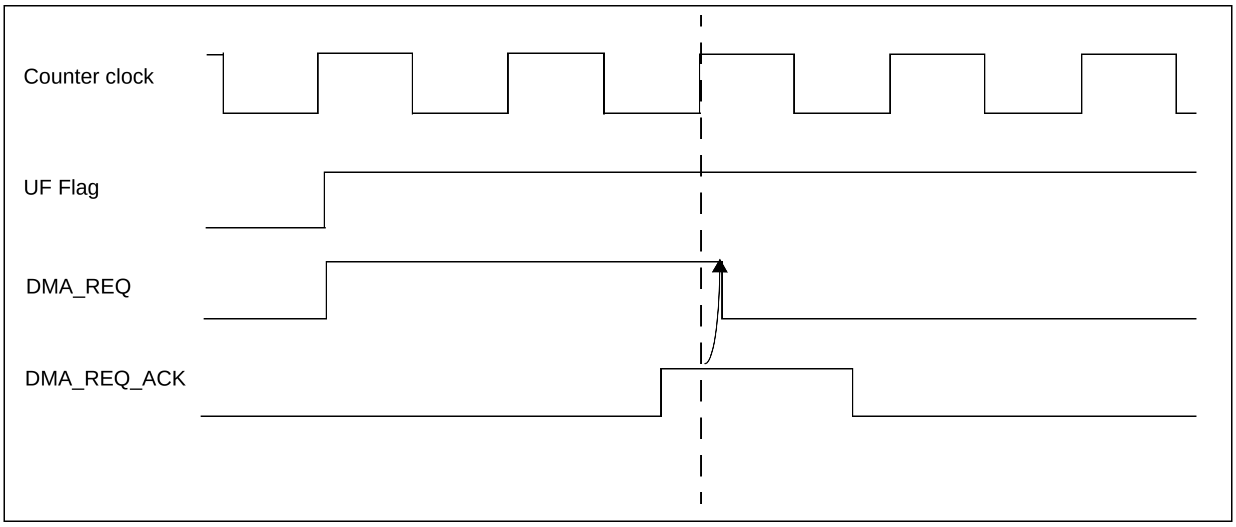


Figure 6-91: De-asserted DMA_REQ signal

6.9.11 Reload Timer Register Overview

Table 6-33: Registers Overview

Base Address(es)	**Instance no 0: BASEADDR0="000A0000"** **Instance no 1: BASEADDR1="000A0800"** **Instance no 2: BASEADDR2="000A1000"** **Instance no 3: BASEADDR3="000A1800"** **Instance no 4: BASEADDR4="000A2000"** **Instance no 5: BASEADDR5="000A2800"** **Instance no 6: BASEADDR6="000A3000"** **Instance no 7: BASEADDR7="000A3800"** **Instance no 8: BASEADDR8="000A4000"** **Instance no 9: BASEADDR9="000A4800"** **Instance no 10: BASEADDR10="000A5000"** **Instance no 11: BASEADDR11="000A5800"** **Instance no 12: BASEADDR12="000A6000"** **Instance no 13: BASEADDR13="000A6800"** **Instance no 14: BASEADDR14="000A7000"** **Instance no 15: BASEADDR15="000A7800"**	
Absolute Address	**Register Name**	**Register Description**
BASEADDRx + 0x0000	RLTn_DMACFG	DMA Configuration Register
BASEADDRx + 0x0004	Reserved	Do not modify
BASEADDRx + 0x0008	RLTn_TMCSR	Timer Control Status Register
BASEADDRx + 0x000C	Reserved	Do not modify
BASEADDRx + 0x0010	RLTn_TMRLR	Timer Reload Register
BASEADDRx + 0x0014	RLTn_TMR	Timer Register

6.9.12 Reload Timer Additional Register Information

6.9.12.1 DMACFGn

This register is only functional for RLT0 and RLT1.

6.9.12.2 Timer Control Status Register (RLTn_TMCSR)

Table 6-34: MOD2/1/0 bit settings for internal clock mode (CSL1/2 = “00_B”, “01_B”, or “10_B”)

MOD2	MOD1	MOD0	Input Function	Active Edge or Level
0	0	0	Trigger disabled	-
0	0	1	Trigger input	Rising edge
0	1	0		Falling edge
0	1	1		Both edges
1	x	0	Gate input	"L" level
1	x	1		"H" level

Table 6-35: MOD2/1/0 bit settings for event counter mode (CSL1/2 = “11_B”)

MOD2	MOD1	MOD0	Input Function	Active Edge or Level
x	0	0	-	-
	0	1	Event input	Rising edge
	1	0		Falling edge
	1	1		Both edges
Bits marked as x in the table can be set to any value.				

Table 6-36: Clock sources for CSL0/1/2 bit settings

CSL2	CSL1	CSL0	Clock Source (Time for peripheral erbus_clk)
0	0	0	erbus_clk
0	0	1	erbus_clk/2
0	1	0	erbus_clk/4
0	1	1	erbus_clk/8
1	0	0	erbus_clk/16
1	0	1	erbus_clk/32
1	1	0	External event count mode
1	1	1	External event count mode/ 2

Table 6-37: OUTE, OUTL, and RELD settings

OUTE	OUTL	RELD	Output Waveform
0	x	x	Timer output disabled
1	0	0	Output an "H" level pulse during counting.
1	1	0	Output an "L" level pulse during counting.
1	0	1	Toggle output. Starts with "L" level output. Changes level on timer reload.
1	1	1	Toggle output. Starts with "H" level output. Changes level on timer reload.

6.10 General Purposes Input Output (GPIO)

All functional pins can be configured to be used as a GPIO. MB88F33x ‘Indigo2(-x)’ has 110 pins which can be used for this purpose. These pins are controlled by a GPIO module with 2 port instances with 64 channels each.

6.10.1 Features of the GPIO Module

- GPIO Module accommodates two GPIO Ports (Port-0 and Port-1).
- Each GPIO Port accommodates 64 GPIO Channels.
- Each GPIO Channel in turn maps to corresponding Port Pin.
- GPIO Module accommodates eight 64-bit registers (DDR, DDSR, DDCR, PODR, POSR, POCR, PIDR, PPER) associated to single GPIO Port.
- Each register is repeated 2 times to support 2 GPIO Ports.
- Two GPIO Ports supports 128 GPIO Channels and in turn 128 Port Pins.
- Each GPIO Module register can be accessed by 8, 16, or 32-bit bus accesses.
- All GPIO Module registers are readable.
- All GPIO Module registers (except PPERn and PIDRn) are bit-wise write protected.

6.10.2 GPIO Functional Description

6.10.2.1 Data Input And Output

For every GPIO pin, a data direction register (DDR) a port data input register (PIDR) and port output data register (PODR) exist.

Depending on the setting in the data direction register, the pin operates either as an output or as an input. When in output mode, the output level can be set with the port output data register. In input mode, the pin value can be read with the PIDR register.

Every individual bit in the data direction register and in the port output data register can be set or reset by a dedicated set or clear register (for DDR the set register is DDSR and the clear register is DDCR. For PODR the register POCR and POSR are used).

6.10.2.2 Bit-wise Write Protection

All GPIO Module registers (except PPERn and PIDRn) are bit-wise write protected. Figure 6-92 describes bit-wise write protection for single bit of a write protected register.

- Each bit GPIO Module write protected register can be written only if corresponding bit in the Port PPU Enable Register (PPERn) is set to “1“.
- Individual write strobe for each bit is generated with logical AND operation of register write strobe with corresponding PPERn bit (as described above and shown in Figure 6-92: "Bit-wise write protection logic").
- The PPERn register is a write once register. It can be written only one time after reset.

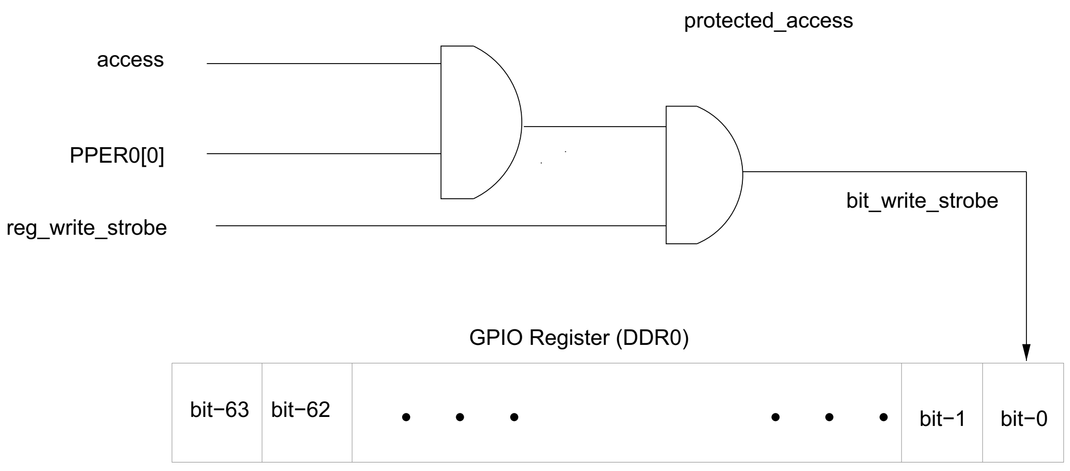


Figure 6-92: Bit-wise write protection logic

6.10.3 Software Interface

6.10.3.1 GPIO Module Register Set

The GPIO Module contains various registers to configure its operation and to monitor its status.

6.10.3.2 Allocation of Control and Status Register (CSRs)

The GPIO Module uses 4kBytes of address space for mapping its own Control and Status. All registers of GPIO Module are shown in the related Register Description chapter. The registers can be accessed with 8-bit, 16-bit, 32-bit access.

6.10.3.3 Numbering of GPIO Channels

Table 6-38: "GPIO Channel Numbering" describes the association between GPIO Ports, GPIO Channels, GPIO Module registers and Port Pins. To illustrate the association only one register (Data Direction Register, DDRn) is shown here. All other registers are similarly associated with the same scheme.

Table 6-38: GPIO Channel Numbering

GPIO Port Number	GPIO Channel Numbers	Associated GPIO Module Register	Port Pin Number
GPIO Port-0	GPIO Channels 0 to 63	DDR0 Bits [0 to 63]	Port Pins 0 to 63
GPIO Port-1	GPIO Channels 64 to 109	DDR1 Bits [0 to 45]	Port Pins 64 to 109

6.10.4 General Purposes IO Register Overview

Table 6-39: Registers Overview

Base Address(es)	**Instance no 0: BASEADDR="000A8000"**	
Absolute Address	**Register Name**	**Register Description**
BASEADDR + 0x0000	POSR0 Low	Port Output Set Register 0
BASEADDR + 0x0004	POSR0 High	Port Output Set Register 0
BASEADDR + 0x0008	POCR0 Low	Port Output Clear Register 0
BASEADDR + 0x000C	POCR0 High	Port Output Clear Register 0
BASEADDR + 0x0010	DDSR0 Low	Data Direction Set Register 0
BASEADDR + 0x0014	DDSR0 High	Data Direction Set Register 0
BASEADDR + 0x0018	DDCR0 Low	Data Direction Clear Register 0
BASEADDR + 0x001C	DDCR0 High	Data Direction Clear Register 0
BASEADDR + 0x0020	POSR1 Low	Port Output Set Register 1
BASEADDR + 0x0024	POSR1 High	Port Output Set Register 1
BASEADDR + 0x0028	POCR1 Low	Port Output Clear Register 1
BASEADDR + 0x002C	POCR1 High	Port Output Clear Register 1
BASEADDR + 0x0030	DDSR1 Low	Data Direction Set Register 1
BASEADDR + 0x0034	DDSR1 High	Data Direction Set Register 1
BASEADDR + 0x0038	DDCR1 Low	Data Direction Clear Register 1
BASEADDR + 0x003C	DDCR1 High	Data Direction Clear Register 1
BASEADDR + 0x0200	PODR0 Low	Port Output Data Register 0
BASEADDR + 0x0204	PODR0 High	Port Output Data Register 0
BASEADDR + 0x0208	DDR0 Low	Data Direction Register 0
BASEADDR + 0x020C	DDR0 High	Data Direction Register 0
BASEADDR + 0x0210	PODR1 Low	Port Output Data Register 1
BASEADDR + 0x0214	PODR1 High	Port Output Data Register 1
BASEADDR + 0x0218	DDR1 Low	Data Direction Register 1
BASEADDR + 0x021C	DDR1 High	Data Direction Register 1
BASEADDR + 0x0300	PIDR0 Low	Port Input Data Register 0
BASEADDR + 0x0304	PIDR0 High	Port Input Data Register 0
BASEADDR + 0x0308	PIDR1 Low	Port Input Data Register 1
BASEADDR + 0x030C	PIDR1 High	Port Input Data Register 1
BASEADDR + 0x0380	PPER0 Low	Port Enable Register 0
BASEADDR + 0x0384	PPER0 High	Port Enable Register 0
BASEADDR + 0x0388	PPER1 Low	Port Enable Register 1
BASEADDR + 0x038C	PPER1 High	Port Enable Register 1

6.10.5 Additional Register Information

6.10.5.0.1 External Interrupt Level Register (ELVR0n)

ELVR0n provides interrupt request level for 0 to 7 external interrupt sources.

Table 6-40: Interrupt Request Detection Factor for External Interrupts

ELVR:LC	ELVR:LB	ELVR:LA	Interrupt Request Detection Factor
0	0	0	L level pin input
0	0	1	H level pin input
0	1	0	Rising edge pin input
0	1	1	Falling edge pin input
1	x	x	Any edge i.e (rising or falling edge)
x in the table is don’t care			

6.11 External Interrupt Input

The External Interrupt detects a signal that is input via an external interrupt pin and generates an interrupt request. MB88F33x ‘Indigo2(-x)’ supports 8 external interrupt input channels.

6.11.1 Features of the External Interrupt Input

- For an external interrupt request, five levels are available: "‘H"’, "‘L"’, rising edge, falling edge and any edge (rising or falling) are available.
- Supports noise filter (< 50ns filtered, > 250ns not filtered).
- Bypassable noise filters.
- Support for DMA for interrupt 0 and 1.

6.11.2 Block Diagram of External Interrupts System

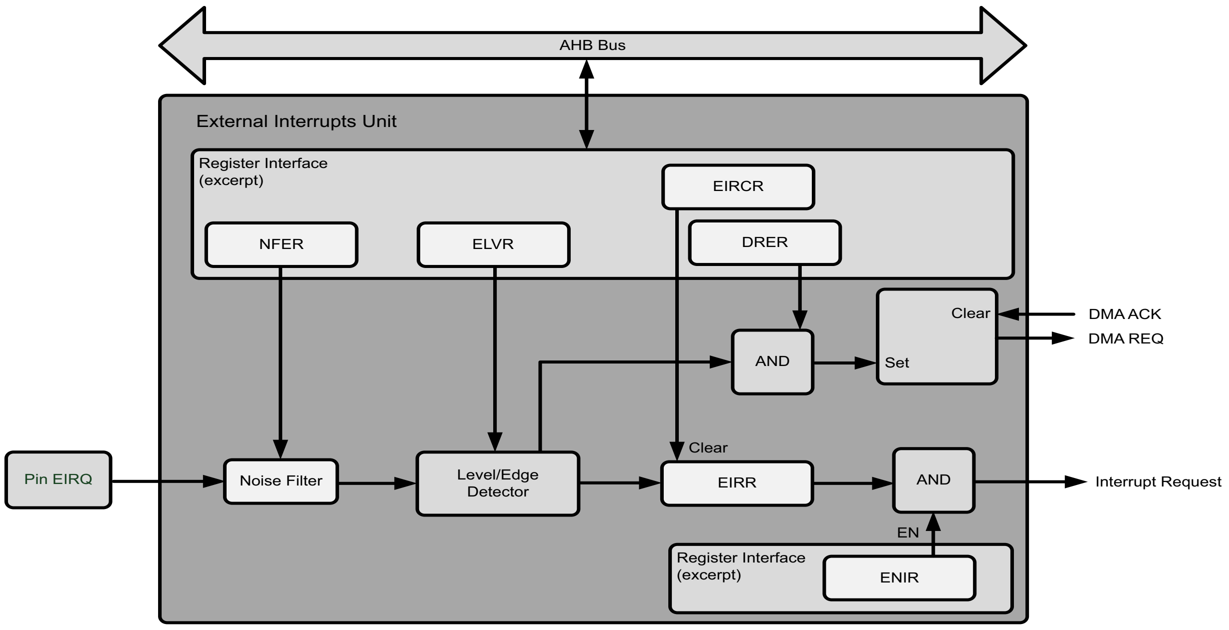


Figure 6-93: Block diagram of external interrupts System

6.11.3 Notes on using the External Interrupt functions

The following points must be considered to use the External Interrupt function.

- Conditions on the behavior of external circuit for use of DMA
- Clearing interrupt flag
- External interrupt request level
- Noise Filter

6.11.3.1 Conditions on the Behaviour of External Circuit for Use of DMA

- An external circuit which uses DMA must be able to inactivate the request signal when the requested DMA transfer is performed. DMACK has higher priority over DMAREQ.

6.11.3.2 Clearing Interrupt Flag

- When it is used as an external interrupt the interrupt flag EIRR:ER must be cleared within the interrupt service routine. Otherwise interrupt is not triggered again after the completion of the first interrupt service.
- When level detection is specified as the event input, the interrupt flag is set again even after it is cleared as long as the active level is kept at the input pin. In this case, the external cause of the request should be cleared or the interrupt enable bit should be cleared. Software clear has higher priority over hardware set.

6.11.3.3 Noise Filter

- Noise Filter removes the noise from the signal coming at pin INTn. The noise filter would filter the signals of width less than 50ns and passes all signals of width greater than 250ns.

6.11.3.4 External Interrupt Request Level

- When edge detection is specified as the event input, the pulse of the input signal must have a minimum width to be recognized as an input edge and not be filtered by the noise filter.
- When level detection is specified as the event input, the interrupt flag keeps active status once the specified level is input even after the input signal changes to the inactive level as shown in Figure 6-95. In order to clear the request, the interrupt flag must be cleared.
- For changing interrupt level or Noise Filter, software should use the following sequence to avoid any false triggering:

  1. Disable interrupt
  2. Change level
  3. Clear interrupt
  4. Enable interrupt

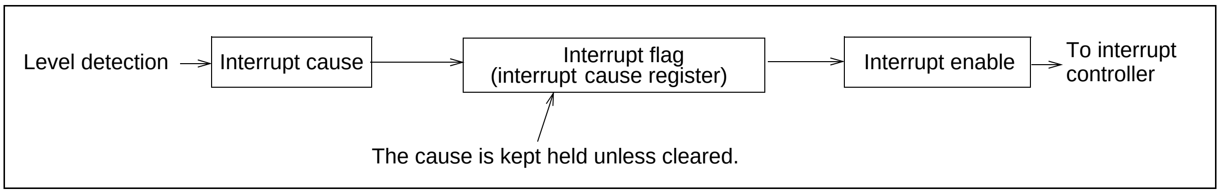


Figure 6-94: Clearing interrupt cause register upon level set

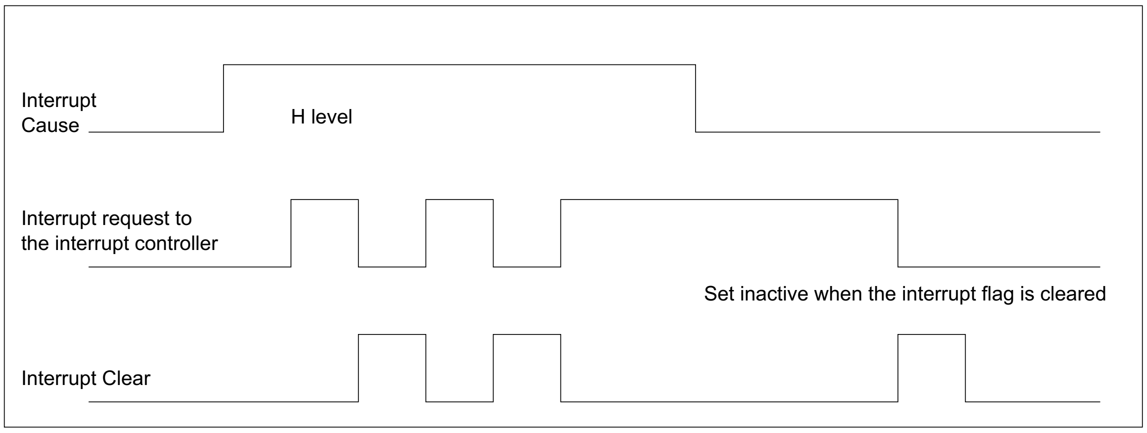


Figure 6-95: Interrupt cause and interrupt request to the interrupt controller while interrupts are enabled

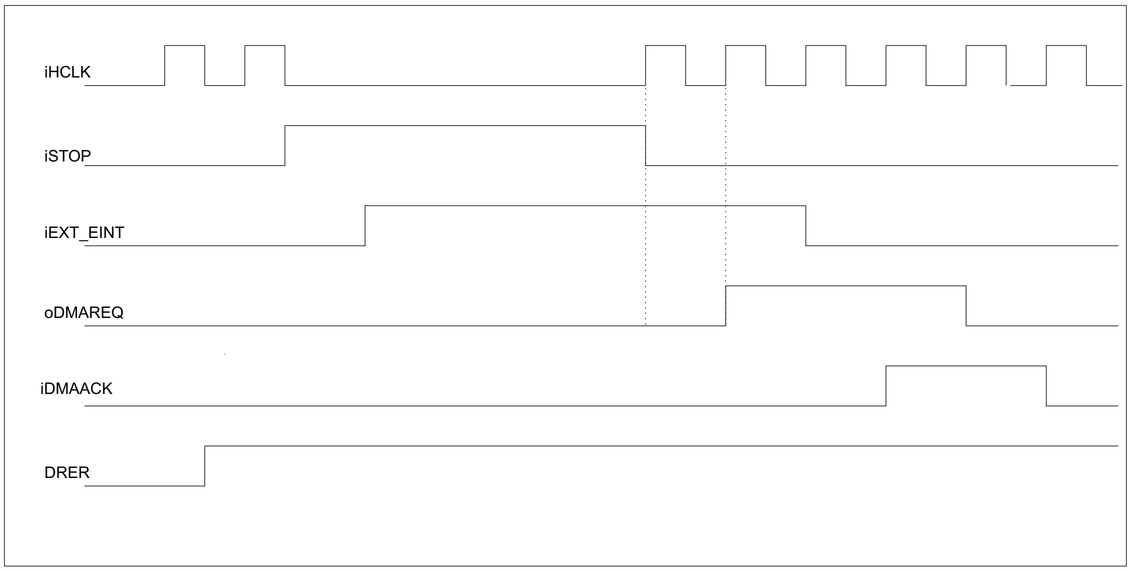


Figure 6-96: Interrupt Cause and DMAREQ to DMA while DMA is enabled

6.11.4 External Interrupt Register Overview

Table 6-41: Registers Overview

Base Address(es)	**Instance no 0: BASEADDR="000B0000"**	
Absolute Address	**Register Name**	**Register Description**
BASEADDR + 0x0000	ENIRn	External Interrupt Enable Register
BASEADDR + 0x0004	ENISRn	External Interrupt Enable Set Register
BASEADDR + 0x0008	ENICRn	External Interrupt Enable Clear R
BASEADDR + 0x000C	EIRRn	External Interrupt Request Register
BASEADDR + 0x0010	EIRCRn	External Interrupt Request Clear Register
BASEADDR + 0x0014	NFERn	Noise Filter Enable Register
BASEADDR + 0x0018	NFESRn	Noise Filter Enable Set Register
BASEADDR + 0x001C	NFECRn	Noise Filter Enable Clear Register
BASEADDR + 0x0020	ELVR0n	External Interrupt Level Register 0
BASEADDR + 0x0024	Reserved	Do not modify
BASEADDR + 0x0028	Reserved	Do not modify
BASEADDR + 0x002C	Reserved	Do not modify
BASEADDR + 0x0030	Reserved	Do not modify
BASEADDR + 0x0034	DRERn	DMA Request Enable Register
BASEADDR + 0x0038	DRESRn	DMA Request Enable Set Register
BASEADDR + 0x003C	DRECRn	DMA Request Enable Clear Register
BASEADDR + 0x0040	DRFRn	DMA Request Flag Register

Chapter 7: Electrical Characteristics

NOTE The content of this section is subject to changes without prior warning.

7.1 Absolute Maximum Ratings

Table 7-1: Absolute Maximum Ratings

Parameter	Symbol	Min	Max	Unit	Comment
Core supply	VDD	VSS – 0.3	VSS + 1.8	V	
Display supply	VDP3	VSS – 0.3	VSS + 4.0	V	
Stepper supply	HVDD	VSS – 0.3	VSS + 6.0	V	≥ VDP5
GPIO supply	VDP5	VSS – 0.3	VSS + 6.0	V	≥ VDP3
ADC supply	AVCC	VSS – 0.3	VSS + 6.0	V	= VDP5
APIX supply	VDDA VDDDA_VCO VDDDA_PLL VDDEA VDEA_PLL	VSS – 0.3 VSS – 0.3 VSS – 0.3 VSS – 0.3 VSS – 0.3	VSS + 1.8 VSS + 1.8 VSS + 1.8 VSS + 4.0 VSS + 4.0	V V V V V	
Input voltage	VI	VSS – 0.3 VSS – 0.3 VSS – 0.3	VDP5 + 0.3 VDP3 + 0.3 HVDD + 0.3	V V V	< 6.0 V < 4.0 V < 6.0 V
Analog input voltage	VIA	VSS – 0.3	AVCC + 0.3	V	< 6.0 V
APIX analog Input Voltage	VIAPX	VSS – 0.3	VDDEA + 0.3	V	< 4.0 V, SD-OUT, SDIN, VCM
Output voltage	VO	VSS – 0.3 VSS – 0.3 VSS – 0.3	VDP5 + 0.3 VDP3 + 0.3 HVDD + 0.3	V V V	< 6.0 V < 4.0 V < 6.0 V
Storage temperature	T_{ST}	-55	150	°C	

NOTE

- Applying stress exceeding the maximum ratings (voltage, current, temperature, etc.) may cause damage to semiconductor devices. Never exceed the ratings above.
- Never connect IC outputs or I/O pins directly, or connect them to VDD or VSS directly; otherwise thermal destruction of elements will result, but which does not apply to pins designed to prevent signal collision.
- Provide ESD protection, such as grounding when handling the product; otherwise externally-charged electric charge flows inside the IC and discharges, which may result in damage to the circuit.
- Applying voltage higher than VDD or lower than VSS to I/O pins of CMOS IC, or applying voltage higher than the ratings between VDD and VSS may cause latch up. The latch up increases supply current, resulting in thermal destruction of elements. When handling the product, never exceed the maximum ratings.

7.2 Recommended Operating Conditions

The recommended operating conditions are required in order to ensure the normal operation of the semiconductor device. All of the devices electrical characteristics are guaranteed when the device is operated within these ranges. Semiconductor devices must always be operated within their recommended operating condition ranges. Operating outside these ranges may adversely affect reliability and could result in device failure. No warranty is made with respect to uses, operating conditions, or combinations not represented in the data sheet. Users considering application fields beyond the listed conditions are advised to contact their Fujitsu representatives beforehand.

Table 7-2: Operating Conditions

Parameter	Symbol	Rating			Unit	Remarks
		Min	Typ	Max		
Core supply	VDD	1.1	1.2	1.3	V	
Display supply	VDP3 VDP3_PLL	3.0 3.0	3.3 3.3	3.6 3.6	V V	
Stepper supply	HVDD	4.5 3.0	5.0 3.3	5.5 3.6	V V	≥ VDP5
GPIO supply	VDP5	4.5 3.0	5.0 3.3	5.5 3.6	V V	 ≥ VDP3, only for IO usage
ADC supply	AVCC	4.5 3.0	5.0 3.3	5.5 3.6	V V	= VDP5
APIX supply	VDDA VDDDA_VCO VDDDA_PLL VDDEA VDEA_PLL	1.1 1.1 1.1 3.0 3.0	1.2 1.2 1.2 3.3 3.3	1.30 1.30 1.30 3.6 3.6	V V V V V	
Junction temperature	T*j*	-40		135	C	
Ambient temperature	Ta *[1]	-40		105	°C	Under JEDEC standard JESD51-2 conditions.
Case temperature	Tc *[1]	-40		115	°C	
*[1] Note: Both,operating conditions, Ta and TC, have to be fulfilled. Please refer to section “Thermal Design Considerations”						

7.2.1 Supply Modes

Three supply modes are supported for MB88F33x ‘Indigo2(-x)’.

Table 7-3: Supply Operational modes

VDP5	AVCC	HVDD	Comment
5.0V	5.0V	5.0V	
3.3V	3.3V	5.0V	no ZPD
3.3V	3.3V	3.3V	no Stepper

WARNING:

AVCC and VDP5 must be set to the same voltage. It is required that AVCC does not exceed VDP5 and that the voltage at the analog inputs does not exceed AVCC neither when the power is switched on.

HVDD, AVCC and VDP5 must be set to the same voltage during zero point detection (ZPD) on any of the SMC ports. If zero point detection is not required on any of the SMC ports, then VDP5 and AVCC can have any value which is equal or lower HVDD.

7.3 Power Consumption

Table 7-4: Supply currents

Parameter	Symbol	Rating			Unit	Remarks
		Min	Typ	Max		
Core supply Note 1)	I_{VDD}			350	mA	
Display supply Note 2)	I_{VDP3}			80	mA	Single TTL @40 MHz
				170	mA	Single TTL @85 MHz
				100	mA	Single RSDS
				200	mA	Dual RSDS
				40	mA	Single LVDS
				80	mA	Dual LVDS
	I_{VDP3_PLL}			10	mA	
Stepper supply Note 3)	I_{HVDD}			720	mA	max. 30 mA per pin
GPIO supply	I_{VDP5}			20	mA	
ADC supply	I_{AVCC}			5.0	mA	
APIX supply Note 4)	I_{VDDA}			55	mA	Daisy chain disabled
				100	mA	Daisy chain enabled
	I_{VDDA_VCO}			8.0	mA	
	I_{VDDA_PLL}			8.0	mA	
	I_{VDDEA}			20	mA	Daisy chain disabled
				50	mA	Daisy chain enabled
	I_{VDEA_PLL}			8.0	mA	

1) See “VDD Supply Current (Note 1)” below
2) See “Display IO Supply Current (Note 2)” below
3) See “Stepper IO Supply Current (Note 3)” below
4) See “APIX Supply Current (Note 4)” below

7.3.1 VDD Supply Current (Note 1)

The core supply current (I_{VDD}) mainly depend on the supply voltage, the chip temperature, and the internal frequencies. The given number is for maximum supply (1.3V), maximum temperature (105°C), and maximum internal frequencies. The following table give some more values, which allows a estimation for different use cases.

Table 7-5: Core supply currents

Operation mode	Ta_{max} =105°C		Ta_{max}=85°C	
	1.3V	1.2V	1.3V	1.2V
"axi_clk = 160MHz, peri_clk = 80MHz, pixel clock = 144MHz"	350mA	310mA	340mA	300mA
"axi_clk = 80MHz, peri_clk = 80MHz, pixel clock = 40MHz"	240mA	210mA	230mA	200mA

Internal clocks should be setup as low as possible for low power consumption. All clock divider can be reprogrammed during operation. So, it is for example possible to increase and decrease the AHB clock divider for short phases of high speed operations. The video clock frequencies depends on the selected display and define the internal pixel clock frequency. The minimum required axi_clk frequency can be estimated from the selected pixel frequency of the display. For standard setups the axi_clk should be set to be 10% to 30% higher than the pixel clock frequency.

The minimum required peri_clk frequency depends on the selected peripherals with their speed requirements.In addition, the core power consumption can be decreased by up to 10%, when disabling unused functions with the register PWR_CTRL.

7.3.2 Display IO Supply Current (Note 2)

For the estimation of the supply current I_{VDP3} the next rules can be followed:

- For every enabled differential pad the current rises by 7.5mA. For example, if 13 differential pads are enabled, it will consume 13 * 7.5mA = 97.5mA. This current is independent of the supply voltage or chip temperature.
- For all pins, when used as a CMOS output, the maximum current depends on the supply voltage, on the toggle rate, and the load capacitance. The current scales nearly linear with these parameters.
- The values in the table for the TTL panels give the maximum value for high supply voltage (3.6V), when using a ‘state-of-the-art’ TTL 24-bit panel connected though a ribbon cable with a realistic video content. Different systems may require more or less current.

7.3.3 Stepper IO Supply Current (Note 3)

The maximum current value in the table is the maximum current which Indigo2 can deliver. For a stepper application, where the stepper is controlled by a sinusoidal way, the current for the 4 pins connected to one stepper can be estimated as:

$$\text{Current\_for\_one\_stepper} = 4 \times \frac{1}{\sqrt{2}} \times \text{Current\_for\_one\_pin}$$

For power dissipation in Indigo2 one has to use the VOL and VOH of the IO cells. The maximum value for both is 0.5V. The power than is estimated as:

$$\text{Power\_for\_one\_stepper} = 0.5V \times \text{Current\_for\_one\_stepper}$$

7.3.4 APIX Supply Current (Note 4)

The supply currents for the APIX in Indigo2 are independent of the operation mode. There are two main influences for this currents.

First, if daisy chain is enabled or disabled (see values in the table).

Second, the selected drive strength for the transmitter outputs. This influences the I_{VDDEA} current. The values in the table give the maximum possible current.

For low power it should be ensure, that the daisy chain part of the APIX PHY is in power down when not needed (register PHY_PWR_CTRL.en_lt = 0). In addition, the drive strength should be set to the minimum setting that is required for the application (register PHY_RX_TST.rx_upstream_swing).

7.3.5 Thermal Design Considerations

The maximum permissible case temperature (T_c) is 115C. To ensure the device’s reliability and its proper operation, do not exceed this temperature.

NOTE Indigo2 is not the only contributor to the thermal performance of the entire system. The PCB characteristics and layout, as well as the ambient temperature must also be taken into consideration to comply with the maximum case temperature restriction.

The estimated junction-to-case thermal resistance (ΦCA) is 34 C/W for a 4-layer PCB with no air flow and no heat sink. This thermal performance depends not only on the Indigo2 package, but also on the characteristics of the PCB on which it is mounted.

The power consumption varies according to the application (i.e., this depends on the use case).

7.4 DC Limits

Latch-up may occur in a CMOS IC, if a voltage higher than (VDD, HVDD, VDP3 or VDP5) or less than (VSS) is applied to an input or output pin. Or, if a voltage exceeding the rating is applied between the power supply pins and ground pins. If latch-up occurs, the power supply current increases rapidly, sometimes resulting in thermal breakdown of the device.

Therefore, be very careful not to apply voltages in excess of the absolute maximum ratings.

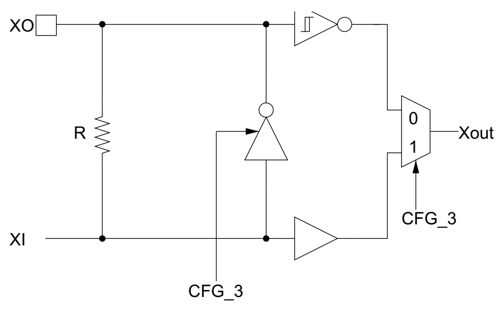
If unused input pins are left open, abnormal operation may result. Any unused input pins should be connected to pull-up or pull-down resistor (2KOhm to 10KOhm) or enable internal pull-up or pull-down resistors.

The supply voltage to the I2C-BUS lines (SDA and SCL) must not exceed the power-supply voltage of this I/O cell (VDP5). You must not supply voltage to the I2C-BUS lines (SDA and SCL), if the power supply of this I/O cell (VDP5) is off.

7.5 IO Circuits

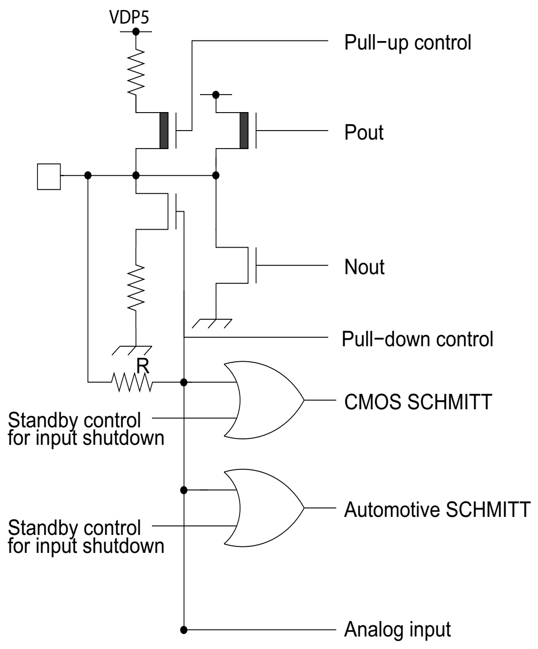
Table 7-6 shows all different IO circuit types used in MB88F33x ‘Indigo2(-x)’. The different IO circuit types listed here, correspond to the column D “Pin Type” in the attached excel file “pinning.xls”.

Table 7-6: IO circuit types

Type	Circuit	Remarks
OSC		■ VDEA-PLL IO supply domain ■ High-speed oscillation circuit ■ Programmable between oscillation mode (external crystal or resonator connected to XI/XO pins) and Clock input (CFG_3) mode (external clock connected to XI pin). ■ Input frequency: 30MHz APIX ■ Internal feedback resistor: 1MOhm (typ.) ■ Clock input mode (XI). Please refer to the following table for this mode:

Parameter	Symbol	Min	Typ	Max
CMOS	VIH	0.8* VDEA_PLL		VDEA_PLL
	VIL			0.2* VDEA_PLL
Input leakage	IL	-1µA		+1µA

Table 7-6: IO circuit types

Type	Circuit	Remarks
BIDI50		■ VDP5 IO supply domain ■ CMOS output level ■ Programmable output drive strength ■ CMOS SCHMITT / Automotive SCHMITT input /Analog input ■ Programmable pull-up and pull-down resistor

■ VDP5 IO supply domain

■ CMOS output level

Parameter	Symbol	Min	Typ	Max
High output	VOH	VDP5-0.5V		VDP5
Low output	VOL	0V		0.4V

■ Programmable output drive strength

Drive Setting	Symbol	Min	Typ	Max
00	IOL / IOH	± 1mA		
01	IOL / IOH	± 2mA		
10	IOL / IOH	± 5mA		
11	IOL / IOH	± 2mA		

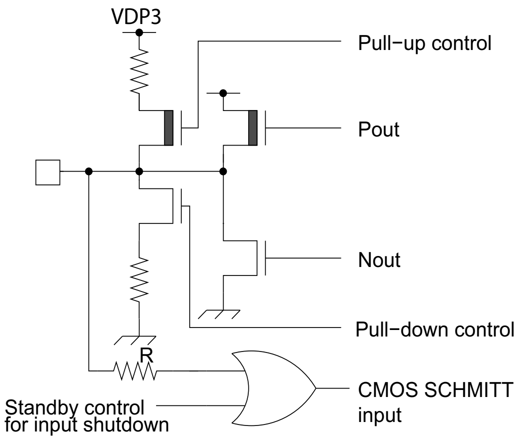
■ CMOS SCHMITT / Automotive SCHMITT input /Analog input

Parameter	Symbol	Min	Typ	Max
CMOS	VIH	0.8*VDP5		VDP5
	VIL	0V		0.2*VDP5
Automo-tive	VIH	0.8*VDP5		VDP5
	VIL	0V		0.5*VDP5
Input leakage	IL	-5µA		+5µA

■ Programmable pull-up and pull-down resistor

Parameter	Symbol	Min	Typ	Max
Pull-up/ pull-down	R	25 kOhm	50 kOhm	100 kOhm

Table 7-6: IO circuit types

Type	Circuit	Remarks
BIDI33	 VDP3 Pull−up control Pout Nout Pull−down control R Standby control for input shutdown CMOS SCHMITT input	(see below)

- VDP3 IO supply domain
- CMOS level output

Parameter	Symbol	Min	Typ	Max
High output	VOH	VDP3-0.5V		VDP3
Low output	VOL	0V		0.4V

- Output drive strength

Drive Setting	Symbol	Min	Typ	Max
	IOL / IOH	± 4mA		

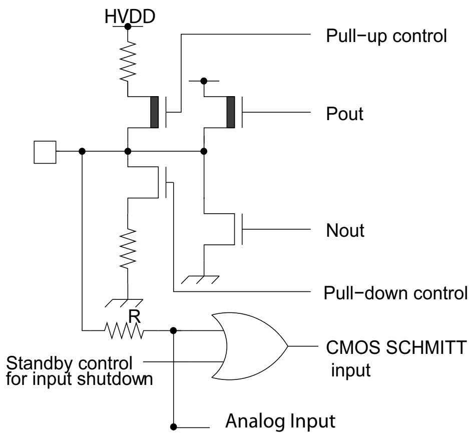
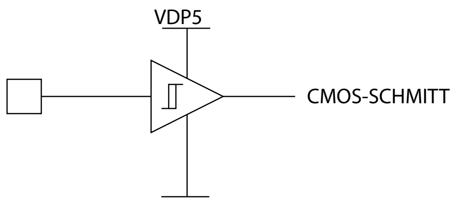
- CMOS SCHMITT input

Parameter	Symbol	Min	Typ	Max
CMOS	VIH	0.8*VDP3		VDP3
	VIL	0V		0.2*VDP3
Input leakage	IL	-5µA		+5µA

- Programmable pull-up and pull-down resistor

Parameter	Symbol	Min	Typ	Max
Pull-up/ pull-down	R	15 kOhm	33 kOhm	70 kOhm

Table 7-6: IO circuit types

Type	Circuit	Remarks
SMC		■ HVDD IO supply domain ■ CMOS output level ■ Programmable output drive strength ■ CMOS SCHMITT / Analog input ■ Programmable pull-up and pull-down resistor
IN50		■ VDP5 IO supply domain ■ CMOS SCHMITT input

SMC:

■ HVDD IO supply domain

■ CMOS output level

Parameter	Symbol	Min	Typ	Max
High output	VOH	HVDD-0.5V		HVDD
Low output	VOL	0V		0.4V

■ Programmable output drive strength

Drive Setting	Symbol	Min	Typ	Max
00	IOL / IOH	± 1mA		
01	IOL / IOH	± 2mA		
10	IOL / IOH	± 30mA		
11	IOL / IOH	± 5mA		

■ CMOS SCHMITT / Analog input

Parameter	Symbol	Min	Typ	Max
CMOS	VIH	0.8*VDP5		VDP5
	VIL	0V		0.2*VDP5
Input leakage	IL	-5µA		+5µA

■ Programmable pull-up and pull-down resistor

Parameter	Symbol	Min	Typ	Max
Pull-up/ pull-down	R	25 kOhm	50 kOhm	100 kOhm

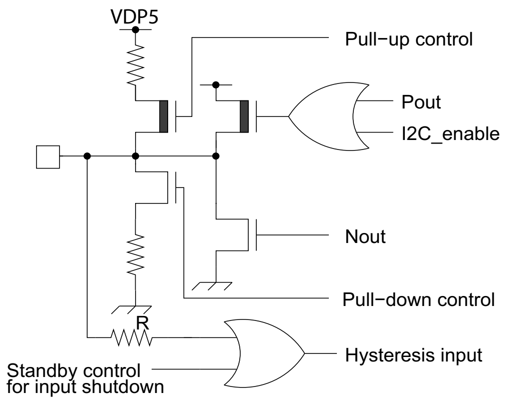
IN50:

■ VDP5 IO supply domain

■ CMOS SCHMITT input

Parameter	Symbol	Min	Typ	Max
CMOS	VIH	0.8*VDP5		VDP5
	VIL	0V		0.2*VDP5
Input leakage	IL	-5µA		+5µA

Table 7-6: IO circuit types

Type	Circuit	Remarks
I2C	 VDP5 Pull−up control Pout I2C_enable Nout Pull−down control R Hysteresis input Standby control for input shutdown	See remarks below

- VDP5 IO supply domain
- CMOS output level

Parameter	Symbol	Min	Typ	Max
High output	VOH	VDP5-0.5V		VDP5
Low output	VOL	0V		0.4V

- Programmable output drive strength

Drive Setting	Symbol	Min	Typ	Max
00	IOL / IOH	± 1mA		
01	IOL / IOH	± 2mA		
10	IOL / IOH	± 5mA		
11	IOL / IOH	± 2mA		
*	IOL	± 3mA		
	IOH	(Pseudo Open drain) [*1]		

*1: For Pseudo Open Drain output logic value “1”, Push/Pull CMOS driver is switched to HIZ state.

- CMOS SCHMITT (Automotive SCHMITT input / Analog input

Parameter	Symbol	Min	Typ	Max
CMOS	VIH	0.8*VDP5		VDP5
	VIL	0V		0.2*VDP5
Input leakage	IL	-5µA		+5µA

- Programmable pull-up and pull-down resistor

Parameter	Symbol	Min	Typ	Max
Pull-up/ pull-down	R	25 kOhm	50 kOhm	100 kOhm

Table 7-6: IO circuit types

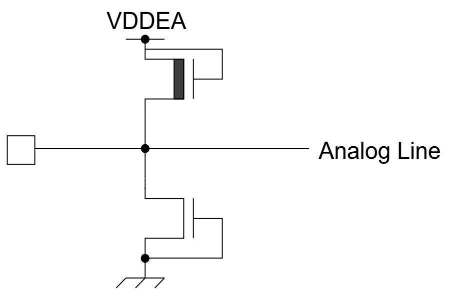
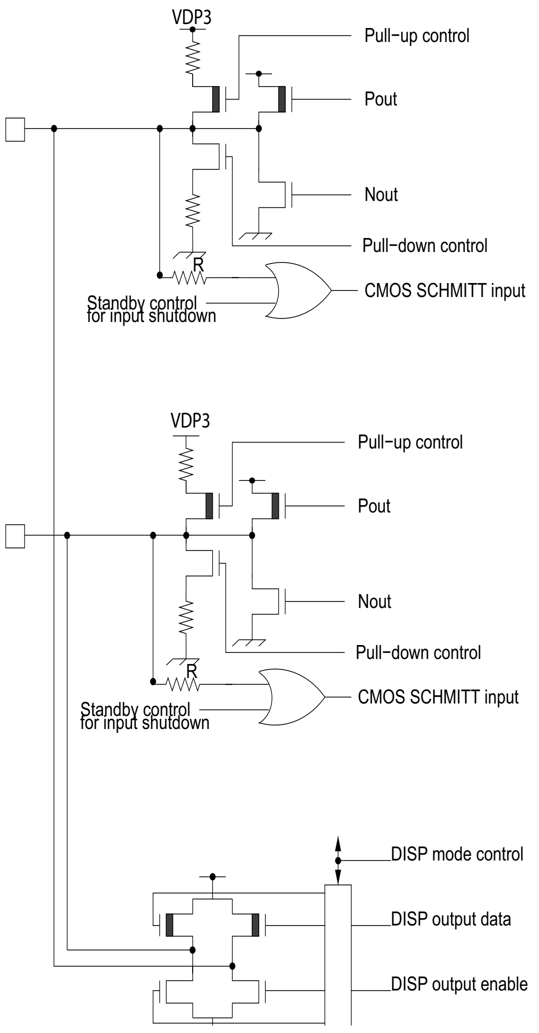
Type	Circuit	Remarks
AIO		■ VDDEA IO supply domain ■ Analog Pin ■ Type INPUT: Analog input pin with ESD protection ■ Type Output: Analog output line with ESD protection.

Table 7-6: IO circuit types

Type	Circuit	Remarks
DISP_D		■ VDP3 IO supply domain

■ CMOS output level

Parameter	Symbol	Min	Typ	Max
High output	VOH	VDP3-0.5V		VDP3
Low output	VOL	0V		0.5V

■ Programmable output drive strength

Drive Setting	Symbol	Min	Typ	Max
00	IOL / IOH	± 2mA		
01	IOL / IOH	± 5mA		
10	IOL / IOH	± 10mA		

■ CMOS SCHMITT input

Parameter	Symbol	Min	Typ	Max
CMOS	VIH	0.8*VDP3		VDP3
	VIL	0V		0.2*VDP3
Input leakage	IL	-5µA		+5µA

■ Programmable pull-up/pull-down resistor

Parameter	Symbol	Min	Typ	Max
Pull-up/ pull-down	R	15kOhm	33kOhm	70kOhm

■ Differential output level

●RSDS 100 Ohm Termination

Parameter	Symbol	Conditions	Min	Typ	Max	Unit
Output differential voltage	VOD	Rload=100 Ohm PAD_CTRLB: 0x0 (RSDS100)	150	250	350	mV
Output offset voltage	VOS		1.0	1.2	1.3	V
Output current amplitude	Iload100		1.5	2.5	3.5	mA

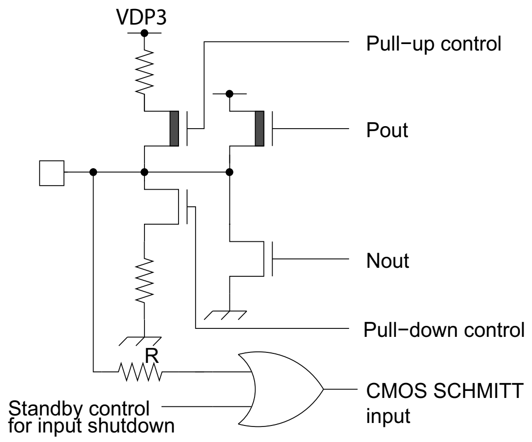
●RSDS 50 Ohm Termination

Parameter	Symbol	Conditions	Min	Typ	Max	Unit
Output differential voltage	VOD	Rload=50 Ohm, PAD_CTRLB: 0x2 (RSDS50 or LVDS)	125	175	225	mV
Output offset voltage	VOS		1.0	1.2	1.3	V
Output current amplitude	Iload50		2.5	3.5	4.5	mA

●LVDS

Parameter	Symbol	Conditions	Min	Typ	Max	Unit
Output differential voltage	VOD	Rload=100 Ohm PAD_CTRLB: 0x2 (RSDS50 or LVDS)	250	350	450	mV
Output offset voltage	VOS		1.125	1.25	1.375	V
Output current amplitude	Iload100		2.5	3.5	4.5	mA

Table 7-6: IO circuit types

Type	Circuit	Remarks
DISP_S	 VDP3 Pull−up control Pout Nout Pull−down control R Standby control for input shutdown CMOS SCHMITT input	■ VDP3 IO supply domain ■ CMOS output level ■ Programmable output drive strength ■ CMOS SCHMITT input ■ Programmable pull-up/pull-down resistor

■ CMOS output level

Parameter	Symbol	Min	Typ	Max
High output	VOH	VDP3-0.5V		VDP3
Low output	VOL	0V		0.5V

■ Programmable output drive strength

Drive Setting	Symbol	Min	Typ	Max
00	IOL / IOH	± 2mA		
01	IOL / IOH	± 5mA		
10	IOL / IOH	± 10mA		
11	IOL / IOH	± 30mA		

■ CMOS SCHMITT input

Parameter	Symbol	Min	Typ	Max
CMOS	VIH	0.8*VDP3		VDP3
	VIL	0V		0.2*VDP3
Input leakage	IL	-5µA		+5µA

■ Programmable pull-up/pull-down resistor

Parameter	Symbol	Min	Typ	Max
Pull-up/ pull-down	R	15 kOhm	33 kOhm	70 kOhm

7.6 AC Limits

7.6.1 Host SPI Characteristics

7.6.1.1 Host SPI Interface

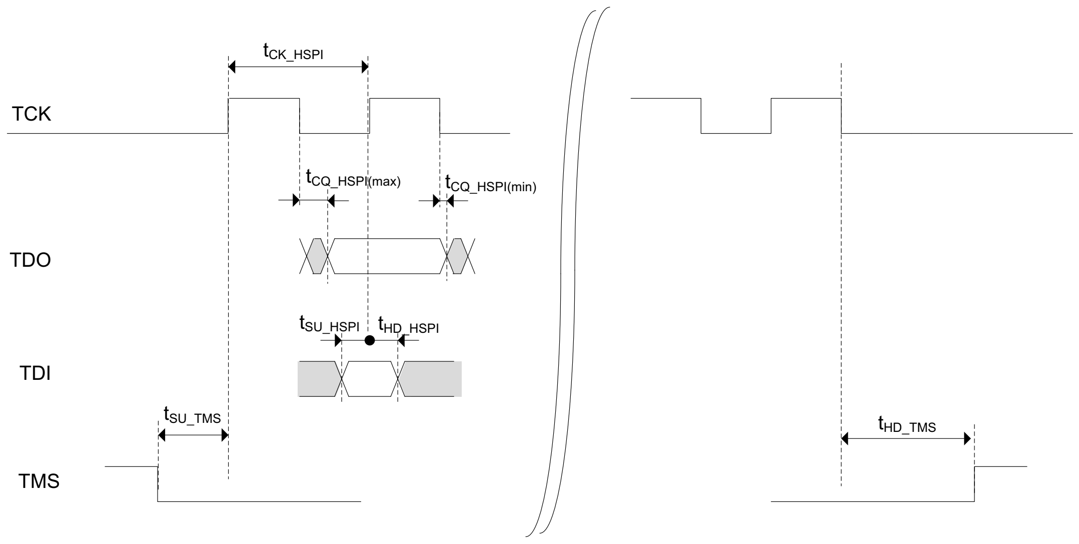


Figure 7-1: Timing SPI Interface

Table 7-7: AC Timing Host-SPI Interface

Parameter	Symbol	Value			Unit	Remarks
		Min	Typ	Max		
clk period	t_{CK_HSPI}	100			ns	Maximum 2x of HCLK period.
clk to output data	t_{CQ_HSPI}	0		20	ns	
Input data setup	t_{SU_HSPI}	10			ns	
Input data hold	t_{HD_HSPI}	5			ns	
Input Control setup	t_{HD_TMS}	50 + 2*tHCLK			ns	
Input Control Hold	t_{HD_TMS}	50 + 2*tHCLK			ns	

7.6.2 Config Interface

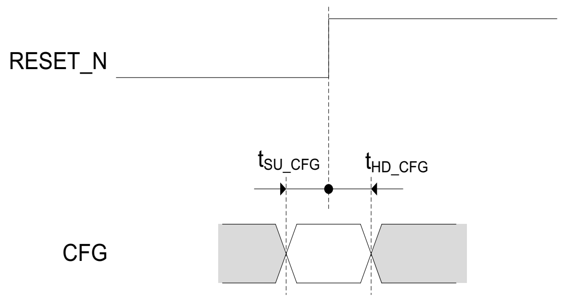


Figure 7-2: Timing configuration pins

Table 7-8: AC Timing configuration pjns

Parameter	Symbol	Value			Unit	Remarks
		Min	Typ	Max		
cfg data setup	t_{SU_CFG}	50			ns	
cfg data hold	t_{HD_CFG}	250			ns	

7.6.3 Display Interface

7.6.3.1 TTL Mode

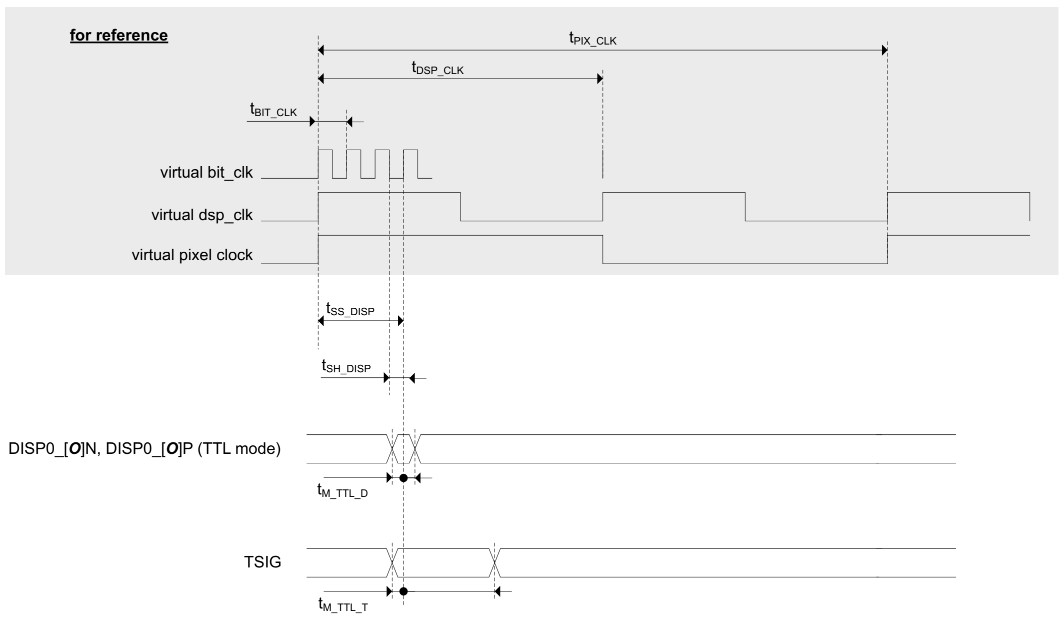


Figure 7-3: Timing Display TTL Interface

Table 7-9: AC Timing TTL Display Interface

Parameter	Symbol	Value			Unit	Remarks
		Min	Typ	Max		
dsp_clk period	t_{DSP_CLK}	5.5			ns	Internal clock for reference only
bit_clk period	t_{BIT_CLK}	1.8			ns	Internal clock for reference only, integer multiple of dsp_clk
Pixel clock period	t_{PIX_CLK}	11	11.7		ns	typical value is maximum pixel frequency, minimum value is due to spread spectrum and clock synthesis
Shift value	t_{SS_DISP}	typ -150	$n \times t_{\mathrm{BIT\_CLK}}$	typ +150	ps	
Half cycle shift	t_{SH_DISP}	typ -200	$\frac{t_{\mathrm{BIT\_CLK}}}{2}$	typ +200	ps	
TTL DISP mismatch	$t_{M\_TTL\_D}$	-0.5		+0.5	ns	
TSIG TTL mismatch	$t_{M\_TTL\_T}$	1.5		4.5	ns	Related to center of DISP outputs

7.6.3.2 RSDS Mode

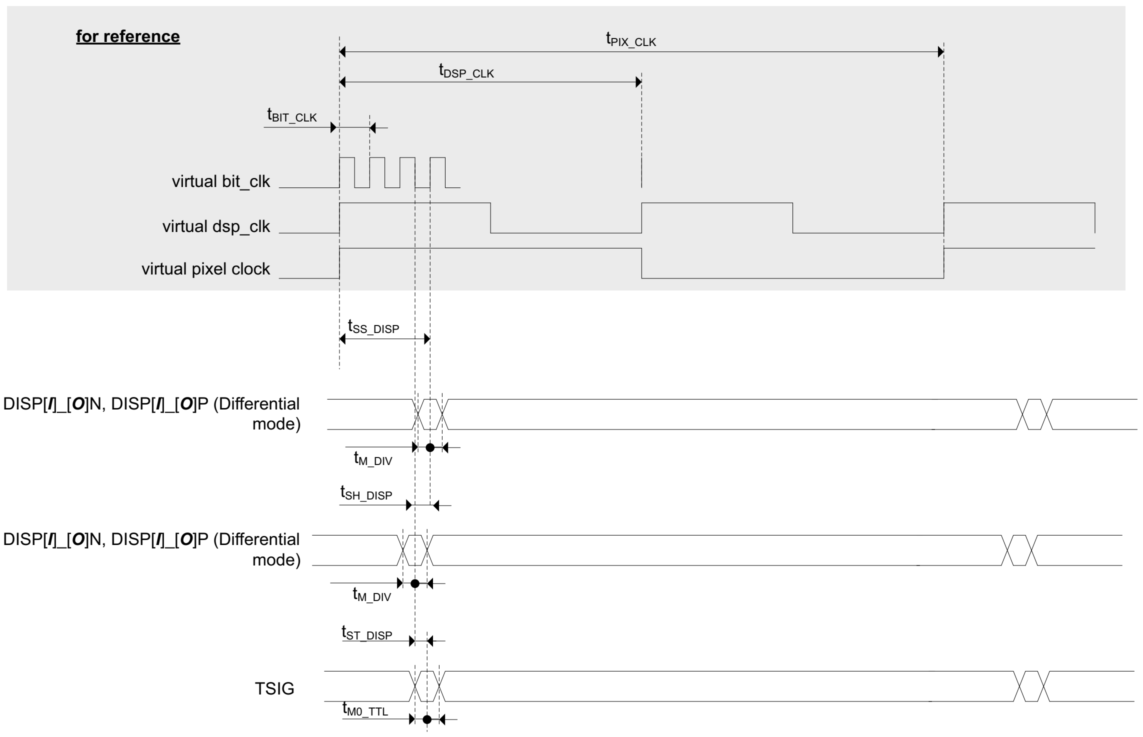


Figure 7-4: Timing Display RSDS Interface

Table 7-10: AC timings RSDS display interface

Parameter	Symbol	Value			Unit	Remarks
		Min	Typ	Max		
dsp_clk period	t_{DSP_CLK}	5.5			ns	Internal clock for reference only
bit_clk period	t_{BIT_CLK}	1.8			ns	Internal clock for reference only, integer multiple of dsp_clk
Pixel clock period	t_{PIX_CLK}	11	11.7		ns	typical value is maximum pixel frequency, minimum value is due to spread spectrum and clock synthesis
Shift value	t_{SS_DISP}	typ-150	$n \times t_{\mathrm{BIT\_CLK}}$	typ+150	ps	
Half cycle shift	t_{SH_DISP}	typ-200	$\frac{t_{\mathrm{BIT\_CLK}}}{2}$	typ+200	ps	
TSIG output mismatch	t_{M_TTL}	-1.0		+1.0	ns	
RSDS to TSIG shift	t_{ST_DISP}	0.4	2.5	4.6	ns	
RSDS output mismatch	t_{M_DIV}	-0.5		+0.5	ns	

7.6.4 SPI Interface (External SPI and Flash SPI)

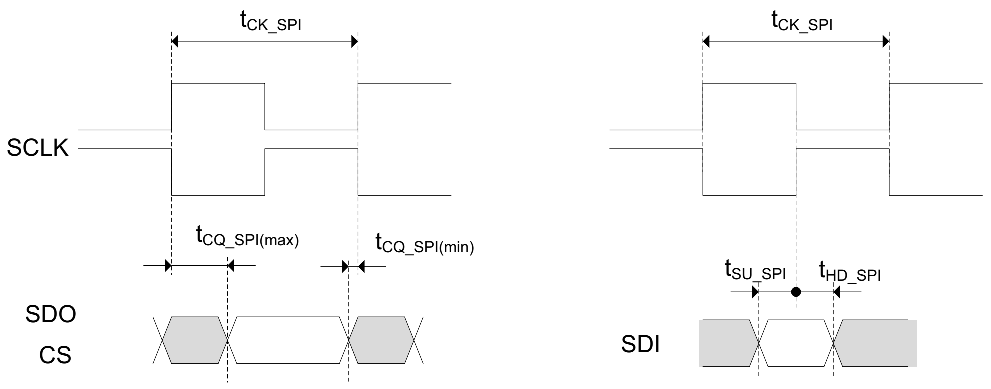


Figure 7-5: Timing SPI Interface

Table 7-11: AC Timings SPI Interface

Parameter	Symbol	Value			Unit	Remarks
		Min	Typ	Max		
clk period	t_{CK_SPI}	25			ns	Period depends on selected AHB clock or Peripheral clock fre-quency.
clk to output data	t_{CQ_SPI}	-4		9.5	ns	Active clock edge depends on interface setup.
input data setup	t_{SU_SPI}	15 7.5			ns ns	Active clock edge depends on interface setup. No re-timing mode. Re-timing mode.
input data hold	t_{HD_SPI}	-3 2.5			ns ns	Active clock edge depends on interface setup. No re-timing mode. Re-timing mode.

7.6.5 I2C Interface

The Indigo2 fulfills the timing requirements for the standard mode and fast mode of the Philips I2C specification.

The supply voltage to the I2C-BUS lines (SDA and SCL) must not exceed the power-supply voltage of this I/O cell (VDP5).

You must not supply voltage to the I2C-BUS lines (SDA and SCL), if the power supply of this I/O cell (VDP5) is off.

7.6.6 USART/LIN Interface

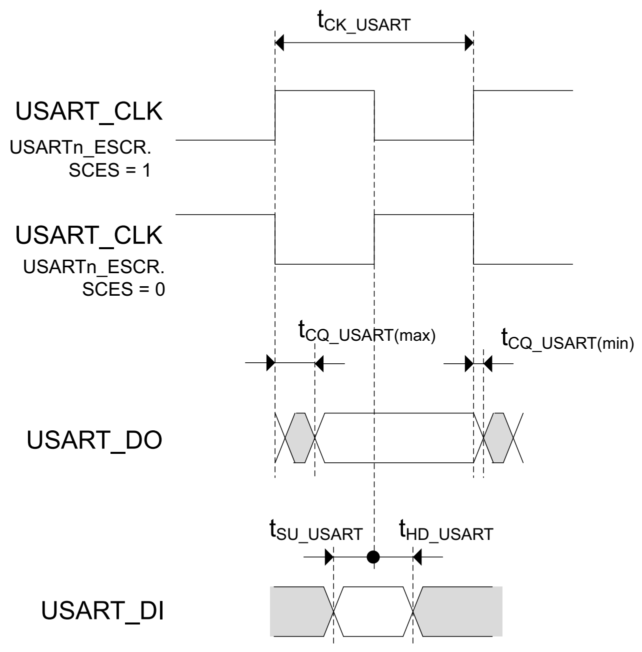


Figure 7-6: Timing U(S)ART Interface

Table 7-12: AC Timings U(S)ART Interface

Parameter	Symbol	Value			Unit	Remarks
		Min	Typ	Max		
CLK period	t_{CK_USART}	4 x t_{rbus_clk}			ns	
CLK to output data	t_{CQ_USART}	-5		20 2 x $_{bus\_clk}$ + 45	ns	internal CLK mode external CLK mode
Input data setup	t_{SU_USART}	t_{rbus_clk} + 25			ns	
Input data hold	t_{HD_USART}	t_{rbus_clk}			ns	

7.6.7 I2S Interface

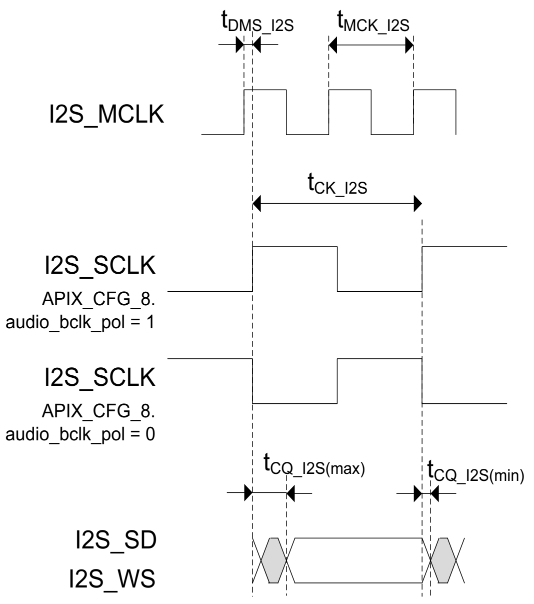


Figure 7-7: Timing I2S Interface

Table 7-13: AC timings I2S Interface

Parameter	Symbol	Value			Unit	Remarks
		Min	Typ	Max		
MCLK period	t_{MCK_I2S}	18.5			ns	
SCLK period	t_{CK_I2S}	37			ns	Half frequency of MCLK.
MCLK to SCLK delay	t_{DMS_I2S}	0		10	ns	
SCLK to output data	t_{CQ_I2S}	-5		10	ns	

7.6.8 MII Interface

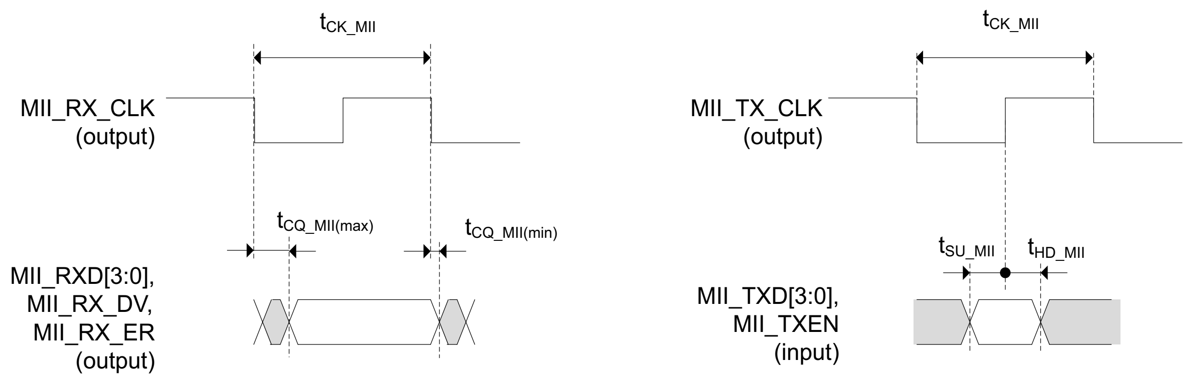


Figure 7-8: Timing MII Interface in APIX IO mode (external Ethernet MAC connected)

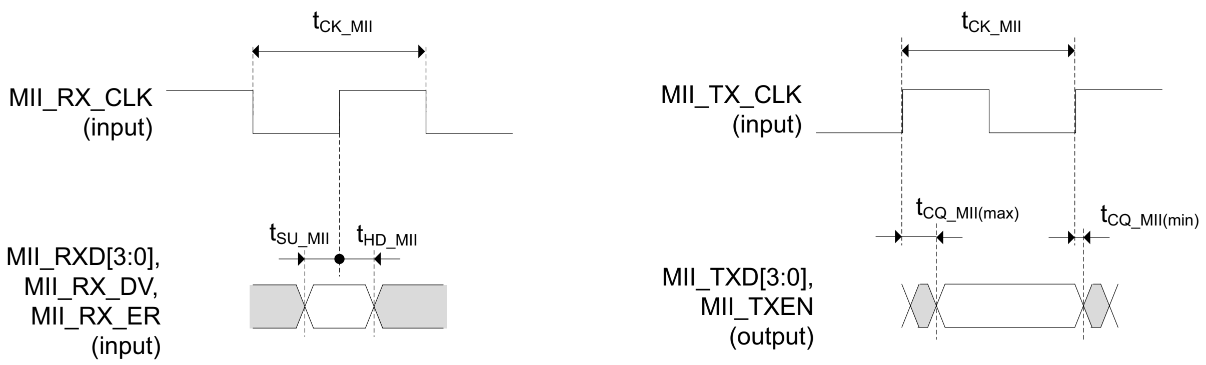


Figure 7-9: Timing MII Interface in E2IP IO mode (external Ethernet PHY connected)

Table 7-14: AC timings MII Interface

Parameter	Symbol	Value			Unit	Remarks
		Min	Typ	Max		
MII_CLK period	t_{CK_MII}		40		ns	100Mbit
			400		ns	10Mbit
Output delay	t_{CQ_MII}	0		10	ns	1)
Input data setup	t_{SU_MII}	20			ns	
Input data hold	t_{HD_MII}	0			ns	
1) For maximum drive strength setting						

7.7 Clock Input

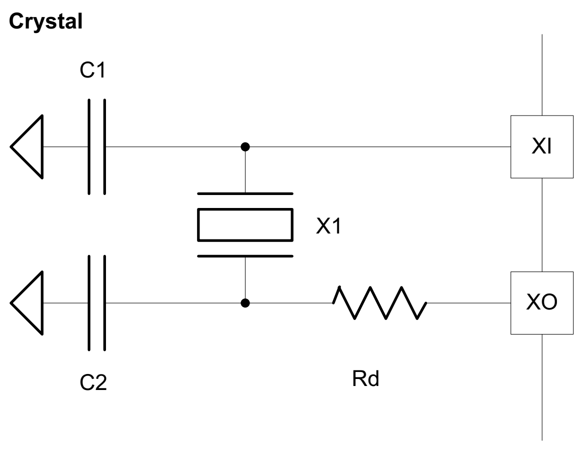


Figure 7-10: Clock Input

Table 7-15: Clock Input

Parameter	Symbol	Value			Unit	Remarks
		Min	Typ	Max		
Crystal frequency	X1	-100 ppm	30	+100 ppm	MHz	
External load capacity	C1,C2		10		pF	Value depends on Crystal
Damping resistor	Rd		0		Ohm	If needed, value depends on Crystal
Coupling capacity	Cc		100		pF	
Input amplitude	V_{IH_XI}	0.8 * VDEA_PLL			V	If external clock is input at XI, see CFG_3 at section "2.6 Bootstrap Configuration"
	V_{IL_XI}			0.2 * VDEA_PLL	V	

7.8 Reset Timing

The low active reset input (RESET_N) has to be low for at least t_{RST}.

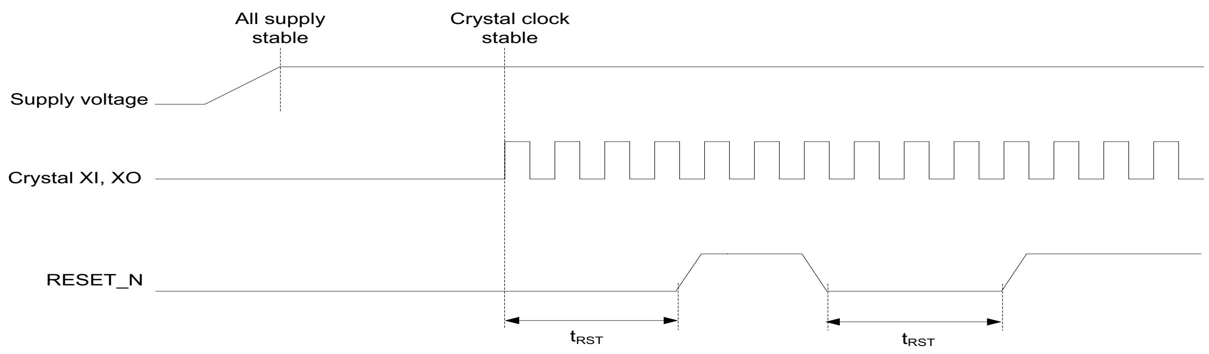


Figure 7-11: Reset Timing

Table 7-16: Clock Input

Parameter	Symbol	Value			Unit	Remarks
		Min	Typ	Max		
Reset low time	t_{RST}	100			us	

7.9 Power-up

At any time, the difference between the power supply pins belonging to the same voltage level must not exceed 0.5V. This especially applies to the power on sequence. Otherwise, the risk of latchup will increase. Figure 7-12 shows the power on sequence and the groups of power supply that might be used, depending on the actual application. Furthermore, VDP5 supply must be switched on before any other power supply or at least at the same time.

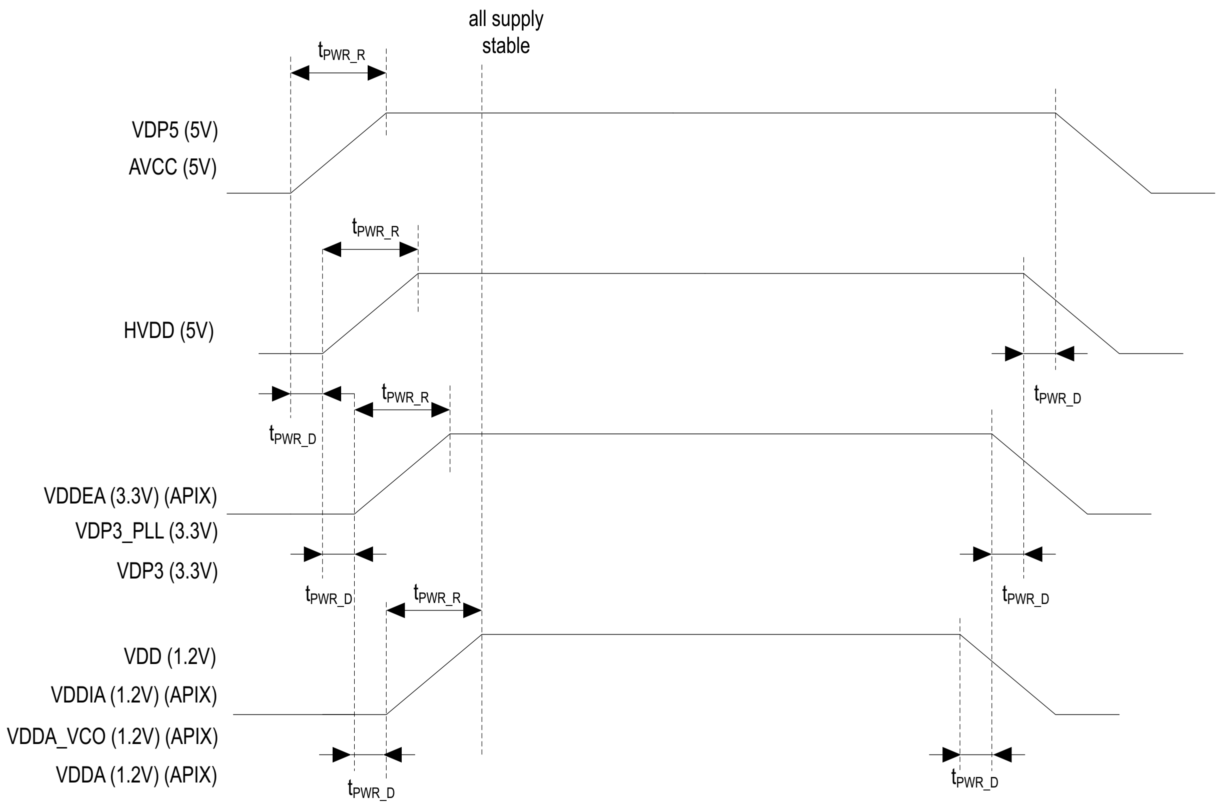


Figure 7-12: Supply Power on Sequence

Table 7-17: Timing power on

Parameter	Symbol	Min	Typ	Max	Unit	Comment
Power Rise Time	t_{PWR_R}	0.05		30	ms	
Power Rise Delay	t_{PWR_D}	0		1	s	
Power Slew Rate		0.1		20	mV/us	

NOTE The supply VDP5 has to be kept higher that VDD in all conditions.

7.10 ADC

7.10.1 Sampling Time

MB88F33x ‘Indigo2(-x)’ has an embedded 10-bit successive approximation ADC with an internal integrated sampling and hold stage. The signal will charge the sampling capacitor at first and then the voltage signal on the sampling capacitor will be evaluated by the 10-bit ADC successively. The time to charge the sampling capacitor to its final value equal to the signal level is a function of the internal and external capacitor and resistor values. To reduce the error caused by the limited settling time to an acceptable level, the settling time should be chosen much larger than the time constant to charge the sampling capacitor. The settling time can be set with the ST register field of the CT register in the ADC register space.

The minimum sampling time can be calculated from the following formula:

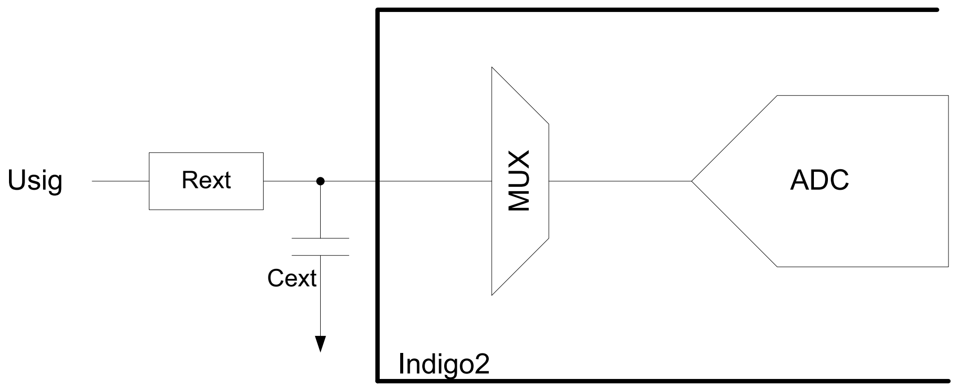


Figure 7-13: ADC input signal

When VDP5 = HVDD = nominal 5V

For pins **ADC0 .. ADC15**:

$$Tsamp[min] = 7.63 \cdot [Rext \cdot (Cext + 16pF) + (Rext + 1.8k\Omega) \cdot 20pF]$$

Without external components:

$$Tsamp[min] = 275ns$$

For pins **ADC16 .. ADC27**:

$$Tsamp[min] = 7.63 \cdot [Rext \cdot (Cext + 16pF) + (Rext + 1.8k\Omega) \cdot 6pF + (Rext + 3.6k\Omega) \cdot 20pF]$$

Without external components:

$$Tsamp[min] = 632ns$$

When VDP5 = HVDD = nominal 3.3V

For pins **ADC0 .. ADC15**:

$$Tsamp[min] = 7.63 \cdot [Rext \cdot (Cext + 16pF) + (Rext + 4.3k\Omega) \cdot 20pF]$$

Without external components:

$$Tsamp[min] = 656ns$$

For pins **ADC16 .. ADC27**:

$$Tsamp[min] = 7.63 \cdot [Rext \cdot (Cext + 16pF) + (Rext + 4.3k\Omega) \cdot 6pF + (Rext + 8.6k\Omega) \cdot 20pF]$$

Without external components:

$$Tsamp[min] = 1.51\mu s$$

7.11 FLASH Memory Program/Erase Characteristics

Table 7-18: Program/Erase Time

Parameter	Value			Unit	Remarks
	Min	Typ [1)]	Max		
Sector erase Time	-	0.3	1.5	s	The internal programming time before the erase procedure starts is included.
Macro Erase Time	-	1.2	12	s	
Word Programming Time	-	12	384	µs	
1) Typical definition: T_a=25°C / V_{DD}=1.2V / Program/Erase cycle= Immediately after shipment					

Table 7-19: Program/Erase Cycle and Data Retention Time [2)]

Program/Erase Cycle at Each Sector		Data Retention Time	
Min Value	Unit	Min Value	Unit
1000	cycles	20	years
10000	cycles	10	years
100000	cycles	5	years
2) These values were converted from the technology qualification using Arrhenius equation to translate high temperature measurements into normalized values at +85°C			

Table 7-20: Execution Time Limit

Parameter	Value [3)]	Unit
Program Execution Time Limit [4)]	1.3	ms
Macro Erase Execution Time Limit	62.4	s
Sector Erase Execution Time Limit	7.8	s
3) These values are development target values and may be changed depending on device evaluation results. 4) This is the time it takes for the macro to detect a ‘Hang-up 1’ error, when 1 is to be programmed to a memory cell, whose memory value is either 0 or X.		

7.12 SMC Outputs

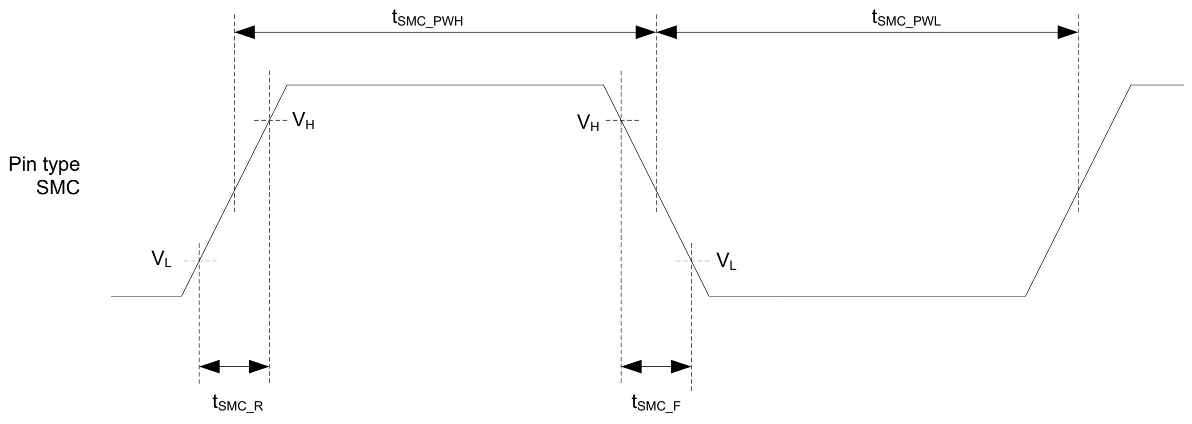


Figure 7-14: Slew Rate of SMC output

Table 7-21: SMC rise/fall time

Parameter	Symbol	Min	Typ	Max	Unit	Comment
Output Rse/Fall Time	t_{SMC_R} t_{SMC_F}	15		100	ns	Min for C_{LOAD} = 0pF Max for C_{LOAD} = 100pF VH = 0.9 x HVDD VL = 0.1 x HVDD Output driving strength set to 30mA
Output Pulse Width	t_{SMC_PWH}	2.5			µs	Output driving strength set to 30mA
Output Pulse Width	t_{SMC_PWL}	2.5			µs	Output driving strength set to 30mA

7.13 Low Voltage Detection

The low voltage detection circuit supervises the core supply (VDD) and the GPIO supply (VDP5). Please refer to section “2.7.3 Low Voltage Detection (LVD)” in Chapter “Global Control”.

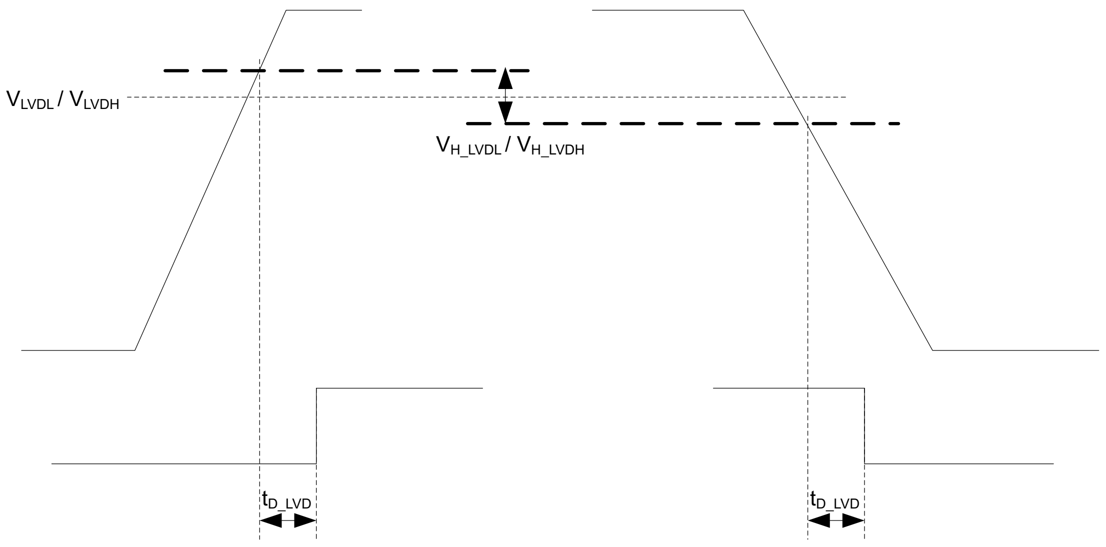


Figure 7-15: Low voltage detection

Table 7-22: Low voltage detection

Parameter	Symbol	Min	Typ	Max	Unit	Comment
VDP5 detection voltage	V_{LVDH}	2.0 2.2 2.4 2.5 3.5 3.7 3.9 4.1	2.2 2.4 2.6 2.7 3.7 3.9 4.1 4.3	2.4 2.6 2.8 2.9 3.9 4.1 4.3 4.5	V V V V V V V V	SVH setting = 0 SVH setting = 1 SVH setting = 3 SVH setting = 2 SVH setting = 6 SVH setting = 7 SVH setting = 5 SVH setting = 4
VDP5 detection hysteresis	V_{H_LVDH}	75	100	150	mV	
VDD detection voltage	V_{LVDL}	0.8 0.9 1.0 1.1	0.9 1.0 1.1 1.2	1.0 1.1 1.2 1.3	V V V V	SVL setting = 6 SVL setting = 7 SVL setting = 5 SVL setting = 4
VDD detection hysteresis	V_{H_LVDL}	20	30	50	mV	
VDD/VDP5 detection delay	t_{D_LVD}			30	us	
Startup Time	t_{PU_LVD}			80	us	

7.14 PCB Layout Recommendations

7.14.1 Automotive Pixel Link (APIX)

Refer to the APIX Layout Recommendation Application Note "Apix PCB-Design Guideline".

7.14.2 Configuration Pins

The following solutions are recommended, when using the configuration pins.

- Unused Pin with Pull-down

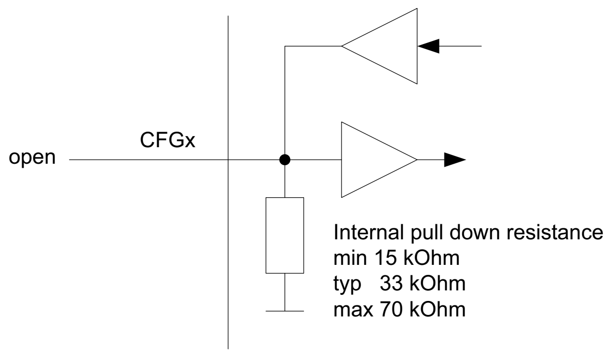


Figure 7-16: Unused pin with pull-down

- Unused Pin with Pull-up

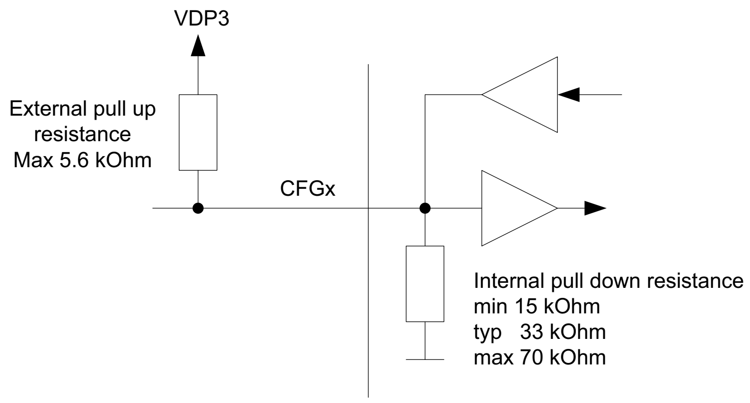


Figure 7-17: Unused pin with pull-up

After power-up, the internal pull-down muss be switched off to avoid power leakage.

- Configuration pins are output - External device does not support pull-up

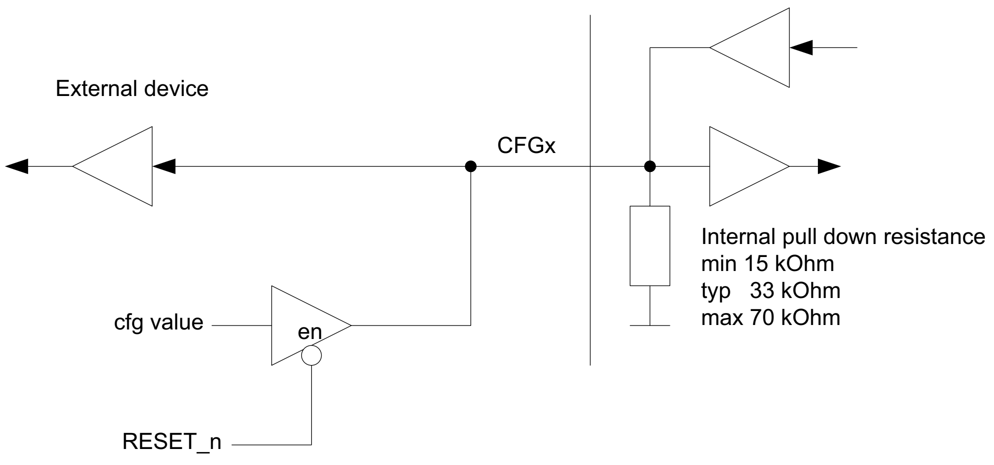


Figure 7-18:

- Configuration pins are input - External device does not support pull-up

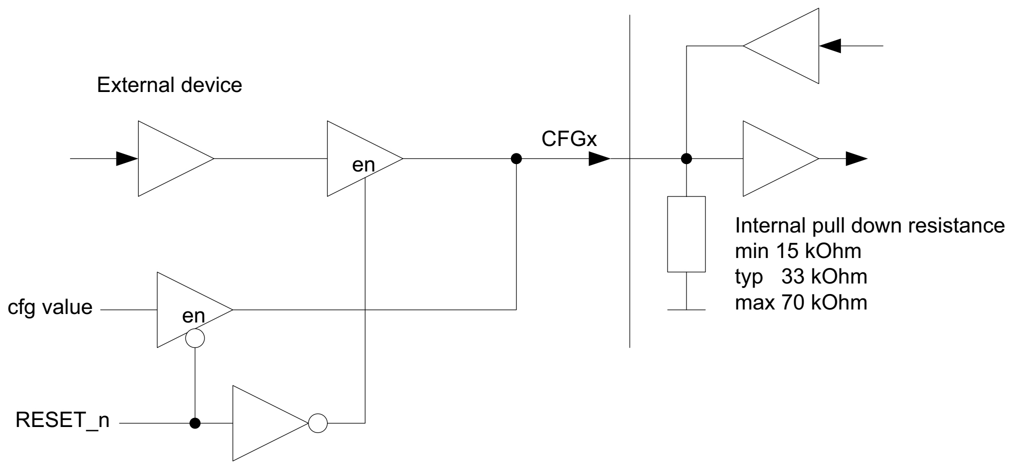


Figure 7-19:

In this case, we recommend to implement a tri-state buffer on the board and an additional tri-state buffer in order to disconnect the external device from the CFG-signals. After power-up, the internal pull-down should be disconnected.

- IO - External Device Does Not Support Pull-up

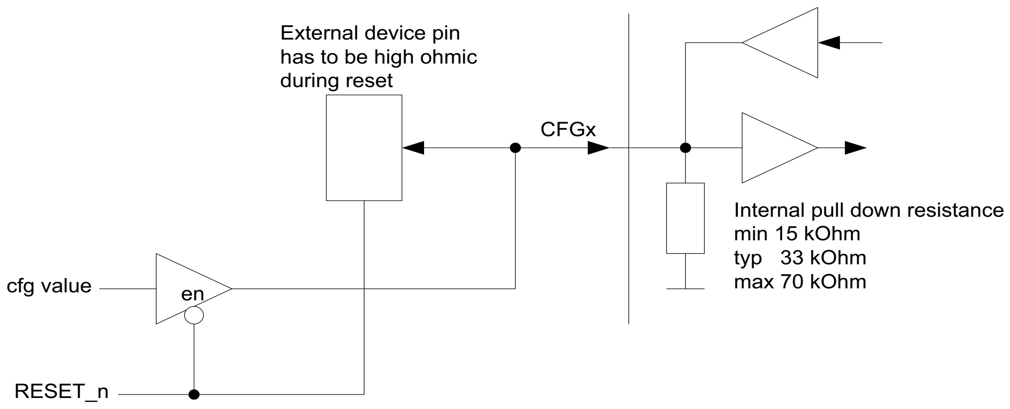


Figure 7-20:

In this case, the external device must be in high-impedance state during reset. After power-up, the internal pull-down should be disconnected.

WARRANTY AND DISCLAIMER

Fujitsu Semiconductor Europe GmbH - Graphics Competence Center
Forstenrieder Str. 10, D - 82061 Neuried Germany
Tel: +49 (89) 218 938 - 0
Fax: +49 (89) 218 938 - 401
http://emea.fujitsu.com/semiconductor