HOLTEK

Battery Charger Flash MCU with CAN Bus

HT45F5QC-5

Revision: V1.20 Date: September 05, 2025

www.holtek.com

Features

CPU Features

- Operating Voltage
  - f_{SYS}=8MHz: 2.2V~5.5V
- Up to 0.5µs instruction cycle with 8MHz system clock at V_{DD}=5V
- Power down and wake-up functions to reduce power consumption
- Oscillator Types
  - Internal High Speed 8MHz RC – HIRC
  - Internal Low Speed 32kHz RC – LIRC
- Fully integrated internal oscillators require no external components
- Multi-mode operation: FAST, SLOW, IDLE and SLEEP
- All instructions executed in 1~3 instruction cycles
- Table read instructions
- 115 powerful instructions
- 8-level subroutine nesting
- Bit manipulation instruction

Peripheral Features

- Flash Program Memory: 8K×16
- Data Memory: 512×8
- True EEPROM Memory: 512×8
- In Application Programming function – IAP
- Watchdog Timer function
- 16 bidirectional I/O lines
- 2 external interrupt lines shared with I/O pins
- Multiple Timer Modules for time measure, input capture, compare match output, PWM output function or single pulse output function
- Dual Time Base functions for generation of fixed time interrupt signals
- 9 external channel 12-bit resolution A/D converter with Internal Reference Voltage V_{VR}
- Battery charger circuit
  - 14-bit D/A Converter and OPA0 are used for constant current control
  - 14-bit D/A Converter and OPA1 are used for constant voltage control
  - OPA2 is 20/40 times amplifier for current sense
- Integrated 16-bit Cyclic Redundancy Check function – CRC
- Low voltage reset function
- Package types: 28-pin SSOP, 32-pin QFN

CAN Bus Controller Features

- Operating Voltage: 3.0V~5.5V
- Oscillator Type: High Speed External Crystal – HXT
- Sleep Mode and Idle Mode

- 32-byte Write Buffer with Data Check Unit
- Clock Out pin with programmable prescaler
- Interrupt output pins with selectable active level configuration
- Conforms to ISO11898-1 and CAN 2.0A/B
- 32 Message Objects
- Each Message Object has its own identifier mask
- Programmable FIFO mode - concatenation of Message Objects
- Maskable interrupt
- Programmable loop-back mode for self-test operation
- Support the SOF (Start of Frame) signal output
- 32×139-bit Message Memory

General Description

The HT45F5QC-5 is a Flash Memory A/D type 8-bit high performance RISC architecture microcontroller especially designed for battery charger and CAN bus applications.

For memory features, the Flash Memory offers users the convenience of Flash Memory multi-programming features. Other memory includes an area of RAM Data Memory as well as an area of true EEPROM memory for storage of non-volatile data such as serial numbers, calibration data etc. By using the In Application Programming technology, user have a convenient means to directly store their measured data in the Flash Program Memory as well as having the ability to easily update their application programs.

Analog feature includes a multi-channel 12-bit A/D converter function. Multiple and extremely flexible Timer Modules provide timing, pulse generation and PWM generation functions. Protective features such as an internal Watchdog Timer and Low Voltage Reset coupled with excellent noise immunity and ESD protection ensure that reliable operation is maintained in hostile electrical environments.

A full choice of internal low and high speed oscillators is provided including two fully integrated system oscillators which require no external components for their implementation. The ability to operate and switch dynamically between a range of operating modes using different clock sources gives users the ability to optimise microcontroller operation and minimise power consumption.

For AC/DC charger applications, the device includes a battery charger management module, which can be used for the constant voltage and constant current closed loop charging control. The device therefore reduces the need for the usually required external TL431 component, operational amplifier and resistance analogic D/A Converter in traditional battery charging circuits. Therefore the peripheral circuit is more reduced, resulting in a smaller PCB area.

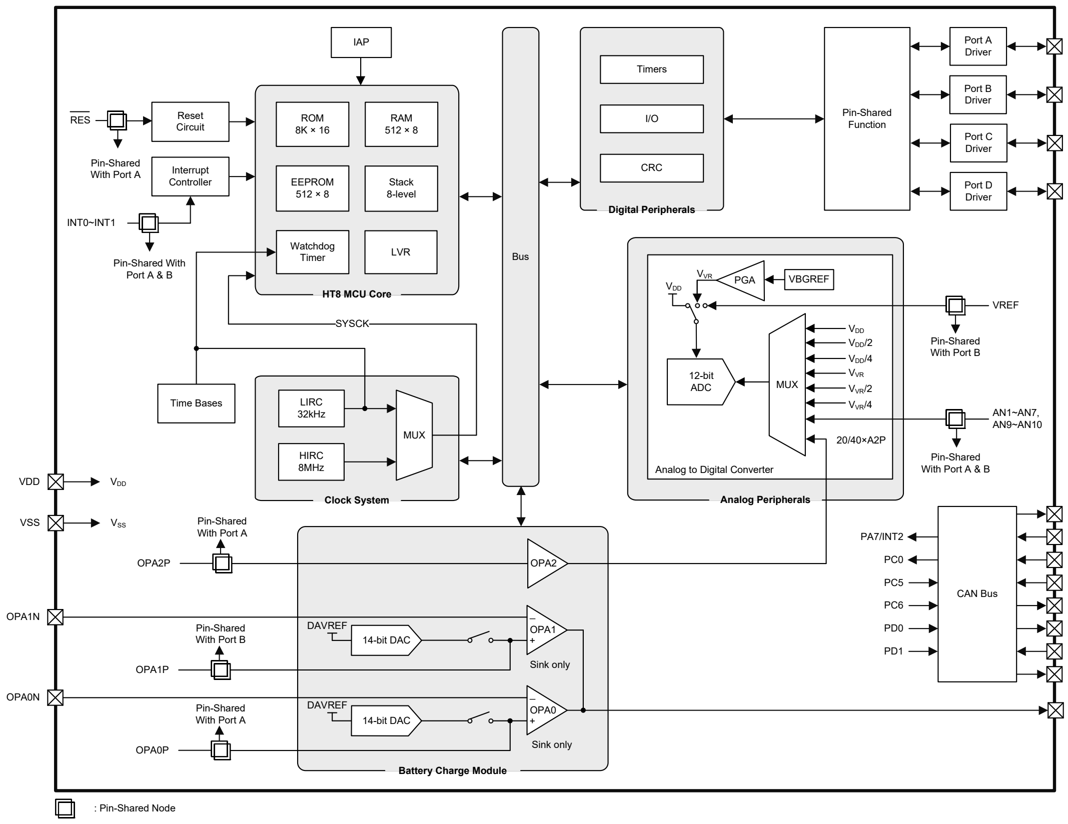
The charger management module is composed of two parts. The first part contains two groups of OPAs and D/A Converters, which are used to control the charging voltage and current. The upper limit value of the charger constant current and constant voltage can be obtained by configuring the D/A Converters in the software. The 14-bit D/A Converter is used for constant current control while another 14-bit D/A Converter is used for constant voltage control. The second part of the charger management contains an operational amplifier with the gain of 20 or 40 which is used for current amplification. This improves the current resolution and allows the use of smaller current detection resistors thus reducing the resistor power consumption.

The D/A Converter in the charger management module is not only used for setting charging voltage and current, but also can be used together with the specific charger production fixtures for improving the traditional manual calibration techniques. By using the external production fixtures, the charger current voltage/current conditions can be confirmed. If the margin of errors is exceeded, the MCU will correct the error by fine tuning the D/A Converter, and store the corrected parameters to EEPROM. When the charger is recharged, the D/A Converter will be given a new correction value to implement correction purpose. Refer to the Holtek application notes for more details.

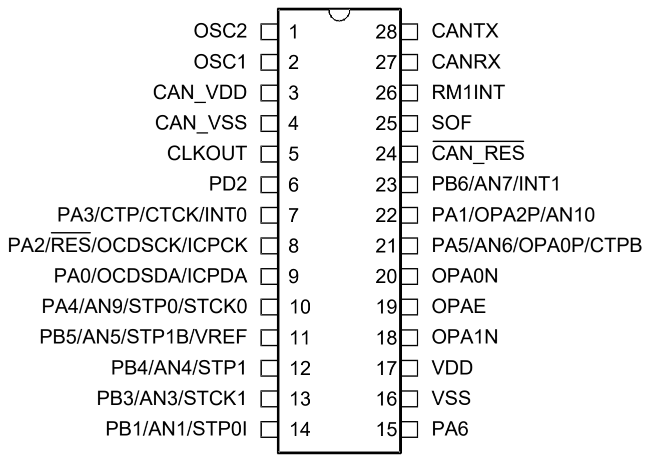
The device includes a fully integrated CAN (Controller Area Network) bus controller. The CAN Module licensed from Bosch supports the CAN 2.0 Part A and B protocol specifications and compatible with the ISO11898-1 standards. This CAN Module abbreviated as C_CAN. It is capable of transmitting and receiving standard and extended messages. It is also capable of both acceptance filtering and message handler and includes 32 Message Objects which can be concatenated to configure FIFO buffer with different depth.

The inclusion of flexible I/O programming features, Time Base functions along with many other features, further enhance device functionality and flexibility for wide range of application possibilities.

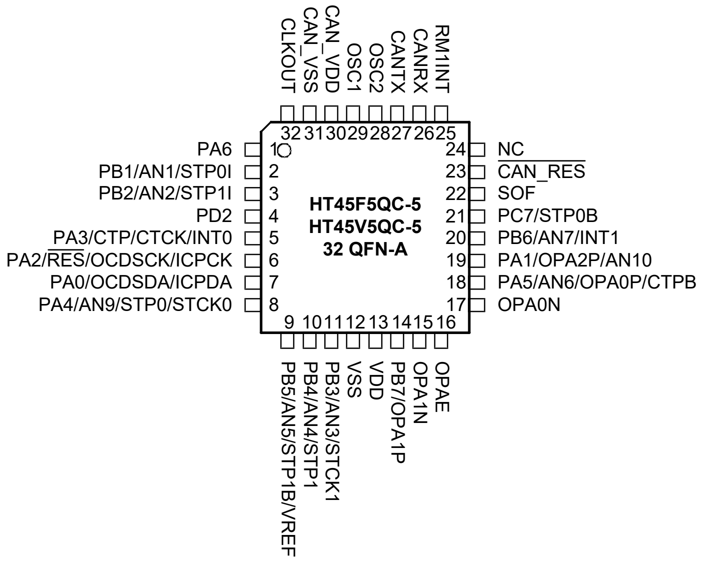
Block Diagram



Pin Assignment



HT45F5QC-5/HT45V5QC-5
28 SSOP-A



Note: 1. If the pin-shared pin functions have multiple outputs, the desired pin-shared function is determined by the corresponding software control bits.

2. The OCDSDA and OCDSCK pins are supplied for the OCDS dedicated pins and as such only available for the HT45V5QC-5 device which is the OCDS EV chip for the HT45F5QC-5 device.

3. For the unbounded pins, the line status should be properly configured to avoid unwanted power consumption resulting from floating input conditions. Refer to the "Standby Current Considerations" and "Input/ Output Ports" sections.

Pin Descriptions

The function of each pin is listed in the following table, however the details behind how each pin is configured is contained in other sections of the datasheet. As the pin description table shows the situation for the package with the most pins, not all pins in the table will be available on smaller package sizes.

Pin Name	Function	OPT	I/T	O/T	Description
PA0/OCDSDA/ ICPDA	PA0	PAPU PAWU	ST	CMOS	General purpose I/O. Register enabled pull-high and wake-up
	OCDSDA	—	ST	CMOS	OCDS data/address pin, for EV chip only
	ICPDA	—	ST	CMOS	ICP data/address pin
PA1/OPA2P/AN10	PA1	PAPU PAWU PAS0	ST	CMOS	General purpose I/O. Register enabled pull-high and wake-up
	OPA2P	PAS0	AN	—	Operational amplifier 2 positive input
	AN10	PAS0	AN	—	A/D Converter analog input
PA2/$\overline{RES}$/OCDSCK/ ICPCK	PA2	RSTC	—	CMOS	General purpose output pin
	$\overline{RES}$	RSTC	ST	—	External reset input
	OCDSCK	—	ST	—	OCDS clock pin, for EV chip only
	ICPCK	—	ST	—	ICP clock pin
PA3/CTP/CTCK/ INT0	PA3	PAPU PAWU PAS0	ST	CMOS	General purpose I/O. Register enabled pull-high and wake-up
	CTP	PAS0	—	CMOS	CTM output
	CTCK	PAS0	ST	—	CTM clock input
	INT0	PAS0 INTEG INTC0	ST	—	External interrupt
PA4/AN9/STP0/ STCK0	PA4	PAPU PAWU PAS1	ST	CMOS	General purpose I/O. Register enabled pull-high and wake-up
	AN9	PAS1	AN	—	A/D Converter analog input
	STP0	PAS1	—	CMOS	STM0 output
	STCK0	PAS1	ST	—	STM0 clock input
PA5/AN6/OPA0P/ CTPB	PA5	PAPU PAWU PAS1	ST	CMOS	General purpose I/O. Register enabled pull-high and wake-up
	AN6	PAS1	AN	—	A/D Converter analog input
	OPA0P	PAS1	AN	—	Operational amplifier 0 positive input
	CTPB	PAS1	—	CMOS	CTM inverted output
PA6	PA6	PAPU PAWU PAS1	ST	CMOS	General purpose I/O. Register enabled pull-high and wake-up
PB1/AN1/STP0I	PB1	PBPU PBS0	ST	CMOS	General purpose I/O. Register enabled pull-high
	AN1	PBS0	AN	—	A/D Converter analog input
	STP0I	PBS0	ST	—	STM0 capture input
PB2/AN2/STP1I	PB2	PBPU PBS0	ST	CMOS	General purpose I/O. Register enabled pull-high
	AN2	PBS0	AN	—	A/D Converter analog input
	STP1I	PBS0	ST	—	STM1 capture input

Pin Name	Function	OPT	I/T	O/T	Description
PB3/AN3/STCK1	PB3	PBPU PBS0	ST	CMOS	General purpose I/O. Register enabled pull-high
	AN3	PBS0	AN	—	A/D Converter analog input
	STCK1	PBS0	ST	—	STM1 clock input
PB4/AN4/STP1	PB4	PBPU PBS1	ST	CMOS	General purpose I/O. Register enabled pull-high
	AN4	PBS1	AN	—	A/D Converter analog input
	STP1	PBS1	—	CMOS	STM1 output
PB5/AN5/STP1B/ VREF	PB5	PBPU PBS1	ST	CMOS	General purpose I/O. Register enabled pull-high
	AN5	PBS1	AN	—	A/D Converter analog input
	STP1B	PBS1	—	CMOS	STM1 inverted output
	VREF	PBS1	AN	—	A/D Converter external reference voltage input
PB6/AN7/INT1	PB6	PBPU PBS1	ST	CMOS	General purpose I/O. Register enabled pull-high
	AN7	PBS1	AN	—	A/D Converter analog input
	INT1	PBS1 INTEG INTC0	ST	—	External interrupt
PB7/OPA1P	PB7	PBPU PBS1	ST	CMOS	General purpose I/O. Register enabled pull-high
	OPA1P	PBS1	AN	—	Operational amplifier 1 positive input
PC7/STP0B	PC7	PCPU PCS1	ST	CMOS	General purpose I/O. Register enabled pull-high
	STP0B	PCS1	—	CMOS	STM0 inverted output
PD2	PD2	PDPU	ST	CMOS	General purpose I/O. Register enabled pull-high
OPA0N	OPA0N	—	AN	—	Operational amplifier 0 negative input
OPA1N	OPA1N	—	AN	—	Operational amplifier 1 negative input
OPAE	OPAE	—	—	AN	Operational amplifier output
CLKOUT	CLKOUT	—	—	CMOS	CAN bus clock output pin with programmable prescaler
OSC1	OSC1	—	AN	—	CAN Bus HXT oscillator input
OSC2	OSC2	—	—	AN	CAN Bus HXT oscillator output
CANTX	CANTX	—	—	CMOS	Transmit output pin to CAN Bus
CANRX	CANRX	—	ST	—	Receive input pin from CAN Bus
RM1INT	RM1INT	—	—	CMOS	Receive a Message into Message Object 1 Successfully Interrupt output
$\overline{\text{CAN\_RES}}$	$\overline{\text{CAN\_RES}}$	—	ST	—	CAN Bus external reset input
SOF	SOF	—	—	CMOS	Start of Frame signal output
VDD	VDD	—	PWR	—	Positive power supply
CAN_VDD	CAN_VDD	—	PWR	—	CAN Bus digital positive power supply
VSS	VSS	—	PWR	—	Negative power supply
CAN_VSS	CAN_VSS	—	PWR	—	CAN Bus digital negative power supply
NC	NC	—	—	—	Not connected

Legend: I/T: Input type	O/T: Output type
OPT: Optional by register option	ST: Schmitt Trigger input
CMOS: CMOS output	NMOS: NMOS output
AN: Analog signal	PWR: Power

Interconnection Signal Description

Several signals listed in the following table are not connected to external package pins. These signals are interconnection lines between the MCU and the CAN Bus. Users should properly configure the relevant control bits to implement correct interconnection.

MCU Signal	CAN Bus Signal	Function	Description
PA7/INT2	RMXINT	PA7	General purpose I/O Internally connected to the CAN Bus RMXINT
		INT2	External interrupt input Internally connected to the CAN Bus RMXINT
		RMXINT	Receive a Message into Message Object x Successfully Interrupt output, the x value is user-defined using the CANCFG register Internally connected to the MCU PA7/INT2
PC0	CANMINT	PC0	General purpose I/O Internally connected to the CAN Bus CANMINT
		CANMINT	CAN Interrupt output Internally connected to the MCU PC0
PC5	MOSI	PC5	General purpose I/O Internally connected to the CAN Bus MOSI
		MOSI	Master Out Slave In, CAN Bus SPI interface data input Internally connected to the MCU PC5
PC6	MISO	PC6	General purpose I/O Internally connected to the CAN Bus MISO
		MISO	Master In Slave Out, CAN Bus SPI interface data output Internally connected to the MCU PC6
PD0	SCK	PD0	General purpose I/O Internally connected to the CAN Bus SCK
		SCK	CAN Bus SPI interface clock input Internally connected to the MCU PD0
PD1	$\overline{CS}$	PD1	General purpose I/O Internally connected to the CAN Bus $\overline{CS}$
		$\overline{CS}$	CAN Bus SPI interface chip select input Internally connected to the MCU PD1

Absolute Maximum Ratings

Supply Voltage	V_{SS}-0.3V to 6.0V
Input Voltage	V_{SS}-0.3V to V_{DD}+0.3V
Storage Temperature	-60°C~150°C
Operating Temperature	-40°C to 105°C
I_{OH} Total	-80mA
I_{OL} Total	80mA
Total Power Dissipation	500mW

Note: These are stress ratings only. Stresses exceeding the range specified under "Absolute Maximum Ratings" may cause substantial damage to the device. Functional operation of the device at other conditions beyond those listed in the specification is not implied and prolonged exposure to extreme conditions may affect device reliability.

D.C. Characteristics

For data in the following tables, note that factors such as oscillator type, operating voltage, operating frequency, pin load conditions, temperature and program instruction type, etc., can all exert an influence on the measured values.

Operating Voltage Characteristics

Ta=-40°C~105°C

Symbol	Parameter	Test Conditions	Min.	Typ.	Max.	Unit
V_{DD}	Operating Voltage – HIRC	f_{SYS}=8MHz	2.2	—	5.5	V
	Operating Voltage – LIRC	f_{SYS}=32kHz	2.2	—	5.5	V

CAN Bus Operating Voltage Characteristics

Ta=-40°C~105°C

Symbol	Parameter	Test Conditions	Min.	Typ.	Max.	Unit
CAN_V_{DD}	Operating Voltage – HXT	CAN_f_{SYS}=f_{HXT}=8MHz	3.0	—	5.5	V
		CAN_f_{SYS}=f_{HXT}=16MHz	3.0	—	5.5	V
		CAN_f_{SYS}=f_{HXT}=24MHz	4.5	—	5.5	V

Operating Current Characteristics

Ta=-40°C~105°C

Symbol	Operating Mode	Test Conditions		Min.	Typ.	Max.	Unit
		V_{DD}	Conditions				
I_{DD}	SLOW Mode – LIRC	2.2V	f_{SYS}=32kHz, OPA0/1 always enable, PGA always enable, Bandgap always enable	—	1200	1600	μA
		3V		—	1300	1800	
		5V		—	1600	2200	
	FAST Mode – HIRC	2.2V	f_{SYS}=8MHz, OPA0/1 always enable, PGA always enable, Bandgap always enable	—	1.6	2.0	mA
		3V		—	2.1	3.0	
		5V		—	3.2	4.6	

Note: When using the characteristic table data, the following notes should be taken into consideration:

1. Any digital inputs are setup in a non floating condition.
2. All measurements are taken under conditions of no load and with all peripherals in an off state.
3. There are no DC current paths.
4. All Operating Current values are measured using a continuous NOP instruction program loop.

CAN Bus Operating Current Characteristics

Ta=-40°C~105°C

Symbol	Parameter	Test Conditions		Min.	Typ.	Max.	Unit
		CAN_V_{DD}	Conditions				
CAN_I_{DD}	Operating Current – HXT	3V	No load, all peripherals off, CAN_f_{SYS}=f_{HXT}=8MHz	—	8	13	mA
		5V		—	13	18	mA
		3V	No load, all peripherals off, CAN_f_{SYS}=f_{HXT}=16MHz	—	15	20	mA
		5V		—	25	30	mA
		4.5V	No load, all peripherals off, CAN_f_{SYS}=f_{HXT}=24MHz	—	23	28	mA
		5V		—	38	41	mA

Standby Current Characteristics

Ta=25°C, unless otherwise specified

Symbol	Standby Mode	Test Conditions		Min.	Typ.	Max.	Max. @105°C	Unit
		V_{DD}	Conditions					
I_{STB}	SLEEP Mode	2.2V	WDT off, OPA0/1 always enable, PGA always enable, Bandgap always enable	—	1.6	1.8	2.2	mA
		3V		—	1.7	1.9	2.3	
		5V		—	1.8	2.0	2.4	
	IDLE0 Mode – LIRC	2.2V	f_{SUB} on, OPA0/1 always enable, PGA always enable, Bandgap always enable	—	1.6	1.8	2.2	
		3V		—	1.7	1.9	2.3	
		5V		—	1.8	2.0	2.4	
	IDLE1 Mode – HIRC	2.2V	f_{SUB} on, f_{SYS}=8MHz, OPA0/1 always enable, PGA always enable, Bandgap always enable	—	2.2	2.4	2.8	
		3V		—	2.4	2.6	3.0	
		5V		—	2.6	2.8	3.2	

Note: When using the characteristic table data, the following notes should be taken into consideration:
1. Any digital inputs are setup in a non floating condition.
2. All measurements are taken under conditions of no load and with all peripherals in an off state.
3. There are no DC current paths.
4. All Standby Current values are taken after a HALT instruction execution thus stopping all instruction execution.

CAN Bus Standby Current Characteristics

Ta=25°C, unless otherwise specified

Symbol	Parameter	Test Conditions		Min.	Typ.	Max.	Max. @105°C	Unit
		CAN_V_{DD}	Conditions					
CAN_I_{STB}	Standby Current – SLEEP Mode	3V	No load, all peripherals off	—	—	1	1	μA
		5V		—	—	2	2	μA
	Standby Current – IDLE Mode	3V	No load, all peripherals off, CAN_f_{SYS}=f_{HXT}=8MHz	—	—	1	1	mA
		5V		—	—	2.0	2.0	mA
		3V	No load, all peripherals off, CAN_f_{SYS}=f_{HXT}=16MHz	—	—	1.2	1.2	mA
		5V		—	—	2.2	2.2	mA
		4.5V	No load, all peripherals off, CAN_f_{SYS}=f_{HXT}=24MHz	—	—	3.0	3.0	mA
		5V		—	—	3.8	3.8	mA

A.C. Characteristics

For data in the following tables, note that factors such as oscillator type, operating voltage, operating frequency and temperature etc., can all exert an influence on the measured values.

Internal High Speed Oscillator – HIRC – Frequency Accuracy

During the program writing operation the writer will trim the HIRC oscillator at a user selected HIRC frequency and user selected voltage of either 3V or 5V.

Symbol	Parameter	Test Conditions		Min.	Typ.	Max.	Unit
		V_{DD}	Temp.				
f_{HIRC}	8MHz Writer Trimmed HIRC Frequency	3V/5V	25°C	-1%	8	+1%	MHz
			-40°C~105°C	-2.5%	8	+2.5%	
		2.2V~5.5V	25°C	-2.5%	8	+2.5%	
			-40°C~105°C	-3%	8	+3%	

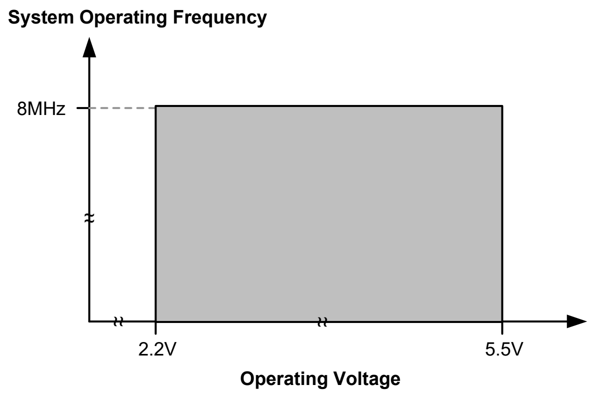
Note: 1. The 3V/5V values for V_{DD} are provided as these are the two selectable fixed voltages at which the HIRC frequency is trimmed by the writer.

2. The row below the 3V/5V trim voltage row is provided to show the values for the full V_{DD} range operating voltage. It is recommended that the trim voltage is fixed at 3V for application voltage ranges from 2.2V to 3.6V and fixed at 5V for application voltage ranges from 3.3V to 5.5V.

Internal Low Speed Oscillator Characteristics – LIRC

Symbol	Parameter	Test Conditions		Min.	Typ.	Max.	Unit
		V_{DD}	Temp.				
f_{LIRC}	LIRC Frequency	2.2V~5.5V	25°C	-10%	32	+10%	kHz
			-40°C~105°C	-50%	32	+60%	
t_{START}	LIRC Start Up Time	—	-40°C~105°C	—	—	500	μs

Operating Frequency Characteristic Curves



CAN Bus System Frequency Characteristics

Ta=-40°C~105°C

Symbol	Parameter	Test Conditions		Min.	Typ.	Max.	Unit
		CAN_V_{DD}	Conditions				
CAN_f_{SYS}	System Clock – HXT	3.0V~5.5V	CAN_f_{SYS}=f_{HXT}=8MHz	—	8	—	MHz
		3.0V~5.5V	CAN_f_{SYS}=f_{HXT}=16MHz	—	16	—	MHz
		4.5V~5.5V	CAN_f_{SYS}=f_{HXT}=24MHz	—	24	—	MHz

System Start Up Time Characteristics

Ta=-40°C~105°C

Symbol	Parameter	Test Conditions		Min.	Typ.	Max.	Unit
		V_{DD}	Conditions				
t_{SST}	System Start-up Time (Wake-up from Conditions where f_{SYS} is off)	—	f_{SYS}=f_H~f_H/64, f_H=f_{HIRC}	—	16	—	t_{HIRC}
			f_{SYS}=f_{SUB}=f_{LIRC}	—	2	—	t_{LIRC}
	System Start-up Time (Wake-up from Conditions where f_{SYS} is on)	—	f_{SYS}=f_H~f_H/64, f_H=f_{HIRC}	—	2	—	t_H
			f_{SYS}=f_{SUB}=f_{LIRC}	—	2	—	t_{SUB}
	System Speed Switch Time (FAST to Slow Mode or SLOW to FAST Mode)	—	f_{HIRC} switches from off → on	—	16	—	t_{HIRC}

Symbol	Parameter	Test Conditions		Min.	Typ.	Max.	Unit
		V_{DD}	Conditions				
t_{RSTD}	System Reset Delay Time (Reset Source from Power-on Reset or LVR Hardware Reset) Note: The fast power on mode is enabled.	—	RR_{POR}=5V/ms (Start the time when the V_{DD} reaches the minimum operating voltage and the reset pin is high)	0.6	0.8	1.0	ms
	System Reset Delay Time (LVRC/WDTC/RSTC Software Reset) Note: The fast power on mode is enabled.	—	—				
	System Reset Delay Time (Reset Source from WDT Overflow or $\overline{RES}$ Pin Reset) Note: The fast power on mode is enabled.	—	—				
	System Reset Delay Time (Reset Source from Power-on Reset or LVR Hardware Reset) Note: The fast power on mode is disabled.	—	RR_{POR}=5V/ms	14	16	18	ms
	System Reset Delay Time (LVRC/WDTC/RSTC Software Reset) Note: The fast power on mode is disabled.	—	—				
	System Reset Delay Time (Reset Source from WDT Overflow or $\overline{RES}$ Pin Reset) Note: The fast power on mode is disabled.	—	—				
t_{SRESET}	Minimum Software Reset Width to Reset	—	—	45	90	120	μs

Note: 1. For the System Start-up time values, whether f_{SYS} is on or off depends upon the mode type and the chosen f_{SYS} system oscillator. Details are provided in the System Operating Modes section.
2. The time units, shown by the symbols t_{HIRC} etc. are the inverse of the corresponding frequency values as provided in the frequency tables. For example, $t_{HIRC}=1/f_{HIRC}$, $t_{SYS}=1/f_{SYS}$ etc.
3. If the LIRC is used as the system clock and if it is off when in the SLEEP Mode, then an additional LIRC start up time, t_{START}, as provided in the LIRC frequency table, must be added to the t_{SST} time in the table above.
4. The System Speed Switch Time is effectively the time taken for the newly activated oscillator to start up.

CAN Bus Timing Characteristics

Ta=-40°C~105°C

Symbol	Parameter	Test Conditions		Min.	Typ.	Max.	Unit
		CAN_V_{DD}	Conditions				
CAN_t_{RES}	External Reset Minimum Low Pulse Width	—	—	10	—	—	μs
CAN_$t_{START(HXT)}$	HXT Oscillator Start-up Time	3V	CAN_$f_{SYS}=f_{HXT}$=16MHz	—	—	25	ms
		5V	CAN_$f_{SYS}=f_{HXT}$=16MHz	—	—	10	ms
CAN_t_{CKOSST}	CLKOUT Start-up Timer Period (Wake up from condition where HXT is off)	—	CAN_$f_{SYS}=f_{HXT}$	—	1024	—	CAN_t_{SYS}
CAN_t_{SOF}	SOF Signal Width	—	CAN_$f_{SYS}=f_{HXT}$=24MHz SOFT[2:0]=101	9.6	10.6	11.6	μs

Input/Output Characteristics

Ta=-40°C~105°C

Symbol	Parameter	Test Conditions		Min.	Typ.	Max.	Unit
		V_{DD}	Conditions				
V_{IL}	Input Low Voltage for I/O Ports	5V	—	0	—	1.5	V
		—	—	0	—	$0.2V_{DD}$	
	Input Low Voltage for $\overline{RES}$ Pin	5V	—	0	—	2.0	V
		—	V_{DD}≥2.7V	0	—	$0.4V_{DD}$	
		—	2.2V≤V_{DD}<2.7V	0	—	$0.3V_{DD}$	
V_{IH}	Input High Voltage for I/O Ports	5V	—	3.5	—	5.0	V
		—	—	$0.8V_{DD}$	—	V_{DD}	
	Input High Voltage for $\overline{RES}$ Pin	5V	—	4.5	—	5.0	V
		—	—	$0.9V_{DD}$	—	V_{DD}	
I_{OL}	Sink Current for I/O Ports	3V	V_{OL}=0.1V_{DD}	16	32	—	mA
		5V		32	65	—	
I_{OH}	Source Current for I/O Ports	3V	V_{OH}=0.9V_{DD}	-4	-8	—	mA
		5V		-8	-16	—	
R_{PH}	Pull-high Resistance for I/O Ports Except PA2 (1)	3V	—	20	60	100	kΩ
		5V	—	10	30	50	
	Pull-high Resistance for Reset Pin	3V	—	6.67	15	23	kΩ
		5V	—	3.5	7.5	12	
I_{LEAK}	Input Leakage Current for I/O Ports	5V	V_{IN}=V_{DD} or V_{IN}=V_{SS}	—	—	±1	μA
t_{INT}	External Interrupt Minimum Pulse Width	—	—	10	—	—	μs
t_{RES}	External Reset Pin Minimum Pulse Width	—	—	10	—	—	μs
t_{TPI}	TM Input Pin Minimum Pulse Width	—	—	0.3	—	—	μs
t_{TCK}	STPnI Input Pin Minimum Pulse Width	—	—	0.3	—	—	μs
f_{TMCLK}	STMn Maximum Timer Clock Source Frequency	5V	—	—	—	1	f_{SYS}
t_{CPW}	STMn Minimum Capture Pulse Width	—	—	t_{CPW}(2)	—	—	μs

Note: 1. The R_{PH} internal pull-high resistance value is calculated by connecting to ground and enabling input pin with a pull-high resistor and then measuring the pin current at the specified supply voltage level. Dividing the voltage by this measured current provides the R_{PH} value.

2. $t_{TMCLK}=1/f_{TMCLK}$.
$t_{CPW}=\max(2\times t_{TMCLK}, t_{TPI})$
Ex1: If f_{TMCLK}=8MHz, t_{TPI}=0.3μs, then t_{CPW}=max(0.25μs, 0.3μs)=0.3μs
Ex2: If f_{TMCLK}=4MHz, t_{TPI}=0.3μs, then t_{CPW}=max(0.5μs, 0.3μs)=0.5μs

CAN Bus Input/Output Characteristics

Ta=-40°C~105°C

Symbol	Parameter	Test Conditions		Min.	Typ.	Max.	Unit
		CAN_V_{DD}	Conditions				
CAN_V_{IL}	Input Low Voltage for CANRX Pin	—	—	0	—	0.2CAN_V_{DD}	V
CAN_V_{IH}	Input High Voltage for CANRX Pin	—	—	0.8CAN_V_{DD}	—	CAN_V_{DD}	V
CAN_I_{OL}	Sink Current for CANTX Pin	3V	CAN_V_{OL}=0.1CAN_V_{DD}	4	8	—	mA
		5V		10	20	—	mA
CAN_I_{OH}	Source Current for CANTX Pin	3V	CAN_V_{OH}=0.9CAN_V_{DD}	-2	-4	—	mA
		5V		-5	-10	—	mA

Memory Characteristics

Ta=-40°C~105°C, unless otherwise specified

Symbol	Parameter	Test Conditions		Min.	Typ.	Max.	Unit
		V_{DD}	Conditions				
Flash Program Memory							
V_{DD}	V_{DD} for Read	—	—	2.2	—	5.5	V
	V_{DD} for Erase/Write	—	—	2.2	—	5.5	V
t_{FER}	IAP Erase Time	—	FWERTS=0	—	3.2	4.9	ms
		—	FWERTS=1	—	3.7	5.6	
t_{FWR}	IAP Write Time	—	FWERTS=0	—	2.2	3.5	ms
		—	FWERTS=1	—	3.0	4.6	
I_{DDPGM}	Programming/Erase Current on V_{DD}	—	—	—	—	5.0	mA
E_P	Cell Endurance	—	—	100K	—	—	E/W
t_{RETD}	ROM Data Retention Time	—	Ta=25°C	—	40	—	Year
t_{ACTV}	ROM Activation Time – Wake-up from IDLE/SLEEP Mode	—	—	32	—	64	μs
Data EEPROM Memory							
V_{DD}	V_{DD} for Read/Write	—	—	2.2	—	5.5	V
t_{EERD}	EEPROM Read Time	—	—	—	—	4	t_{SYS}
t_{EEER}	EEPROM Erase Time	—	EWERTS=0	—	3.2	4.7	ms
		—	EWERTS=1	—	3.7	5.4	
t_{EEWR}	EEPROM Write Time (Byte Mode)	—	EWERTS=0	—	5.4	9.0	ms
		—	EWERTS=1	—	6.7	10.0	
	EEPROM Write Time (Page Mode)	—	EWERTS=0	—	2.2	4.5	
		—	EWERTS=1	—	3.0	5.5	
E_P	Cell Endurance	—	—	100K	—	—	E/W
t_{RETD}	Data Retention Time	—	Ta=25°C	—	40	—	Year
RAM Data Memory							
V_{DR}	RAM Data Retention Voltage	—	—	2.2	—	—	V

Note: 1. "E/W" means Erase/Write times.

2. The ROM activation time t_{ACTV} should be added when calculating the total system start-up time of a wake-up from the IDLE/SLEEP mode.

LVR Electrical Characteristics

Ta=-40°C~105°C

Symbol	Parameter	Test Conditions		Min.	Typ.	Max.	Unit
		V_{DD}	Conditions				
V_{DD}	Operating Voltage	—	—	2.2	—	5.5	V
V_{LVR}	Low Voltage Reset Voltage	—	LVR enable, voltage select 2.1V	-3%	2.1	+3%	V
I_{LVROP}	LVR Operating Current	3V	LVR enable, V_{LVR}=2.1V	—	—	10	μA
		5V		—	10	15	
t_{LVR}	Minimum Low Voltage Width to Reset	—	TLVR[1:0]=00B	120	240	480	μs
			TLVR[1:0]=01B	0.5	1.0	2.0	ms
			TLVR[1:0]=10B	1	2	4	
			TLVR[1:0]=11B	2	4	8	
I_{LVR}	Additional Current for LVR Enable	5V	—	—	—	14	μA

A/D Converter Electrical Characteristics

Ta=-40°C~105°C

Symbol	Parameter	Test Conditions		Min.	Typ.	Max.	Unit
		V_{DD}	Conditions				
V_{DD}	A/D Converter Operating Voltage	—	—	2.2	—	5.5	V
V_{ADI}	A/D Converter Input Voltage	—	—	0	—	V_{REF}	V
V_{REF}	A/D Converter Reference Voltage	—	—	2.2	—	V_{DD}	V
N_R	A/D Converter Resolution	—	—	—	—	12	Bit
DNL	A/D Converter Differential Non-linearity	5V	SAINS[3:0]=0000B, SAVRS[1:0]=01B, V_{REF}=V_{DD}, t_{ADCK}=0.5μs	-3	—	3	LSB
			SAINS[3:0]=0000B, SAVRS[1:0]=01B, V_{REF}=V_{DD}, t_{ADCK}=10μs				
INL	A/D Converter Integral Non-linearity	5V	SAINS[3:0]=0000B, SAVRS[1:0]=01B, V_{REF}=V_{DD}, t_{ADCK}=0.5μs	-4	—	4	LSB
			SAINS[3:0]=0000B, SAVRS[1:0]=01B, V_{REF}=V_{DD}, t_{ADCK}=10μs				
I_{ADC}	Additional Current for A/D Converter Enable	5V	No load, t_{ADCK}=0.5μs	—	850	1000	μA
t_{ADCK}	A/D Converter Clock Period	—	4.2V≤V_{DD}≤5.5V	0.5	—	10.0	μs
t_{ON2ST}	A/D Converter On-to-Start Time	—	—	4	—	—	μs
t_{ADS}	A/D Sampling Time	—	—	—	4	—	t_{ADCK}
t_{ADC}	A/D Conversion Time (Including A/D Sample and Hold Time)	—	—	—	16	—	t_{ADCK}
I_{PGA}	Additional Current for PGA Enable	5V	No load	—	900	1800	μA
V_{OR}	PGA Maximum Output Voltage Range	5V	—	V_{SS} +0.1	—	V_{DD} -0.1	V
V_{VR}	Fix Voltage Output of PGA	—	V_{DD}=4.2V~5.5V V_{RI}=V_{BGREF}	-1%	4	+1%	V
V_{OS_PGA}	PGA Input Offset Voltage	5V	—	-15	—	+15	mV

Reference Voltage Characteristics

Ta=-40°C~105°C, unless otherwise specified

Symbol	Parameter	Test Conditions		Min.	Typ.	Max.	Unit
		V_{DD}	Conditions				
V_{DD}	Operating Voltage	—	—	2.2	–	5.5	V
V_{BGREF}	Bandgap Reference Voltage	2.2V~5.5V	—	-1%	1.2	+1%	V
I_{BGREF}	Operating Current	5.5V	—	—	25	35	μA
PSRR	Power Supply Rejection Ratio	—	Ta=25°C, V_{RIPPLE}=1V_{P-P}, f_{RIPPLE}=100Hz	75	—	—	dB
En	Output Noise	—	Ta=25°C, no load current, f=0.1Hz~10Hz	—	300	—	μV_{RMS}
t_{START}	Startup Time	2.2V~5.5V	Ta=25°C	—	—	400	μs

Note: The V_{BGREF} voltage is used as the A/D converter PGA input.

Operational Amplifier Electrical Characteristics

Ta=-40°C~105°C, unless otherwise specified

Symbol	Parameter	Test Conditions		Min.	Typ.	Max.	Unit
		V_{DD}	Conditions				
I_{OPA}	Additional Current for OPA	5V	No load	—	300	600	μA
I_{PGA}	OPA2 PGA Current	5V	Gain=20/40	—	320	630	μA
V_{OS}	Input Offset Voltage	5V	Ta=25°C, OPA0/1 without calibration	-7	5	7	mV
			OPA2 without calibration (OOF [5:0]=100000B)	-15	—	15	
			OPA2 with calibration	-2	—	2	
V_{CM}	Common Mode Voltage Range	5V	—	V_{SS}	—	V_{DD} -1.4	V
I_{SC}	Output Short Circuit Current	5V	RLOAD=5.1Ω	±10	±20	—	mA
Ga	OPA2 PGA Gain Accuracy	5V	Relative gain	-5	—	5	%

14-bit D/A Converter Electrical Characteristics

Ta=-40°C~105°C

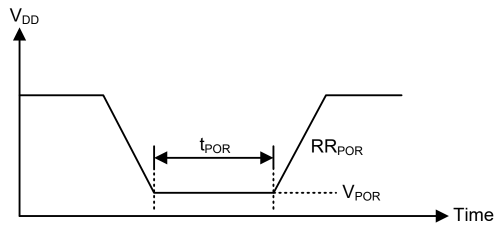
Symbol	Parameter	Test Conditions		Min.	Typ.	Max.	Unit
		V_{DD}	Conditions				
V_{DACO}	Output Voltage Range	—	—	V_{SS}	—	DAVREF	V
DAVREF	Reference Voltage	—	—	2	—	V_{DD}	V
I_{DAC}	Additional Current for DAC Enable	5V	—	—	600	800	μA
t_{ST}	Settling Time	5V	C_{LOAD}=50pF	—	—	5	μs
DNL	Differential Non-linearity	5V	V_{REF}=V_{DD}, DA[13:0]=0000h~1fffh$^{(Note)}$	—	±4	±10	LSB
			V_{REF}=V_{DD}, DA[13:0]=2000h~3fffh$^{(Note)}$	—	±4	±17	LSB
			V_{REF}=V_{RO}, full range	—	—	±18	LSB
INL	Integral Non-linearity	5V	V_{REF}=V_{DD}, DA[13:0]=0000h~1fffh$^{(Note)}$	—	±5	±12	LSB
			V_{REF}=V_{DD}, DA[13:0]=2000h~3fffh$^{(Note)}$	—	±5	±18	LSB
			V_{REF}=V_{RO}, full range	—	—	±18	LSB
Ro	D/A Converter 0/1 R2R Output Resistor	5V	—	—	13	—	kΩ

Note: Keeping 1LSB=$V_{REF}/2^{14}$ unchanged, consider the interval code as a new 14-bit DAC to calculate the DNL/INL.

Power-on Reset Characteristics

Ta=-40°C~105°C

Symbol	Parameter	Test Conditions		Min.	Typ.	Max.	Unit
		V_{DD}	Conditions				
V_{POR}	V_{DD} Start Voltage to Ensure Power-on Reset	—	—	—	—	100	mV
RR_{POR}	V_{DD} Rising Rate to Ensure Power-on Reset	—	—	0.035	—	—	V/ms
t_{POR}	Minimum Time for V_{DD} Stays at V_{POR} to Ensure Power-on Reset	—	—	1	—	—	ms



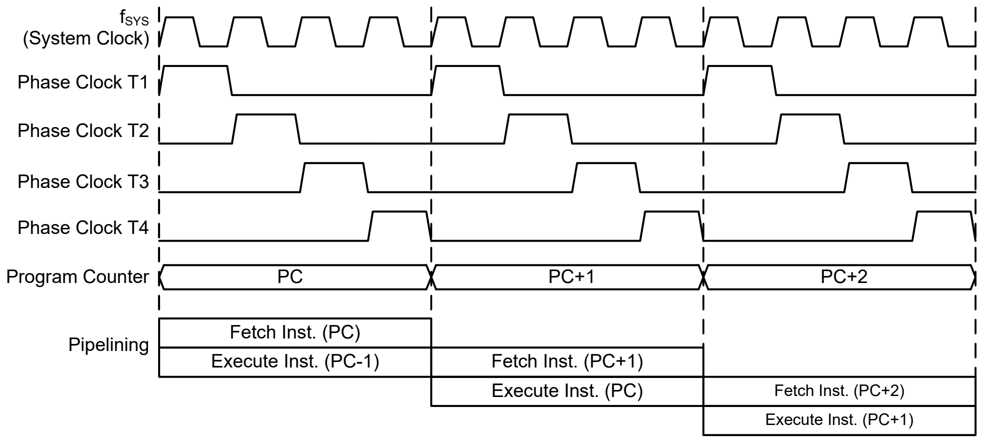
System Architecture

A key factor in the high-performance features of the Holtek range of microcontrollers is attributed to their internal system architecture. The device takes advantage of the usual features found within RISC microcontrollers providing increased speed of operation and enhanced performance. The pipelining scheme is implemented in such a way that instruction fetching and instruction execution are overlapped, hence instructions are effectively executed in one or two cycles for most of the standard or extended instructions respectively. The exceptions to these are branch or call instructions which need one more cycle. An 8-bit wide ALU is used in practically all instruction set operations, which carries out arithmetic operations, logic operations, rotation, increment, decrement, branch decisions, etc. The internal data path is simplified by moving data through the Accumulator and the ALU. Certain internal registers are implemented in the Data Memory and can be directly or indirectly addressed. The simple addressing methods of these registers along with additional architectural features ensure that a minimum of external components is required to provide a functional I/O and A/D control system with maximum reliability and flexibility. This makes the device suitable for affordable, high-volume production for controller applications.

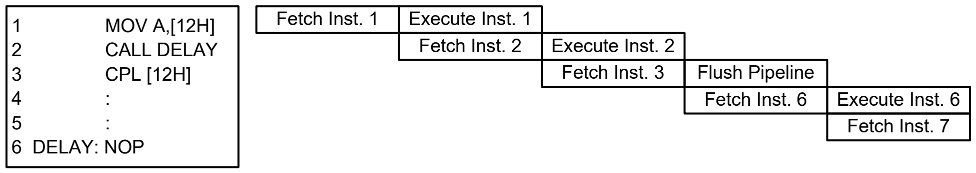
Clocking and Pipelining

The main system clock, derived from either an HIRC or LIRC oscillator is subdivided into four internally generated non-overlapping clocks, T1~T4. The Program Counter is incremented at the beginning of the T1 clock during which time a new instruction is fetched. The remaining T2~T4 clocks carry out the decoding and execution functions. In this way, one T1~T4 clock cycle forms one instruction cycle. Although the fetching and execution of instructions takes place in consecutive instruction cycles, the pipelining structure of the microcontroller ensures that instructions are effectively executed in one instruction cycle. The exception to this are instructions where the contents of the Program Counter are changed, such as subroutine calls or jumps, in which case the instruction will take one more instruction cycle to execute.

For instructions involving branches, such as jump or call instructions, two machine cycles are required to complete instruction execution. An extra cycle is required as the program takes one cycle to first obtain the actual jump or call address and then another cycle to actually execute the branch. The requirement for this extra cycle should be taken into account by programmers in timing sensitive applications.



System Clocking and Pipelining



```
1            MOV A,[12H]
2            CALL DELAY
3            CPL [12H]
4            :
5            :
6  DELAY: NOP
```

Instruction Fetching

Program Counter

During program execution, the Program Counter is used to keep track of the address of the next instruction to be executed. It is automatically incremented by one each time an instruction is executed except for instructions, such as "JMP" or "CALL" that demand a jump to a non-consecutive Program Memory address. Only the lower 8 bits, known as the Program Counter Low Register, are directly addressable by the application program.

When executing instructions requiring jumps to non-consecutive addresses such as a jump instruction, a subroutine call, interrupt or reset, etc., the microcontroller manages program control by loading the required address into the Program Counter. For conditional skip instructions, once the condition has been met, the next instruction, which has already been fetched during the present instruction execution, is discarded and a dummy cycle takes its place while the correct instruction is obtained.

Program Counter	
High Byte	**Low Byte (PCL)**
PC12~PC8	PCL7~PCL0

Program Counter

The lower byte of the Program Counter, known as the Program Counter Low register or PCL, is available for program control and is a readable and writeable register. By transferring data directly into this register, a short program jump can be executed directly; however, as only this low byte is available for manipulation, the jumps are limited to the present page of memory that is 256 locations. When such program jumps are executed it should also be noted that a dummy cycle will be inserted. Manipulating the PCL register may cause program branching, so an extra cycle is needed to pre-fetch.

Program Counter Read Registers

The Program Counter read registers are a read only register pair for reading the program counter value which indicates the current program execution address. Read the low byte register first then the high byte register. Reading the low byte register, PCRL, will read the low byte data of the current program execution address, and place the high byte data of the program counter into the 8-bit PCRH buffer. Then reading the PCRH register will read the corresponding data from the 8-bit PCRH buffer.

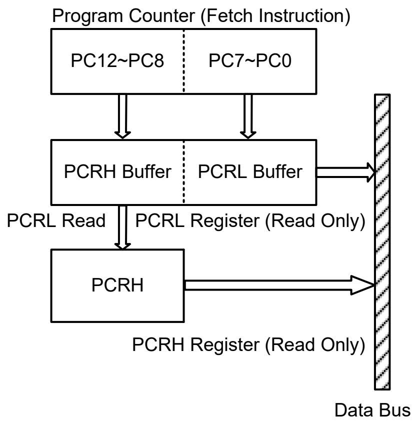
The following example shows how to read the current program execution address. When the current program execution address is 123H, the steps to execute the instructions are as follows:

(1) MOV A, PCRL → the ACC value is 23H, and the PCRH value is 01H;

MOV A, PCRH → the ACC value is 01H.

(2) LMOV A, PCRL → the ACC value is 23H, and the PCRH value is 01H;

LMOV A, PCRH → the ACC value is 01H.


Program Counter (Fetch Instruction)
PC12~PC8
PC7~PC0
PCRH Buffer
PCRL Buffer
PCRL Read
PCRL Register (Read Only)
PCRH
PCRH Register (Read Only)
Data Bus

• **PCRL Register**

Bit	7	6	5	4	3	2	1	0
Name	D7	D6	D5	D4	D3	D2	D1	D0
R/W	R	R	R	R	R	R	R	R
POR	0	0	0	0	0	0	0	0

Bit 7~0 **D7~D0**: Program Counter Read low byte register bit 7 ~ bit 0

• **PCRH Register**

Bit	7	6	5	4	3	2	1	0
Name	—	—	—	D12	D11	D10	D9	D8
R/W	—	—	—	R	R	R	R	R
POR	—	—	—	0	0	0	0	0

Bit 7~5 Unimplemented, read as “0”

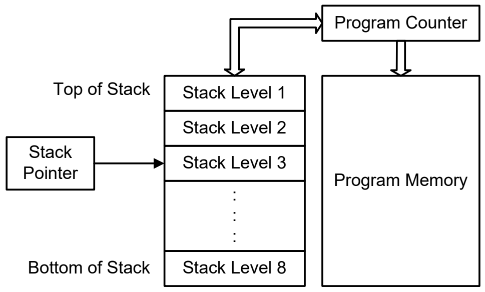
Bit 4~0 **D12~D8**: Program Counter Read high byte register bit 4 ~ bit 0

Stack

This is a special part of the memory which is used to save the contents of the Program Counter only. The stack is organized into 8 levels and is neither part of the data nor part of the program space, and is neither readable nor writeable. The activated level is indexed by the Stack Pointer, STKPTR[3:0]. At a subroutine call or interrupt acknowledge signal, the contents of the Program Counter are pushed onto the stack. At the end of a subroutine or an interrupt routine, signaled by a return instruction, RET or RETI, the Program Counter is restored to its previous value from the stack. After a device reset, the Stack Pointer will point to the top of the stack.

If the stack is full and an enabled interrupt takes place, the interrupt request flag will be recorded but the acknowledge signal will be inhibited. When the Stack Pointer is decremented, by RET or RETI, the interrupt will be serviced. This feature prevents stack overflow allowing the programmer to use the structure more easily. However, when the stack is full, a CALL subroutine instruction can still be executed which will result in a stack overflow. Precautions should be taken to avoid such cases which might cause unpredictable program branching.

If the stack is overflow, the first Program Counter save in the stack will be lost.



- **STKPTR Register**

Bit	7	6	5	4	3	2	1	0
Name	OSF	—	—	—	D3	D2	D1	D0
R/W	R/W	—	—	—	R	R	R	R
POR	0	—	—	—	0	0	0	0

Bit 7 **OSF**: Stack overflow flag
0: No stack overflow occurred
1: Stack overflow occurred
When the stack is full and a CALL instruction is executed or when the stack is empty and a RET instruction is executed, the OSF bit will be set high. The OSF bit is cleared only by software and cannot be reset automatically by hardware.

Bit 6~4 Unimplemented, read as "0"

Bit 3~0 **D3~D0**: Stack pointer register bit 3 ~ bit 0

The following example shows how the Stack Pointer and Stack Overflow Flag change when program branching conditions occur.

(1) When the CALL subroutine instruction is executed 9 times continuously and the RET instruction is not executed during the period, the corresponding changes of the STKPTR[3:0] and OSF bits are as follows:

CALL Execution Times	0	1	2	3	4	5	6	7	8	9
STKPTR[3:0] Bit Value	0	1	2	3	4	5	6	7	0	1
OSF Bit Value	0	0	0	0	0	0	0	0	0	1

(2) When the OSF bit is set high and not cleared, it will remain high no matter how many times the RET instruction is executed.

(3) When the stack is empty, the RET instruction is executed 8 times continuously, the corresponding changes of the STKPTR[3:0] and OSF bits are as follows:

RET Execution Times	0	1	2	3	4	5	6	7	8
STKPTR[3:0] Bit Value	0	7	6	5	4	3	2	1	0
OSF Bit Value	0	1	1	1	1	1	1	1	1

Arithmetic and Logic Unit – ALU

The arithmetic-logic unit or ALU is a critical area of the microcontroller that carries out arithmetic and logic operations of the instruction set. Connected to the main microcontroller data bus, the ALU receives related instruction codes and performs the required arithmetic or logical operations after which the result will be placed in the specified register. As these ALU calculation or operations may result in carry, borrow or other status changes, the status register will be correspondingly updated to reflect these changes. The ALU supports the following functions:

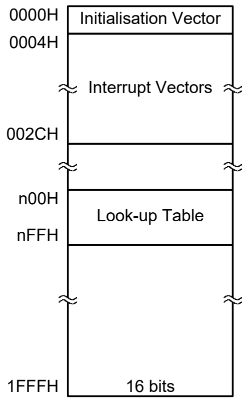
- Arithmetic operations:
  ADD, ADDM, ADC, ADCM, SUB, SUBM, SBC, SBCM, DAA,
  LADD, LADDM, LADC, LADCM, LSUB, LSUBM, LSBC, LSBCM, LDAA
- Logic operations:
  AND, OR, XOR, ANDM, ORM, XORM, CPL, CPLA,
  LAND, LOR, LXOR, LANDM, LORM, LXORM, LCPL, LCPLA
- Rotation:
  RRA, RR, RRCA, RRC, RLA, RL, RLCA, RLC,
  LRRA, LRR, LRRCA, LRRC, LRLA, LRL, LRLCA, LRLC
- Increment and Decrement:
  INCA, INC, DECA, DEC,
  LINCA, LINC, LDECA, LDEC
- Branch decision:
  JMP, SZ, SZA, SNZ, SIZ, SDZ, SIZA, SDZA, CALL, RET, RETI,
  LSZ, LSZA, LSNZ, LSIZ, LSDZ, LSIZA, LSDZA

Flash Program Memory

The Program Memory is the location where the user code or program is stored. For this device the Program Memory is Flash type, which means it can be programmed and re-programmed a large number of times, allowing the user the convenience of code modification on the same device. By using the appropriate programming tools, the Flash device offers users the flexibility to conveniently debug and develop their applications while also offering a means of field programming and updating.

Structure

The Program Memory has a capacity of 8K×16 bits. The Program Memory is addressed by the Program Counter and also contains data, table information and interrupt entries. Table data, which can be setup in any location within the Program Memory, is addressed by a separate table pointer registers.



Program Memory Structure

Special Vectors

Within the Program Memory, certain locations are reserved for the reset and interrupts. The location 0000H is reserved for use by the device reset for program initialisation. After a device reset is initiated, the program will jump to this location and begin execution.

Look-up Table

Any location within the Program Memory can be defined as a look-up table where programmers can store fixed data. To use the look-up table, the table pointer must first be setup by placing the address of the look up data to be retrieved in the table pointer registers, TBLP and TBHP. These registers define the total address of the look-up table.

After setting up the table pointer, the table data can be retrieved from the Program Memory using the "TABRD [m]" or "TABRDL [m]" instructions respectively when the memory [m] is located in sector 0. If the memory [m] is located in other sectors except sector 0, the data can be retrieved from the program memory using the corresponding extended table read instruction such as "LTABRD [m]" or "LTABRDL [m]" respectively. When the instruction is executed, the lower order table byte from the Program Memory will be transferred to the user defined Data Memory register [m] as specified in the instruction. The higher order table data byte from the Program Memory will be transferred to the TBLH special register.

The accompanying diagram illustrates the addressing data flow of the look-up table.

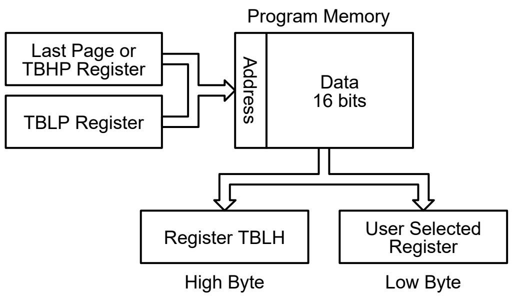


Table Program Example

The accompanying example shows how the table pointer and table data is defined and retrieved from the device. This example uses raw table data located in the last page which is stored there using the ORG statement. The value at this ORG statement is "1F00H" which refers to the start address of the last page within the 8K words Program Memory. The table pointer low byte register is setup here to have an initial value of "06H". This will ensure that the first data read from the data table will be at the Program Memory address “1F06H” or 6 locations after the start of the last page. Note that the value for the table pointer is referenced to the specific address pointed by TBHP and TBLP if the “TABRD [m]” or “LTABRD [m]” instruction is being used. The high byte of the table data which in this case is equal to zero will be transferred to the TBLH register automatically when the “TABRD [m]” or “LTABRD [m]” instruction is executed.

Because the TBLH register is a read/write register and can be restored, care should be taken to ensure its protection if both the main routine and Interrupt Service Routine use table read instructions. If using the table read instructions, the Interrupt Service Routines may change the value of the TBLH and subsequently cause errors if used again by the main routine. As a rule it is recommended that simultaneous use of the table read instructions should be avoided. However, in situations where simultaneous use cannot be avoided, the interrupts should be disabled prior to the execution of any main routine table-read instructions. Note that all table related instructions require two instruction cycles to complete their operation.

Table Read Program Example

```
tempreg1 db?      ; temporary register #1
tempreg2 db?      ; temporary register #2
:
:
mov a,06H         ; initialise table pointer - note that this address is referenced
mov tblp,a        ; to the last page or the page that tbhp pointed
mov a,1FH         ; initialise high table pointer
mov tbhp,a        ; it is not necessary to set tbhp if executing tabrdl or ltabrdl
:
:
tabrd tempreg1    ; transfers value in table referenced by table pointer data at
                  ; program memory address “1F06H” transferred to tempreg1 and TBLH
dec tblp          ; reduce value of table pointer by one
tabrd tempreg2    ; transfers value in table referenced by table pointer data at
                  ; program memory address “1F05H” transferred to tempreg2 and TBLH
                  ; in this example the data “1AH” is transferred to tempreg1 and
                  ; data “0FH” to tempreg2 the value “00H” will be
                  ; transferred to the high byte register TBLH
:
:
org 1F00H         ; sets initial address of last page
dc  00Ah,00Bh,00Ch,00Dh,00Eh,00Fh,01Ah,01Bh
```

In Circuit Programming – ICP

The provision of Flash type Program Memory provides the user with a means of convenient and easy upgrades and modifications to their programs on the same device.

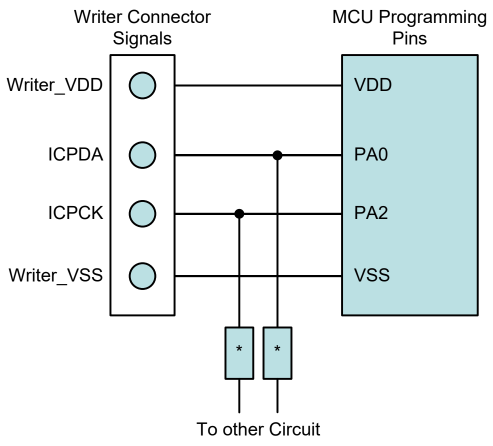
As an additional convenience, Holtek has provided a means of programming the microcontroller in-circuit using a 4-pin interface. This provides manufacturers with the possibility of manufacturing their circuit boards complete with a programmed or un-programmed microcontroller, and then programming or upgrading the program at a later stage. This enables product manufacturers to easily keep their manufactured products supplied with the latest program releases without removal and re-insertion of the device.

The Flash MCU to Writer Programming Pin correspondence table is as follows:

Holtek Writer Pins	MCU Programming Pins	Pin Description
ICPDA	PA0	Programming Serial Data/Address
ICPCK	PA2	Programming Clock
VDD	VDD	Power Supply
VSS	VSS	Ground

The Program Memory can be programmed serially in-circuit using this 4-wire interface. Data is downloaded and uploaded serially on a single pin with an additional line for the clock. Two additional lines are required for the power supply. The technical details regarding the in-circuit programming of the device are beyond the scope of this document and will be supplied in supplementary literature.

During the programming process, the user must take care of the ICPDA and ICPCK pins for data and clock programming purposes to ensure that no other outputs are connected to these two pins.



Note: * may be resistor or capacitor. The resistance of * must be greater than 1kΩ or the capacitance of * must be less than 1nF.

On-Chip Debug Support – OCDS

There is an EV chip named HT45V5QC-5 which is used to emulate the real MCU device named HT45F5QC-5. The EV chip device also provides the “On-Chip Debug” function to debug the real MCU device during the development process. The EV chip and the real MCU device are almost functionally compatible except for “On-Chip Debug” function. Users can use the EV chip device to emulate the real chip device behavior by connecting the OCDSDA and OCDSCK pins to the Holtek HT-IDE development tools. The OCDSDA pin is the OCDS Data/Address input/output pin while the OCDSCK pin is the OCDS clock input pin. When users use the EV chip device for debugging, the corresponding pin functions shared with the OCDSDA and OCDSCK pins will have no effect in the EV chip. However, the two OCDS pins which are pin-shared with the ICP programming pins are still used as the Flash Memory programming pins for ICP. For more detailed OCDS information, refer to the corresponding document named “Holtek e-Link for 8-bit MCU OCDS User’s Guide”.

Holtek e-Link Pins	EV Chip OCDS Pins	Pin Description
OCDSDA	OCDSDA	On-Chip Debug Support Data/Address input/output
OCDSCK	OCDSCK	On-Chip Debug Support Clock input
VDD	VDD	Power Supply
VSS	VSS	Ground

In Application Programming – IAP

Flash type Program Memory provides the user with a means of convenient and easy upgrades and modifications to their programs on the same device. The provision of the IAP function offers users the convenience of Flash Memory multi-programming features. The convenience of the IAP function is that it can execute the updated program procedure using its internal firmware, without requiring an external Program Writer or PC. In addition, the IAP interface can also be any type of communication protocol, such as UART, using I/O pins. Regarding the internal firmware, the user can select versions provided by Holtek or create their own. The following section illustrates the procedures regarding how to implement the IAP firmware.

Flash Memory Read/Write Size

The Flash memory Erase and Write operations are carried out in a page format while the Read operation is carried out in a word format. The page size and write buffer size are both assigned with a capacity of 32 words. Note that the Erase operation should be executed before the Write operation is executed.

When the Flash Memory Erase/Write Function is successfully enabled, the CFWEN bit will be set high. When the CFWEN bit is set high, the data can be written into the write buffer. The FWT bit is used to initiate the write process and then indicate the write operation status. This bit is set high by application program to initiate a write process and will be cleared by hardware if the write process is finished.

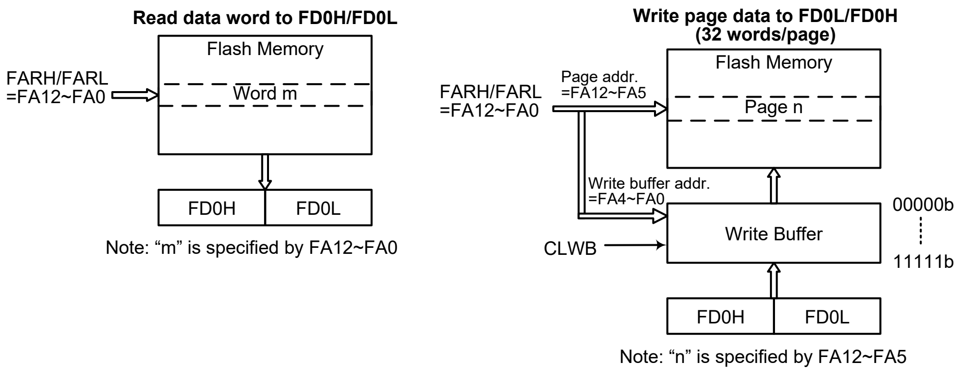
The Read operation can be carried out by executing a specific read procedure. The FRDEN bit is used to enable the read function and the FRD bit is used to initiate the read process by application programs and then indicate the read operation status. When the read process is finished, this bit will be cleared by hardware.

Operations	Format
Erase	32 words/time
Write	32 words/time
Read	1 word/time
Note: Page size=Write buffer size=32 words.	

IAP Operation Format

Page	FARH	FARL[7:5]	FARL[4:0]
0	0000 0000	000	Tag Address
1	0000 0000	001	
2	0000 0000	010	
3	0000 0000	011	
4	0000 0000	100	
5	0000 0000	101	
6	0000 0000	110	
7	0000 0000	111	
8	0000 0001	000	
: :	: :	: :	
254	0001 1111	110	
255	0001 1111	111	

Page Number and Address Selection



Flash Memory IAP Read/Write Structure

Write Buffer

The write buffer is used to store the written data temporarily when executing the write operation. The Write Buffer can be filled with written data after the Flash Memory Erase/Write Function has been successfully enabled by executing the Flash Memory Erase/Write Function Enable procedure. The write buffer can be cleared by configuring the CLWB bit in the FC2 register. The CLWB bit can be set high to enable the Clear Write Buffer procedure. When the procedure is finished this bit will be cleared to zero by hardware. It is recommended that the write buffer should be cleared by setting the CLWB bit high before the write buffer is used for the first time or when the data in the write buffer is updated.

The write buffer size is 32 words corresponding to a page. The write buffer address is mapped to a specific Flash memory page specified by the memory address bits, FA12~FA5. The data written into the FD0L and FD0H registers will be loaded into the write buffer. When data is written into the high byte data register, FD0H, it will result in the data stored in the high and low byte data registers both being written into the write buffer. It will also cause the Flash memory address to be incremented by one, after which the new address will be loaded into the FARH and FARL address registers. When the Flash memory address reaches the page boundary, 11111b of a page with 32 words, the address will now not be incremented but stop at the last address of the page. At this point a new page address should be specified for any other erase/write operations.

After a write process is finished, the write buffer will automatically be cleared by hardware. Note that the write buffer should be cleared manually by the application program when the data written into the Flash memory is incorrect in the data verification step. The data should again be written into the write buffer after the write buffer has been cleared when the data is found to be incorrect during the data verification step.

IAP Flash Program Memory Registers

There are two address registers, four pairs of 16-bit data registers and three control registersregisters, which are all located in Sector 0. Read and Write operations to the Flash memory are carried out using 16-bit data operations using the address and data registers and the control registers. Several registers control the overall operation of the internal Flash Program Memory. The address registers are named FARL and FARH, the data registers are named FDnL and FDnH, where n is equal to 0~3, and the control registers are named FC0, FC1 and FC2.

Register Name	Bit							
	7	6	5	4	3	2	1	0
FC0	CFWEN	FMOD2	FMOD1	FMOD0	FWPEN	FWT	FRDEN	FRD
FC1	D7	D6	D5	D4	D3	D2	D1	D0
FC2	—	—	—	—	—	—	FWERTS	CLWB
FARL	FA7	FA6	FA5	FA4	FA3	FA2	FA1	FA0
FARH	—	—	—	FA12	FA11	FA10	FA9	FA8
FD0L	D7	D6	D5	D4	D3	D2	D1	D0
FD0H	D15	D14	D13	D12	D11	D10	D9	D8
FD1L	D7	D6	D5	D4	D3	D2	D1	D0
FD1H	D15	D14	D13	D12	D11	D10	D9	D8
FD2L	D7	D6	D5	D4	D3	D2	D1	D0
FD2H	D15	D14	D13	D12	D11	D10	D9	D8
FD3L	D7	D6	D5	D4	D3	D2	D1	D0
FD3H	D15	D14	D13	D12	D11	D10	D9	D8

IAP Register List

• FC0 Register

Bit	7	6	5	4	3	2	1	0
Name	CFWEN	FMOD2	FMOD1	FMOD0	FWPEN	FWT	FRDEN	FRD
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7 **CFWEN**: Flash Memory Erase/Write function enable control
0: Flash memory erase/write function is disabled
1: Flash memory erase/write function has been successfully enabled

When this bit is cleared to 0 by application program, the Flash memory erase/write function is disabled. Note that writing a "1" into this bit results in no action. This bit is used to indicate the Flash memory erase/write function status. When this bit is set to 1 by hardware, it means that the Flash memory erase/write function is enabled successfully. Otherwise, the Flash memory erase/write function is disabled as the bit is zero.

Bit 6~4 **FMOD2~FMOD0**: Flash memory Mode selection
000: Write Mode
001: Page Erase Mode
011: Read Mode
110: Flash memory Erase/Write function Enable Mode
Other values: Reserved

These bits are used to select the Flash Memory operation modes. Note that the "Flash memory Erase/Write function Enable Mode" should first be successfully enabled before the Erase or Write Flash memory operation is executed.

Bit 3 **FWPEN**: Flash memory Erase/Write function enable procedure trigger control
0: Erase/Write function enable procedure is not triggered or procedure timer times out
1: Erase/Write function enable procedure is triggered and procedure timer starts to count

This bit is used to activate the Flash memory Erase/Write function enable procedure and an internal timer. It is set by the application programs and then cleared by hardware when the internal timer times out. The correct patterns must be written into the FD1L/FD1H, FD2L/FD2H and FD3L/FD3H register pairs respectively as soon as possible after the FWPEN bit is set high.

Bit 2 **FWT**: Flash memory write initiate control
0: Do not initiate Flash memory write or indicating that a Flash memory write process has completed
1: Initiate Flash memory write process

This bit is set by software and cleared by hardware when the Flash memory write process has completed.

Bit 1 **FRDEN**: Flash memory read enable control
0: Flash memory read disable
1: Flash memory read enable
This is the Flash memory Read Enable Bit which must be set high before any Flash memory read operations are carried out. Clearing this bit to zero will inhibit Flash memory read operations.

Bit 0 **FRD**: Flash memory read initiate control
0: Do not initiate Flash memory read or indicating that a Flash memory read process has completed
1: Initiate Flash memory read process
This bit is set by software and cleared by hardware when the Flash memory read process has completed.

Note: 1. The FWT, FRDEN and FRD bits cannot be set to "1" at the same time with a single instruction.
2. Ensure that the f_{SUB} clock is stable before executing the erase or write operation.
3. Note that the CPU will be stopped when a read, write or erase operation is successfully activated.
4. Ensure that the read, erase or write operation is totally complete before executing other operations.

• FC1 Register

Bit	7	6	5	4	3	2	1	0
Name	D7	D6	D5	D4	D3	D2	D1	D0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7~0 **D7~D0**: Chip Reset Pattern
When a specific value of "55H" is written into this register, a reset signal will be generated to reset the whole chip.

• FC2 Register

Bit	7	6	5	4	3	2	1	0
Name	—	—	—	—	—	—	FWERTS	CLWB
R/W	—	—	—	—	—	—	R/W	R/W
POR	—	—	—	—	—	—	0	0

Bit 7~2 Unimplemented, read as "0"

Bit 1 **FWERTS**: Erase time and Write time selection
0: Erase time is 3.2ms (t_{FER})/Write time is 2.2ms (t_{FWR})
1: Erase time is 3.7ms (t_{FER})/Write time is 3.0ms (t_{FWR})

Bit 0 **CLWB**: Flash memory Write Buffer Clear control
0: Do not initiate a Write Buffer Clear process or indicating that a Write Buffer Clear process has completed
1: Initiate Write Buffer Clear process
This bit is set by software and cleared by hardware when the Write Buffer Clear process has completed.

• FARL Register

Bit	7	6	5	4	3	2	1	0
Name	FA7	FA6	FA5	FA4	FA3	FA2	FA1	FA0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7~0 **FA7~FA0**: Flash Memory Address bit 7 ~ bit 0

- **FARH Register**

Bit	7	6	5	4	3	2	1	0
Name	—	—	—	FA12	FA11	FA10	FA9	FA8
R/W	—	—	—	R/W	R/W	R/W	R/W	R/W
POR	—	—	—	0	0	0	0	0

Bit 7~5 Unimplemented, read as "0"

Bit 4~0 **FA12~FA8**: Flash Memory Address bit 12 ~ bit 8

- **FD0L Register**

Bit	7	6	5	4	3	2	1	0
Name	D7	D6	D5	D4	D3	D2	D1	D0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7~0 **D7~D0**: The first Flash Memory data bit 7 ~ bit 0

Note that data written into the low byte data register FD0L will only be stored in the FD0L register and not loaded into the lower 8-bit write buffer.

- **FD0H Register**

Bit	7	6	5	4	3	2	1	0
Name	D15	D14	D13	D12	D11	D10	D9	D8
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7~0 **D15~D8**: The first Flash Memory data bit 15 ~ bit 8

Note that when 8-bit data is written into the high byte data register FD0H, the whole 16 bits of data stored in the FD0H and FD0L registers will simultaneously be loaded into the 16-bit write buffer after which the contents of the Flash memory address register pair, FARH and FARL, will be incremented by one.

- **FD1L Register**

Bit	7	6	5	4	3	2	1	0
Name	D7	D6	D5	D4	D3	D2	D1	D0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7~0 **D7~D0**: The second Flash Memory data bit 7 ~ bit 0

- **FD1H Register**

Bit	7	6	5	4	3	2	1	0
Name	D15	D14	D13	D12	D11	D10	D9	D8
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7~0 **D15~D8**: The second Flash Memory data bit 15 ~ bit 8

- **FD2L Register**

Bit	7	6	5	4	3	2	1	0
Name	D7	D6	D5	D4	D3	D2	D1	D0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7~0 **D7~D0**: The third Flash Memory data bit 7 ~ bit 0

- **FD2H Register**

Bit	7	6	5	4	3	2	1	0
Name	D15	D14	D13	D12	D11	D10	D9	D8
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7~0 **D15~D8**: The third Flash Memory data bit 15 ~ bit 8

- **FD3L Register**

Bit	7	6	5	4	3	2	1	0
Name	D7	D6	D5	D4	D3	D2	D1	D0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7~0 **D7~D0**: The fourth Flash Memory data bit 7 ~ bit 0

- **FD3H Register**

Bit	7	6	5	4	3	2	1	0
Name	D15	D14	D13	D12	D11	D10	D9	D8
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7~0 **D15~D8**: The fourth Flash Memory data bit 15 ~ bit 8

Flash Memory Erase/Write Flow

It is important to understand the Flash memory Erase/Write flow before the Flash memory contents are updated. Users can refer to the corresponding operation procedures when developing their IAP program to ensure that the Flash memory contents are correctly updated.

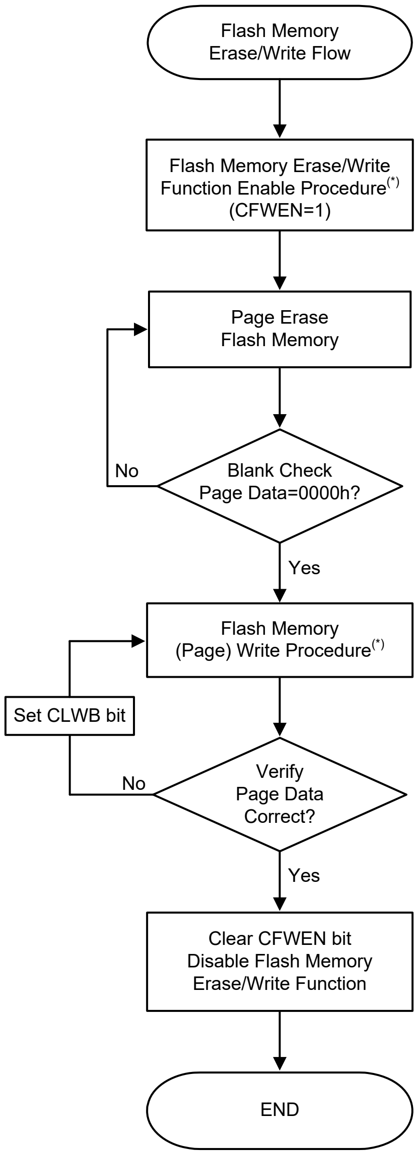
Flash Memory Erase/Write Flow Descriptions

1. Activate the “Flash Memory Erase/Write function enable procedure” first. When the Flash Memory Erase/Write function is successfully enabled, the CFWEN bit in the FC0 register will automatically be set high by hardware. After this, Erase or Write operations can be executed on the Flash memory. Refer to the “Flash Memory Erase/Write Function Enable Procedure” for details.
2. Configure the Flash memory address to select the desired erase page, tag address and then erase this page.

   For a page erase operation, set the FARL and FARH registers to specify the start address of the erase page, then write dummy data into the FD0H register to tag address. The current address will be internally incremented by one after each dummy data is written into the FD0H register. When the address reaches the page boundary, 11111b, the address will not be further incremented but stop at the last address of the page. Note that the write operation to the FD0H register is used to tag address, it must be implemented to determine which addresses to be erased.
3. Execute a Blank Check operation to ensure whether the page erase operation is successful or not. The “TABRD” instruction should be executed to read the Flash memory contents and to check if the contents is 0000h or not. If the Flash memory page erase operation fails, users should go back to Step 2 and execute the page erase operation again.
4. Write data into the specific page. Refer to the “Flash Memory Write Procedure” for details.
5. Execute the “TABRD” instruction to read the Flash memory contents and check if the written data is correct or not. If the data read from the Flash memory is different from the written data, it

means that the page write operation has failed. The CLWB bit should be set high to clear the write buffer and then write the data into the specific page again if the write operation has failed.

6. Clear the CFWEN bit to disable the Flash Memory Erase/Write function enable mode if the current page Erase and Write operations are complete if no more pages need to be erased or written.


Flash Memory Erase/Write Flow
Flash Memory Erase/Write Function Enable Procedure(*) (CFWEN=1)
Page Erase Flash Memory
No
Blank Check Page Data=0000h?
Yes
Flash Memory (Page) Write Procedure(*)
Set CLWB bit
No
Verify Page Data Correct?
Yes
Clear CFWEN bit Disable Flash Memory Erase/Write Function
END

Flash Memory Erase/Write Flow

Note: *The Flash Memory Erase/Write Function Enable procedure and Flash Memory Write procedure will be described in the following sections.

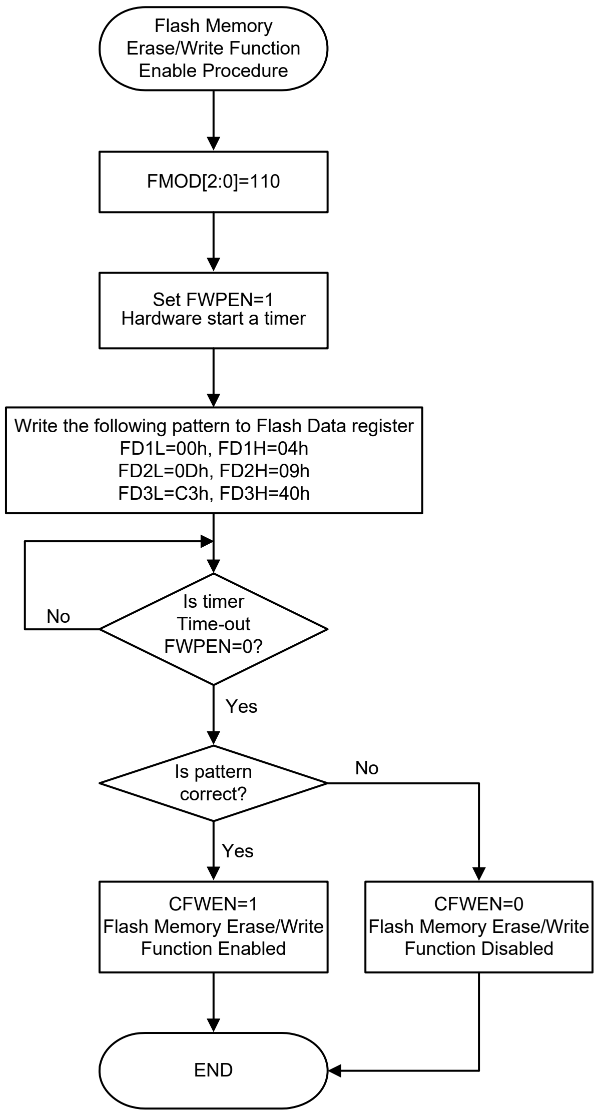
Flash Memory Erase/Write Function Enable Procedure

The Flash Memory Erase/Write Function Enable Mode is specially designed to prevent the Flash memory contents from being wrongly modified. In order to allow users to change the Flash memory data using the IAP control registers, users must first enable the Flash memory Erase/Write function.

Flash Memory Erase/Write Function Enable Procedure Description

1. Write data “110” to the FMOD [2:0] bits in the FC0 register to select the Flash Memory Erase/Write Function Enable Mode.
2. Set the FWPEN bit in the FC0 register to “1” to activate the Flash Memory Erase/Write Enable Function. This will also activate an internal timer.
3. Write the correct data pattern into the Flash data registers, FD1L~FD3L and FD1H~FD3H, as soon as possible after the FWPEN bit is set high. The enable Flash memory erase/write function data pattern is 00H, 0DH, C3H, 04H, 09H and 40H corresponding to the FD1L~FD3L and FD1H~FD3H registers respectively.
4. Once the timer has timed out, the FWPEN bit will automatically be cleared to 0 by hardware regardless of the input data pattern.
5. If the written data pattern is incorrect, the Flash memory erase/write function will not be enabled successfully and the above steps should be repeated. If the written data pattern is correct, the Flash memory erase/write function will be enabled successfully.
6. Once the Flash memory erase/write function is enabled, the Flash memory contents can be updated by executing the page erase and write operations using the IAP control registers.

To disable the Flash memory erase/write function, the CFWEN bit in the FC0 register can be cleared. There is no need to execute the above procedure.


Flash Memory Erase/Write Function Enable Procedure
FMOD[2:0]=110
Set FWPEN=1
Hardware start a timer
Write the following pattern to Flash Data register
FD1L=00h, FD1H=04h
FD2L=0Dh, FD2H=09h
FD3L=C3h, FD3H=40h
Is timer Time-out FWPEN=0?
No
Yes
Is pattern correct?
No
Yes
CFWEN=1
Flash Memory Erase/Write Function Enabled
CFWEN=0
Flash Memory Erase/Write Function Disabled
END

Flash Memory Erase/Write Function Enable Procedure

Flash Memory Write Procedure

After the Flash memory erase/write function has been successfully enabled as the CFWEN bit is set high, the data to be written into the Flash memory can be loaded into the write buffer. The selected Flash memory page data should be erased by properly configuring the IAP control registers before the data write procedure is executed.

The write buffer size is 32 words, known as a page, whose address is mapped to a specific Flash memory page specified by the memory address bits, FA12~FA5. It is important to ensure that the page where the write buffer data is located is the same one which the memory address bits, FA12~FA5, specify.

Flash Memory Consecutive Write Description

The maximum amount of write data is 32 words for each write operation. The write buffer address will be automatically incremented by one when consecutive write operations are executed. The start address of a specific page should first be written into the FARL and FARH registers. Then the data word should be written into the FD0L register and then the FD0H register. At the same time the write buffer address will be incremented by one and then the next data word can be written into the FD0L and FD0H registers for the next address without modifying the address register pair, FARH and FARL. When the write buffer address reaches the page boundary the address will not be further incremented but will stop at the last address of the page.

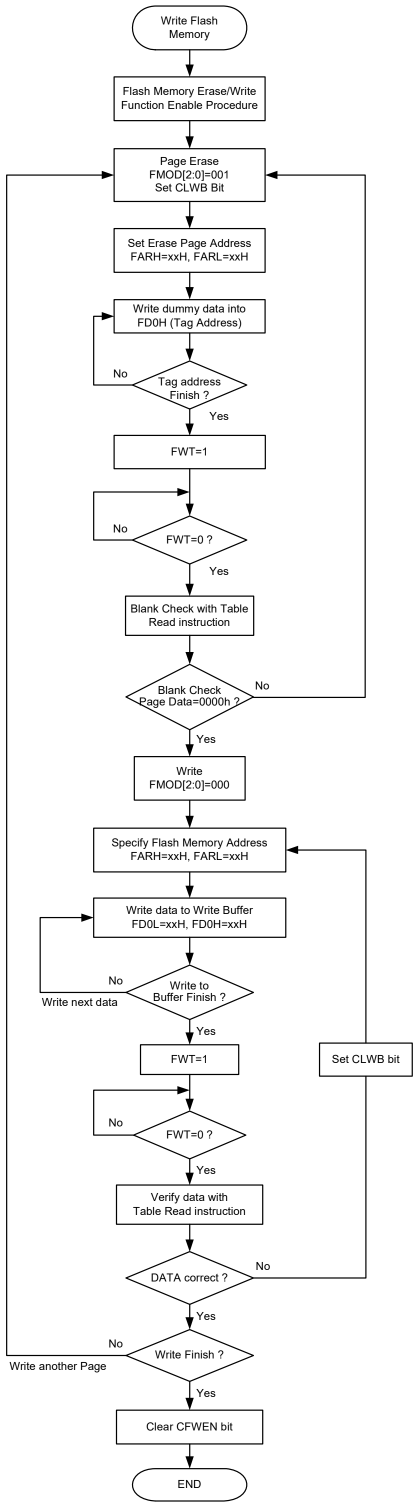
1. Activate the "Flash Memory Erase/Write function enable procedure". Check the CFWEN bit value and then execute the erase/write operations if the CFWEN bit is set high. Refer to the "Flash Memory Erase/Write function enable procedure" for more details.
2. Set the FMOD2~FMOD0 to "001" to select the erase operation and set the CLWB bit high to clear the write buffer. Set the FWT bit high to erase the desired page which is specified by the FARH and FARL registers and has been tagged address. Wait until the FWT bit goes low.
3. Execute a Blank Check operation using the table read instruction to ensure that the erase operation has successfully completed.

   Go to step 2 if the erase operation is not successful.

   Go to step 4 if the erase operation is successful.
4. Set the FMOD2~FMOD0 to "000" to select the write operation.
5. Setup the desired start address in the FARH and FARL registers. Write the desired data words consecutively into the FD0L and FD0H registers within a page as specified by their consecutive addresses. The maximum written data number is 32 words.
6. Set the FWT bit high to write the data words from the write buffer to the Flash memory. Wait until the FWT bit goes low.
7. Verify the data using the table read instruction to ensure that the write operation has successfully completed.

   If the write operation has not successfully completed, set the CLWB bit high to clear the write buffer and then go to step 5.

   Go to step 8 if the write operation is successful.
8. Clear the CFWEN bit low to disable the Flash memory erase/write function.


Write Flash Memory
Flash Memory Erase/Write Function Enable Procedure
Page Erase FMOD[2:0]=001 Set CLWB Bit
Set Erase Page Address FARH=xxH, FARL=xxH
Write dummy data into FD0H (Tag Address)
Tag address Finish ?
No
Yes
FWT=1
FWT=0 ?
No
Yes
Blank Check with Table Read instruction
Blank Check Page Data=0000h ?
No
Yes
Write FMOD[2:0]=000
Specify Flash Memory Address FARH=xxH, FARL=xxH
Write data to Write Buffer FD0L=xxH, FD0H=xxH
Write to Buffer Finish ?
No
Write next data
Yes
FWT=1
Set CLWB bit
FWT=0 ?
No
Yes
Verify data with Table Read instruction
DATA correct ?
No
Yes
Write Finish ?
No
Write another Page
Yes
Clear CFWEN bit
END

Flash Memory Consecutive Write Procedure

Note: 1. When the erase or write operation is successfully activated, all CPU operations will temporarily cease.

2. It will take certain time for the FWT bit state changing from high to low in the erase or write operation, which can be selected by the FWERTS bit in the FC2 register.

Flash Memory Non-consecutive Write Description

The main difference between Flash Memory Consecutive and Non-Consecutive Write operations is whether the data words to be written are located in consecutive addresses or not. If the data to be written is not located in consecutive addresses the desired address should be re-assigned after a data word is successfully written into the Flash Memory.

A two data word non-consecutive write operation is taken as an example here and described as follows:

1. Activate the “Flash Memory Erase/Write function enable procedure”. Check the CFWEN bit value and then execute the erase/write operation if the CFWEN bit is set high. Refer to the “Flash Memory Erase/Write function enable procedure” for more details.
2. Set the FMOD2~FMOD0 to “001” to select the erase operation and set the CLWB bit high to clear the write buffer. Set the FWT bit high to erase the desired page which is specified by the FARH and FARL registers and has been tagged address. Wait until the FWT bit goes low.
3. Execute a Blank Check operation using the table read instruction to ensure that the erase operation has successfully completed.

   Go to step 2 if the erase operation is not successful.

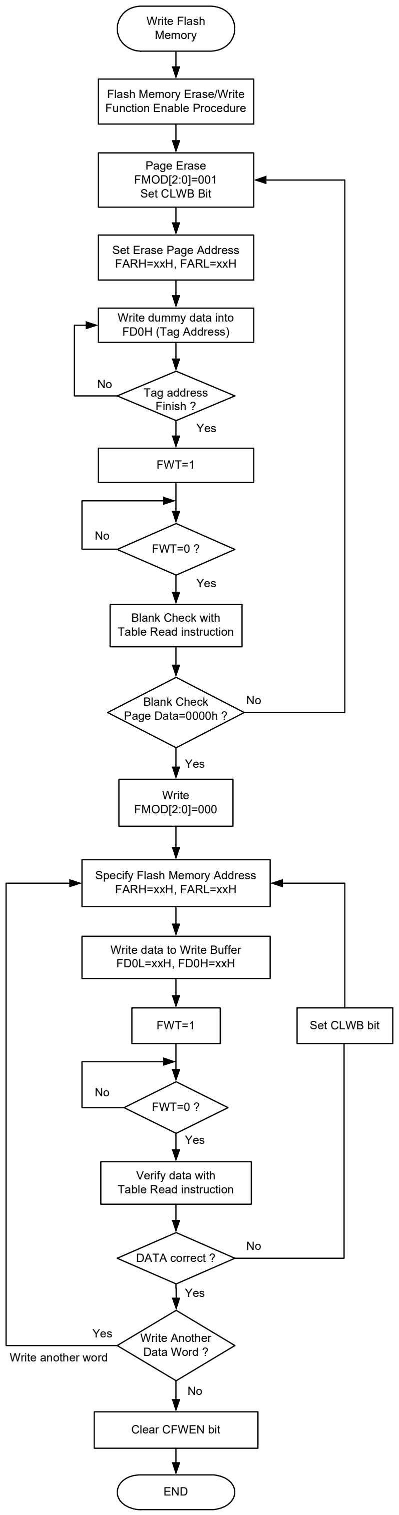
   Go to step 4 if the erase operation is successful.
4. Set the FMOD2~FMOD0 to “000” to select the write operation.
5. Setup the desired address ADDR1 in the FARH and FRARL registers. Write the desired data word DATA1 first into the FD0L register and then into the FD0H register.
6. Set the FWT bit high to transfer the data word from the write buffer to the Flash memory. Wait until the FWT bit goes low.
7. Verify the data using the table read instruction to ensure that the write operation has successfully completed.

   If the write operation has not successfully completed, set the CLWB bit high to clear the write buffer and then go to step 5.

   Go to step 8 if the write operation is successful.
8. Setup the desired address ADDR2 in the FARH and FARL registers. Write the desired data word DATA2 first into the FD0L register and then into the FD0H register.
9. Set the FWT bit high to transfer the data word from the write buffer to the Flash memory. Wait until the FWT bit goes low.
10. Verify the data using the table read instruction to ensure that the write operation has successfully completed.

    If the write operation has not successfully completed, set the CLWB bit high to clear the write buffer and then go to step 8.

    Go to step 11 if the write operation is successful.
11. Clear the CFWEN bit low to disable the Flash memory erase/write function.


Write Flash Memory
Flash Memory Erase/Write Function Enable Procedure
Page Erase FMOD[2:0]=001 Set CLWB Bit
Set Erase Page Address FARH=xxH, FARL=xxH
Write dummy data into FD0H (Tag Address)
Tag address Finish ?
No
Yes
FWT=1
FWT=0 ?
No
Yes
Blank Check with Table Read instruction
Blank Check Page Data=0000h ?
No
Yes
Write FMOD[2:0]=000
Specify Flash Memory Address FARH=xxH, FARL=xxH
Write data to Write Buffer FD0L=xxH, FD0H=xxH
FWT=1
Set CLWB bit
FWT=0 ?
No
Yes
Verify data with Table Read instruction
DATA correct ?
No
Yes
Write Another Data Word ?
Yes
Write another word
No
Clear CFWEN bit
END

Flash Memory Non-consecutive Write Procedure

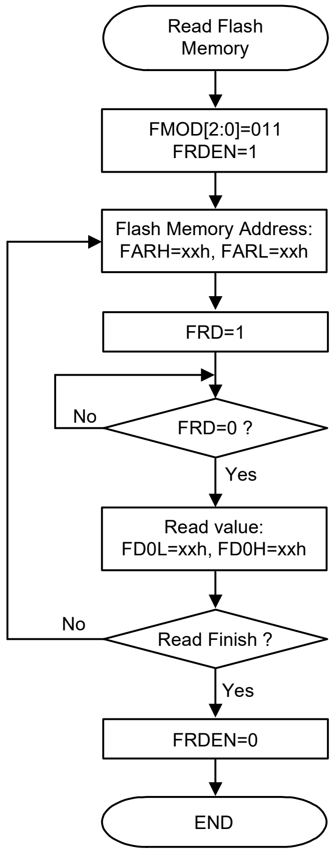
Note: 1. When the erase or write operation is successfully activated, all CPU operations will temporarily cease.
2. It will take certain time for the FWT bit state changing from high to low in the erase or write operation, which can be selected by the FWERTS bit in the FC2 register.

Important Points to Note for Flash Memory Write Operations

1. The “Flash Memory Erase/Write Function Enable Procedure” must be successfully activated before the Flash Memory erase/write operation is executed.
2. The Flash Memory erase operation is executed to erase a whole page.
3. The whole write buffer data will be written into the Flash memory in a page format. The corresponding address cannot exceed the page boundary.
4. After the data is written into the Flash memory the Flash memory contents must be read out using the table read instruction, TABRD, and checked if it is correct or not. If the data written into the Flash memory is incorrect, the write buffer should be cleared by setting the CLWB bit high and then writing the data again into the write buffer. Then activate a write operation on the same Flash memory page without erasing it. The data check, buffer clear and data re-write steps should be repeatedly executed until the data written into the Flash memory is correct.
5. The system frequency should be setup to the maximum application frequency when data write and data check operations are executed using the IAP function.

Flash Memory Read Procedure

To activate the Flash Memory Read procedure, the FMOD2~FMOD0 should be set to “011” to select the Flash memory read mode and the FRDEN bit should be set high to enable the read function. The desired Flash memory address should be written into the FARH and FARL registers and then the FRD bit should be set high. After this the Flash memory read operation will be activated. The data stored in the specified address can be read from the data registers, FD0H and FD0L, when the FRD bit goes low. There is no need to first activate the Flash Memory Erase/Write Function Enable Procedure before the Flash memory read operation is executed.



Flash Memory Read Procedure

Note: 1. When the read operation is successfully activated, all CPU operations will temporarily cease.

2. It will take a typical time of three instruction cycles for the FRD bit state changing from high to low.

Data Memory

The Data Memory is an 8-bit wide RAM internal memory and is the location where temporary information is stored.

Categorised into two types, the first of these is an area of RAM, known as the Special Function Data Memory. These registers have fixed locations and are necessary for correct operation of the device. Many of these registers can be read from and written to directly under program control, however, some remain protected from user manipulation. The second area of Data Memory is known as the General Purpose Data Memory, which is reserved for general purpose use. All locations within this area are read and write accessible under program control.

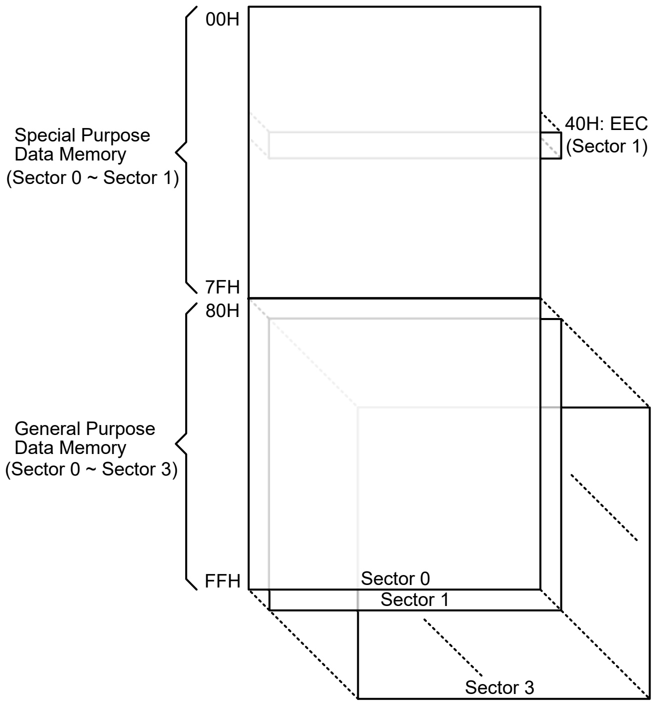
Switching between the different Data Memory sectors is achieved by properly setting the Memory Pointers to correct value when using the indirectly accessing method.

Structure

The Data Memory is subdivided into several sectors, all of which are implemented in 8-bit wide Memory. Each of the Data Memory Sector is categorized into two types, the Special Purpose Data Memory and the General Purpose Data Memory. The address range of the Special Purpose Data Memory for the device is from 00H to 7FH while the General Purpose Data Memory address range is from 80H to FFH.

Special Purpose Data Memory	General Purpose Data Memory	
Located Sectors	Capacity	Sector: Address
Sector 0: 00H~7FH Sector 1: 40H (EEC Only)	512×8	0: 80H~FFH 1: 80H~FFH 2: 80H~FFH 3: 80H~FFH

Data Memory Summary



Data Memory Structure

Data Memory Addressing

For device that supports the extended instructions, there is no Bank Pointer for Data Memory. For Data Memory the desired Sector is pointed by the MP1H or MP2H register and the certain Data Memory address in the selected sector is specified by the MP1L or MP2L register when using indirect addressing access.

Direct Addressing can be used in all sectors using the corresponding instruction which can address all available data memory space. For the accessed data memory which is located in any data memory sectors except sector 0, the extended instructions can be used to access the data memory instead of using the indirect addressing access. The main difference between standard instructions and extended instructions is that the data memory address "m" in the extended instructions has 10 valid bits for this device, the high byte indicates a sector and the low byte indicates a specific address.

General Purpose Data Memory

All microcontroller programs require an area of read/write memory where temporary data can be stored and retrieved for use later. It is this area of RAM memory that is known as General Purpose Data Memory. This area of Data Memory is fully accessible by the user programming for both reading and writing operations. By using the bit operation instructions individual bits can be set or reset under program control giving the user a large range of flexibility for bit manipulation in the Data Memory.

Special Purpose Data Memory

This area of Data Memory is where registers, necessary for the correct operation of the microcontroller, are stored. Most of the registers are both readable and writeable but some are protected and are readable only, the details of which are located under the relevant Special Function Register section. Note that for locations that are unused, any read instruction to these addresses will return the value "00H".

	Sector 0	Sector 1		Sector 0	Sector 1
00H	IAR0	▨	40H	EEAL	EEC
01H	MP0	▒	41H	EEAH	▒
02H	IAR1	▒	42H	EED	▒
03H	MP1L	▒	43H	FC0	▒
04H	MP1H	▒	44H	FC1	▒
05H	ACC	▒	45H	FC2	▒
06H	PCL	▒	46H	FARL	▒
07H	TBLP	▒	47H	FARH	▒
08H	TBLH	▒	48H	FD0L	▒
09H	TBHP	▒	49H	FD0H	▒
0AH	STATUS	▒	4AH	FD1L	▒
0BH	▒	▒	4BH	FD1H	▒
0CH	IAR2	▒	4CH	FD2L	▒
0DH	MP2L	▒	4DH	FD2H	▒
0EH	MP2H	▒	4EH	FD3L	▒
0FH	RSTFC	▒	4FH	FD3H	▒
10H	LVRC	▒	50H	CTMC0	▒
11H	TLVRC	▒	51H	CTMC1	▒
12H	SCC	▒	52H	CTMDL	▒
13H	HIRCC	▒	53H	CTMDH	▒
14H	PA	▒	54H	CTMAL	▒
15H	PAC	▒	55H	CTMAH	▒
16H	PAPU	▒	56H	STM0C0	▒
17H	PAWU	▒	57H	STM0C1	▒
18H	PB	▒	58H	STM0DL	▒
19H	PBC	▒	59H	STM0DH	▒
1AH	PBPU	▒	5AH	STM0AL	▒
1BH	PC	▒	5BH	STM0AH	▒
1CH	PCC	▒	5CH	STM1C0	▒
1DH	PCPU	▒	5DH	STM1C1	▒
1EH	PD	▒	5EH	STM1DL	▒
1FH	PDC	▒	5FH	STM1DH	▒
20H	PDPU	▒	60H	STM1AL	▒
21H	PAS0	▒	61H	STM1AH	▒
22H	PAS1	▒	62H	STM1RP	▒
23H	PBS0	▒	63H	STKPTR	▒
24H	PBS1	▒	64H	PCRL	▒
25H	▨	▒	65H	PCRH	▒
26H	PCS1	▒	66H	▨	▒
27H	▒	▒	67H	▨	▒
28H	IECC	▒	68H	▨	▒
29H	WDTC	▒	69H	▨	▒
2AH	TB0C	▒	6AH	▨	▒
2BH	TB1C	▒	6BH	▨	▒
2CH	SADC0	▒	6CH	▒	▒
2DH	SADC1	▒	6DH	▒	▒
2EH	SADC2	▒	6EH	DA0L	▒
2FH	SADOL	▒	6FH	DA0H	▒
30H	SADOH	▒	70H	DA1L	▒
31H	▒	▒	71H	DA1H	▒
32H	INTC0	▒	72H	DAOPC	▒
33H	INTC1	▒	73H	OPVOS	▒
34H	INTC2	▒	74H	PSCR	▒
35H	▨	▒	75H	▨	▒
36H	MFI0	▒	76H	RSTC	▒
37H	MFI1	▒	77H	▨	▒
38H	MFI2	▒	78H	▨	▒
39H	INTEG	▒	79H	▨	▒
3AH	▨	▒	7AH	▨	▒
3BH	▨	▒	7BH	▨	▒
3CH	CRCCR	▒	7CH	▨	▒
3DH	CRCIN	▒	7DH	▨	▒
3EH	CRCDL	▒	7EH	▨	▒
3FH	CRCDH	▒	7FH	▨	▒

▒ : Unused, read as 00H

▨ : Reserved, cannot be changed

Special Purpose Data Memory Structure

Special Function Register Description

Most of the Special Function Register details will be described in the relevant functional section. However, several registers require a separate description in this section.

Indirect Addressing Registers – IAR0, IAR1, IAR2

The Indirect Addressing Registers, IAR0, IAR1 and IAR2, although having their locations in normal RAM register space, do not actually physically exist as normal registers. The method of indirect addressing for RAM data manipulation uses these Indirect Addressing Registers and Memory Pointers, in contrast to direct memory addressing, where the actual memory address is specified. Actions on the IAR0, IAR1 and IAR2 registers will result in no actual read or write operation to these registers but rather to the memory location specified by their corresponding Memory Pointers, MP0, MP1L/MP1H or MP2L/MP2H. Acting as a pair, IAR0 and MP0 can together access data only from Sector 0 while the IAR1 register together with MP1L/MP1H register pair and IAR2 register together with MP2L/MP2H register pair can access data from any Data Memory sector. As the Indirect Addressing Registers are not physically implemented, reading the Indirect Addressing Registers will return a result of “00H” and writing to the registers will result in no operation.

Memory Pointers – MP0, MP1H/MP1L, MP2H/MP2L

Five Memory Pointers, known as MP0, MP1L, MP1H, MP2L and MP2H, are provided. These Memory Pointers are physically implemented in the Data Memory and can be manipulated in the same way as normal registers providing a convenient way with which to address and track data. When any operation to the relevant Indirect Addressing Registers is carried out, the actual address that the microcontroller is directed to is the address specified by the related Memory Pointer. MP0, together with Indirect Addressing Register, IAR0, are used to access data from Sector 0, while MP1L/MP1H together with IAR1 and MP2L/MP2H together with IAR2 are used to access data from all data sectors according to the corresponding MP1H or MP2H register. Direct Addressing can be used in all data sectors using the extended instructions which can address all available data memory space.

The following example shows how to clear a section of four Data Memory locations already defined as locations adres1 to adres4.

Indirect Addressing Program Example

Example 1

```
data .section ‘data’
adres1 db ?
adres2 db ?
adres3 db ?
adres4 db ?
block  db ?
code .section at 0 code
org 00h
start:
     mov a,04h              ; setup size of block
     mov block,a
     mov a,offset adres1    ; Accumulator loaded with first RAM address
     mov mp0,a              ; setup memory pointer with first RAM address
loop:
     clr IAR0               ; clear the data at address defined by MP0
     inc mp0                ; increment memory pointer
     sdz block              ; check if last memory location has been cleared
     jmp loop
continue:
```

Example 2

```
data .section ‘data’
adres1 db ?
adres2 db ?
adres3 db ?
adres4 db ?
block  db ?
code .section at 0 ‘code’
org 00h
start:
    mov a,04h              ; setup size of block
    mov block,a
    mov a,01h              ; setup the memory sector
    mov mp1h,a
    mov a,offset adres1    ; Accumulator loaded with first RAM address
    mov mp1l,a             ; setup memory pointer with first RAM address
loop:
    clr IAR1               ; clear the data at address defined by MP1L
    inc mp1l               ; increment memory pointer MP1L
    sdz block              ; check if last memory location has been cleared
    jmp loop
continue:
:
```

The important point to note here is that in the example shown above, no reference is made to specific RAM addresses.

Direct Addressing Program Example using extended instructions

```
data .section ‘data’
temp db ?
code .section at 0 code
org 00h
start:
    lmov a,[m]             ; move [m] data to acc
    lsub a, [m+1]          ; compare [m] and [m+1] data
    snz  c                 ; [m]>[m+1]?
    jmp  continue          ; no
    lmov a,[m]             ; yes, exchange [m] and [m+1] data
    mov  temp,a
    lmov a,[m+1]
    lmov [m],a
    mov  a,temp
    lmov [m+1],a
continue:
:
```

Note: Here “m” is a data memory address located in any data memory sectors. For example, m=1F0H, it indicates address 0F0H in Sector 1.

Accumulator – ACC

The Accumulator is central to the operation of any microcontroller and is closely related with operations carried out by the ALU. The Accumulator is the place where all intermediate results from the ALU are stored. Without the Accumulator it would be necessary to write the result of each calculation or logical operation such as addition, subtraction, shift, etc., to the Data Memory resulting in higher programming and timing overheads. Data transfer operations usually involve the temporary storage function of the Accumulator; for example, when transferring data between one user-defined register and another, it is necessary to do this by passing the data through the Accumulator as no direct transfer between two registers is permitted.

Program Counter Low Byte Register – PCL

To provide additional program control functions, the low byte of the Program Counter is made accessible to programmers by locating it within the Special Purpose area of the Data Memory. By manipulating this register, direct jumps to other program locations are easily implemented. Loading a value directly into this PCL register will cause a jump to the specified Program Memory location; however, as the register is only 8-bit wide, only jumps within the current Program Memory page are permitted. When such operations are used, note that a dummy cycle will be inserted.

Look-up Table Registers – TBLP, TBHP, TBLH

These three special function registers are used to control operation of the look-up table which is stored in the Program Memory. The TBLP and TBHP registers are the table pointer pair and indicates the location where the table data is located. Their value must be setup before any table read instructions are executed. Their value can be changed, for example using the "INC" or "DEC" instructions, allowing for easy table data pointing and reading. TBLH is the location where the high order byte of the table data is stored after a table read data instruction has been executed. Note that the lower order table data byte is transferred to a user defined location.

Status Register – STATUS

This 8-bit register contains the zero flag (Z), carry flag (C), auxiliary carry flag (AC), overflow flag (OV), SC flag, CZ flag, power down flag (PDF), and watchdog time-out flag (TO). These arithmetic/logical operation and system management flags are used to record the status and operation of the microcontroller.

With the exception of the TO and PDF flags, bits in the status register can be altered by instructions like most other registers. Any data written into the status register will not change the TO or PDF flag. In addition, operations related to the status register may give different results due to the different instruction operations. The TO flag can be affected only by a system power-up, a WDT time-out or by executing the "CLR WDT" or "HALT" instruction. The PDF flag is affected only by executing the "HALT" or "CLR WDT" instruction or during a system power-up.

The Z, OV, AC, C, SC and CZ flags generally reflect the status of the latest operations.

- C is set if an operation results in a carry during an addition operation or if a borrow does not take place during a subtraction operation; otherwise C is cleared. C is also affected by a rotate through carry instruction.
- AC is set if an operation results in a carry out of the low nibbles in addition, or no borrow from the high nibble into the low nibble in subtraction; otherwise AC is cleared.
- Z is set if the result of an arithmetic or logical operation is zero; otherwise Z is cleared.
- OV is set if an operation results in a carry into the highest-order bit but not a carry out of the highest-order bit, or vice versa; otherwise OV is cleared.

- PDF is cleared by a system power-up or executing the "CLR WDT" instruction. PDF is set by executing the "HALT" instruction.
- TO is cleared by a system power-up or executing the "CLR WDT" or "HALT" instruction. TO is set by a WDT time-out.
- SC is the result of the "XOR" operation which is performed by the OV flag and the MSB of the current instruction operation result.
- CZ is the operational result of different flags for different instructions. Refer to register definitions for more details.

In addition, on entering an interrupt sequence or executing a subroutine call, the status register will not be pushed onto the stack automatically. If the content of the status register is important and if the subroutine can corrupt the status register, precautions must be taken to correctly save it.

• STATUS Register

Bit	7	6	5	4	3	2	1	0
Name	SC	CZ	TO	PDF	OV	Z	AC	C
R/W	R/W	R/W	R	R	R/W	R/W	R/W	R/W
POR	x	x	0	0	x	x	x	x

"x": unknown

Bit 7 **SC**: The result of the "XOR" operation which is performed by the OV flag and the MSB of the instruction operation result

Bit 6 **CZ**: The operational result of different flags for different instructions

For SUB/SUBM/LSUB/LSUBM instructions, the CZ flag is equal to the Z flag.

For SBC/SBCM/LSBC/LSBCM instructions, the CZ flag is the "AND" operation result which is performed by the previous operation CZ flag and current operation Z flag. For other instructions, the CZ flag will not be affected.

Bit 5 **TO**: Watchdog Time-out flag
0: After power up or executing the "CLR WDT" or "HALT" instruction
1: A watchdog time-out occurred

Bit 4 **PDF**: Power down flag
0: After power up or executing the "CLR WDT" instruction
1: By executing the "HALT" instruction

Bit 3 **OV**: Overflow flag
0: No overflow
1: An operation results in a carry into the highest-order bit but not a carry out of the highest-order bit or vice versa

Bit 2 **Z**: Zero flag
0: The result of an arithmetic or logical operation is not zero
1: The result of an arithmetic or logical operation is zero

Bit 1 **AC**: Auxiliary flag
0: No auxiliary carry
1: An operation results in a carry out of the low nibbles, in addition, or no borrow from the high nibble into the low nibble in subtraction

Bit 0 **C**: Carry flag
0: No carry-out
1: An operation results in a carry during an addition operation or if a borrow does not take place during a subtraction operation

The "C" flag is also affected by a rotate through carry instruction.

EEPROM Data Memory

The device contains an area of internal EEPROM Data Memory. EEPROM is by its nature a non-volatile form of re-programmable memory, with data retention even when its power supply is removed. By incorporating this kind of data memory, a whole new host of application possibilities are made available to the designer. The availability of EEPROM storage allows information such as product identification numbers, calibration values, specific user data, system setup data or other product information to be stored directly within the product microcontroller. The process of reading and writing data to the EEPROM memory has been reduced to a very trivial affair.

EEPROM Data Memory Structure

The EEPROM Data Memory capacity is 512×8 bits for this device. Unlike the Program Memory and RAM Data Memory, the EEPROM Data Memory is not directly mapped into memory space and is therefore not directly addressable in the same way as the other types of memory. Read and Write operations to the EEPROM are carried out in single byte operations using a pair of address registers and a data register in Sector 0 and a single control register in Sector 1.

EEPROM Registers

Four registers control the overall operation of the internal EEPROM Data Memory. These are the address registers, EEAL and EEAH, the data register, EED and a single control register, EEC. As the EEAL, EEAH and EED registers are located in Sector 0, they can be directly accessed in the same way as any other Special Function Register. The EEC register, however, being located in Sector 1, can only be read from or written to indirectly using the MP1H/MP1L or MP2H/MP2L Memory Pointer pair and Indirect Addressing Register, IAR1 or IAR2. Because the EEC control register is located at address 40H in Sector 1, the Memory Pointer low byte register, MP1L or MP2L, must first be set to the value 40H and the Memory Pointer high byte register, MP1H or MP2H, set to the value, 01H, before any operations on the EEC register are executed.

Register Name	Bit							
	7	6	5	4	3	2	1	0
EEAL	EEAL7	EEAL6	EEAL5	EEAL4	EEAL3	EEAL2	EEAL1	EEAL0
EEAH	—	—	—	—	—	—	—	EEAH0
EED	EED7	EED6	EED5	EED4	EED3	EED2	EED1	EED0
EEC	EWERTS	EREN	ER	MODE	WREN	WR	RDEN	RD

EEPROM Register List

• EEAL Register

Bit	7	6	5	4	3	2	1	0
Name	EEAL7	EEAL6	EEAL5	EEAL4	EEAL3	EEAL2	EEAL1	EEAL0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7~0 **EEAL7~EEAL0**: Data EEPROM address low byte bit 7 ~ bit 0

• EEAH Register

Bit	7	6	5	4	3	2	1	0
Name	—	—	—	—	—	—	—	EEAH0
R/W	—	—	—	—	—	—	—	R/W
POR	—	—	—	—	—	—	—	0

Bit 7~1 Unimplemented, read as "0"
Bit 0 **EEAH0**: Data EEPROM address high byte bit 0

- **EED Register**

Bit	7	6	5	4	3	2	1	0
Name	EED7	EED6	EED5	EED4	EED3	EED2	EED1	EED0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7~0 **EED7~EED0**: Data EEPROM data bit 7 ~ bit 0

- **EEC Register**

Bit	7	6	5	4	3	2	1	0
Name	EWERTS	EREN	ER	MODE	WREN	WR	RDEN	RD
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7 **EWERTS**: Data EEPROM Erase time and Write time select
0: Erase time is 3.2ms (t_{EEER})/Write time is 2.2ms (t_{EEWR})
1: Erase time is 3.7ms (t_{EEER})/Write time is 3.0ms (t_{EEWR})

Bit 6 **EREN**: Data EEPROM erase enable
0: Disable
1: Enable

This bit is used to enable Data EEPROM erase function and must be set high before Data EEPROM erase operations are carried out. This bit will be automatically reset to zero by the hardware after the erase cycle has finished. Clearing this bit to zero will inhibit data EEPROM erase operations.

Bit 5 **ER**: Data EEPROM erase control
0: Erase cycle has finished
1: Activate an erase cycle

This is the Data EEPROM Erase Control Bit. When this bit is set high by the application program, an erase cycle will be activated. This bit will be automatically reset to zero by hardware after the erase cycle has finished. Setting this bit high will have no effect if the EREN bit has not first been set high.

Bit 4 **MODE**: Data EEPROM operation mode selection
0: Byte operation mode
1: Page operation mode

This is the EEPROM operation mode selection bit. When the bit is set high by the application program, the Page write, erase or read function will be selected. Otherwise, the byte write or read function will be selected. The EEPROM page buffer size is 16-byte.

Bit 3 **WREN**: Data EEPROM write enable
0: Disable
1: Enable

This is the Data EEPROM Write Enable bit which must be set high before Data EEPROM write operations are carried out. Clearing this bit to zero will inhibit Data EEPROM write operations. Note that the WREN bit will automatically be cleared to zero after the write operation is finished.

Bit 2 **WR**: Data EEPROM write control
0: Write cycle has finished
1: Activate a write cycle

This is the Data EEPROM Write Control Bit. When this bit is set high by the application program, a write cycle will be activated. This bit will be automatically reset to zero by hardware after the write cycle has finished. Setting this bit high will have no effect if the WREN has not first been set high.

Bit 1 **RDEN**: Data EEPROM read enable
0: Disable
1: Enable

This is the Data EEPROM Read Enable Bit, which must be set high before Data EEPROM read operations are carried out. Clearing this bit to zero will inhibit Data EEPROM read operations.

Bit 0 **RD**: Data EEPROM read control
0: Read cycle has finished
1: Activate a read cycle

This is the Data EEPROM Read Control Bit. When this bit is set high by the application program, a read cycle will be activated. This bit will be automatically reset to zero by hardware after the read cycle has finished. Setting this bit high will have no effect if the RDEN has not first been set high.

Note: 1. The EREN, ER, WREN, WR, RDEN and RD cannot be set to "1" at the same time in one instruction.
2. Ensure that the f_{SUB} clock is stable before executing the erase or write operation.
3. Ensure that the erase or write operation is totally complete before changing contents of the EEPROM related registers or activating the IAP function.

Read Operation from the EEPROM

Reading data from the EEPROM can be implemented by two modes for this device, byte read mode or page read mode, which is controlled by the EEPROM operation mode selection bit, MODE, in the EEC register.

Byte Read Mode

The EEPROM byte read operation can be executed when the mode selection bit, MODE, is cleared to zero. For a byte read operation the desired EEPROM address should first be placed in the EEAH and EEAL registers, as well as the read enable bit, RDEN, in the EEC register should be set high to enable the read function. Then setting the RD bit high will initiate the EEPROM byte read operation. Note that setting only the RD bit high will not initiate a read operation if the RDEN bit is not set high. When the read cycle terminates, the RD bit will automatically be cleared to zero and the EEPROM data can be read from the EED register. The data will remain in the EED register until another read or write operation is executed. The application program can poll the RD bit to determine when the data is valid for reading.

Page Read Mode

The EEPROM page read operation can be executed when the mode selection bit, MODE, is set high. The page size can be up to 16 bytes for the page read operation. For a page read operation the start address of the desired EEPROM page should first be placed in the EEAH and EEAL registers, as well as the read enable bit, RDEN, in the EEC register should be set high to enable the read function. Then setting the RD bit high will initiate the EEPROM page read operation. Note that setting only the RD bit high will not initiate a read operation if the RDEN bit is not set high. When the current byte read cycle terminates, the RD bit will automatically be cleared indicating that the EEPROM data can be read from the EED register, and the current address will be incremented by one by hardware. The data which is stored in the next EEPROM address can continuously be read out when the RD bit is set high again without reconfiguring the EEPROM address and RDEN control bit. The application program can poll the RD bit to determine when the data is valid for reading.

The EEPROM address higher 5 bits are used to specify the desired page location while the lower 4 bits are used to point to the actual address. In the page read operation mode the lower 4-bit address value will automatically be incremented by one. However, the higher 5-bit address value will not be incremented by hardware. When the EEPROM address lower 4-bit value which is internally incremented by one in the page mode reaches the page boundary, known as 0FH, the EEPROM address lower 4-bit value will stop at 0FH. The EEPROM address will not "roll over".

Page Erase Operation to the EEPROM

The EEPROM page erase operation can be executed when the mode selection bit, MODE, is set high. The EEPROM is capable of a 16-byte page erase. The internal page buffer will be cleared by hardware after power on reset. When the EEPROM erase enable control bit, namely EREN, is changed from "1" to "0", the internal page buffer will also be cleared. Note that when the EREN bit is changed from "0" to "1", the internal page buffer will not be cleared. The EEPROM address higher 5 bits are used to specify the desired page location while the lower 4 bits are used to point to the actual address. In the page erase operation mode the lower 4-bit address value will automatically be incremented by one after each dummy data byte is written into the EED register. However, the higher 5-bit address value will not be incremented by hardware. When the EEPROM address lower 4-bit value which is internally incremented by one in the page mode reaches the page boundary, known as 0FH, the EEPROM address lower 4-bit value will stop at 0FH. The EEPROM address will not "roll over".

For page erase operations the start address of the desired EEPROM page should first be placed in the EEAH and EEAL registers, then the dummy data to be written should be placed in the EED register. The maximum data length for a page is 16 bytes. Note that the write operation to the EED register is used to tag address, it must be implemented to determine which addresses to be erased. When the page dummy data is completely written, then the EREN bit in the EEC register should be set high to enable erase operations and the ER bit must be immediately set high to initiate the EEPROM erase process. These two instructions must be executed in two consecutive instruction cycles to activate an erase operation successfully. The global interrupt enable bit EMI should also first be cleared before implementing an erase operation and then set again after a valid erase activation procedure has completed.

Note: The above steps must be executed sequentially to successfully complete the page erase operation, refer to the corresponding programming example.

As the EEPROM erase cycle is controlled using an internal timer whose operation is asynchronous to microcontroller system clock, a certain time will elapse before the data will have been erased from the EEPROM. Detecting when the erase cycle has finished can be implemented either by polling the ER bit in the EEC register or by using the EEPROM interrupt. When the erase cycle terminates, the ER bit will be automatically cleared to zero by the microcontroller, indicating that the page data has been erased. The application program can therefore poll the ER bit to determine when the erase cycle has ended. After the erase operation is finished, the EREN bit will be cleared to zero by hardware. The Data EEPROM erased page content will all be zero after a page erase operation.

Write Operation to the EEPROM

Writing data to the EEPROM can be implemented by two modes for this device, byte write mode or page write mode, which is controlled by the EEPROM operation mode selection bit, MODE, in the EEC register.

Byte Write Mode

The EEPROM byte write operation can be executed when the mode selection bit, MODE, is cleared to zero. For byte write operations the desired EEPROM address should first be placed in the EEAH and EEAL registers, then the data to be written should be placed in the EED register. To write data to the EEPROM, the write enable bit, WREN, in the EEC register must first be set high to enable the write function. After this, the WR bit in the EEC register must be immediately set high to initiate a write cycle. These two instructions must be executed in two consecutive instruction cycles to activate a write operation successfully. The global interrupt bit EMI should also first be cleared before implementing any write operations, and then set high again after a valid write activation procedure has completed. Note that setting the WR bit high only will not initiate a write cycle if the WREN bit is not set.

Note: The above steps must be executed sequentially to successfully complete the byte write operation, refer to the corresponding programming example.

As the EEPROM write cycle is controlled using an internal timer whose operation is asynchronous to microcontroller system clock, a certain time will elapse before the data will have been written into the EEPROM. Detecting when the write cycle has finished can be implemented either by polling the WR bit in the EEC register or by using the EEPROM interrupt. When the write cycle terminates, the WR bit will be automatically cleared to zero by the microcontroller, indicating that the data has been written to the EEPROM. The application program can therefore poll the WR bit to determine when the write cycle has ended. After the write operation is finished, the WREN bit will be cleared to zero by hardware. Note that a byte erase operation will automatically be executed before a byte write operation is successfully activated.

Page Write Mode

Before a page write operation is executed, it is important to ensure that a relevant page erase operation has been successfully executed. The EEPROM page write operation can be executed when the mode selection bit, MODE, is set high. The EEPROM is capable of a 16-byte page write. The internal page buffer will be cleared by hardware after power-on reset. When the EEPROM write enable control bit, namely WREN, is changed from "1" to "0", the internal page buffer will also be cleared. Note that when the WREN bit is changed from "0" to "1", the internal page buffer will not be cleared. A page write is initiated in the same way as a byte write initiation except that the EEPROM data can be written up to 16 bytes. The EEPROM address higher 5 bits are used to specify the desired page location while the lower 4 bits are used to point to the actual address. In the page write operation mode the lower 4-bit address value will automatically be incremented by one after each data byte is written into the EED register. However, the higher 5-bit address value will not be incremented by hardware. When the EEPROM address lower 4-bit value which is internally incremented by one in the page mode reaches the page boundary, known as 0FH, the EEPROM address lower 4-bit value will stop at 0FH. The EEPROM address will not "roll over". At this point any data write operations to the EED register will be invalid.

For page write operations the start address of the desired EEPROM page should first be placed in the EEAH and EEAL registers, then the data to be written should be placed in the EED register. The maximum data length for a page is 16 bytes. Note that when a data byte is written into the EED register, then the data in the EED register will be loaded into the internal page buffer and the current address value will automatically be incremented by one. When the page data is completely written into the page buffer, then the WREN bit in the EEC register should be set high to enable write operations and the WR bit must be immediately set high to initiate the EEPROM write process. These two instructions must be executed in two consecutive instruction cycles to activate a write operation successfully. The global interrupt enable bit EMI should also first be cleared before implementing any write operations, and then set high again after a valid write activation procedure has completed. Note that setting the WR bit high only will not initiate a write cycle if the WREN bit is not set.

Note: The above steps must be executed sequentially to successfully complete the page write operation, refer to the corresponding programming example.

As the EEPROM write cycle is controlled using an internal timer whose operation is asynchronous to microcontroller system clock, a certain time will elapse before the data will have been written into the EEPROM. Detecting when the write cycle has finished can be implemented either by polling the WR bit in the EEC register or by using the EEPROM interrupt. When the write cycle terminates, the WR bit will be automatically cleared to zero by the microcontroller, indicating that the data has been written to the EEPROM. The application program can therefore poll the WR bit to determine when the write cycle has ended. After the write operation is finished, the WREN bit will be cleared to zero by hardware.

Write Protection

Protection against inadvertent write operation is provided in several ways. After the device is powered-on the Write Enable bit in the control register will be cleared preventing any write operations. Also at power-on the Memory Pointer high byte register, MP1H or MP2H, will be reset to zero, which means that Data Memory Sector 0 will be selected. As the EEPROM control register is located in Sector 1, this adds a further measure of protection against spurious write operations. During normal program operation, ensuring that the Write Enable bit in the control register is cleared will safeguard against incorrect write operations.

EEPROM Interrupt

The EEPROM interrupt is generated when an EEPROM erase or write cycle has ended. The EEPROM interrupt must first be enabled by setting the DEE bit in the relevant interrupt register. When an EEPROM erase or write cycle ends, the DEF request flag will be set. If the global and EEPROM interrupts are enabled and the stack is not full, a jump to the EEPROM Interrupt vector will take place. When the interrupt is serviced, the EEPROM interrupt flag will be automatically reset. More details can be obtained in the Interrupt section.

Programming Considerations

Care must be taken that data is not inadvertently written to the EEPROM. Protection can be enhanced by ensuring that the Write Enable bit is normally cleared to zero when not writing. Also the Memory Pointer high byte register, MP1H or MP2H, could be normally cleared to zero as this would inhibit access to Sector 1 where the EEPROM control register exists. Although certainly not necessary, consideration might be given in the application program to the checking of the validity of new write data by a simple read back process.

When writing data the WR bit must be set high immediately after the WREN bit has been set high, to ensure the write cycle executes correctly. When erasing data the ER bit must be set high immediately after the EREN bit has been set high, to ensure the erase cycle executes correctly.The global interrupt bit EMI should also be cleared before a write or erase cycle is executed and then set again after a valid write or erase activation procedure has completed. Note that the device should not enter the IDLE or SLEEP mode until the EEPROM read/write/erase operation is totally complete. Otherwise, the EEPROM read/write/erase operation will fail.

Programming Examples

Reading a Data Byte from the EEPROM – polling method

```
MOV  A, 40H                ; setup memory mointer lower byte MP1L
MOV  MP1L, A               ; MP1L points to EEC register
MOV  A, 01H                ; setup memory pointer high byte MP1H
MOV  MP1H, A
CLR  IAR1.4                ; clear MODE bit, select byte operatiom mode
MOV  A, EEPROM_ADRES_H     ; user defined high byte address
MOV  EEAH, A
MOV  A, EEPROM_ADRES_L     ; user defined low byte address
MOV  EEAL, A
SET  IAR1.1                ; set RDEN bit, enable read operations
SET  IAR1.0                ; start Read Cycle - set RD bit
BACK:
SZ   IAR1.0                ; check for read cycle end
JMP  BACK
CLR  IAR1                  ; disable EEPROM read function
CLR  MP1H
MOV  A, EED                ; move read data to register
MOV  READ_DATA, A
```

Reading a Data Page from the EEPROM – polling method

```
MOV  A, 40H                 ; set memory pointer low byte MP1L
MOV  MP1L, A                ; MP1L points to EEC register
MOV  A, 01H                 ; set memory pointer high byte MP1H
MOV  MP1H, A
SET  IAR1.4                 ; set MODE bit, select page operation mode
MOV  A, EEPROM_ADRES_H      ; user defined high byte address
MOV  EEAH, A
MOV  A, EEPROM_ADRES_L      ; user defined low byte address
MOV  EEAL, A
SET  IAR1.1                 ; set RDEN bit, enable read operations
; ~~~~ The data length can be up to 16 bytes (Start) ~~~~
CALL READ
CALL READ
:
:
JMP PAGE_READ_FINISH
; ~~~~ The data length can be up to 16 bytes (End) ~~~~
READ:
SET  IAR1.0                 ; start Read Cycle - set RD bit
BACK:
SZ   IAR1.0                 ; check for read cycle end
JMP  BACK
MOV  A, EED                 ; move read data to register
MOV  READ_DATA, A
RET
:
PAGE_READ_FINISH:
CLR  IAR1                   ; disable EEPROM read function
CLR  MP1H
```

Erasing a Data Page to the EEPROM – polling method

```
MOV  A, 40H                 ; set memory pointer low byte MP1L
MOV  MP1L, A                ; MP1L points to EEC register
MOV  A, 01H                 ; set memory pointer high byte MP1H
MOV  MP1H, A
SET  IAR1.4                 ; set MODE bit, select page operation mode
MOV  A, EEPROM_ADRES_H      ; user defined high byte address
MOV  EEAH, A
MOV  A, EEPROM_ADRES_L      ; user defined low byte address
MOV  EEAL, A
; ~~~~ The data length can be up to 16 bytes (Start) ~~~~
CALL WRITE_BUF
CALL WRITE_BUF
:
:
JMP  Erase_START
; ~~~~ The data length can be up to 16 bytes (End) ~~~~
WRITE_BUF:
MOV  A, EEPROM_DATA         ; user defined data, erase mode don't care data value
MOV  EED, A
RET
:
Erase_START:
CLR  EMI
SET  IAR1.6                 ; set EREN bit, enable write operations
SET  IAR1.5                 ; start Write Cycle - set ER bit – executed immediately
                            ; after setting EREN bit
SET  EMI
```

```
BACK:
SZ   IAR1.5                 ; check for write cycle end
JMP  BACK
CLR  MP1H
```

Writing a Data Byte to the EEPROM – polling method

```
MOV  A, 40H                 ; set memory pointer low byte MP1L
MOV  MP1L, A                ; MP1L points to EEC register
MOV  A, 01H                 ; set memory pointer high byte MP1H
MOV  MP1H, A
CLR  IAR1.4                 ; clear MODE bit, select byte write mode
MOV  A, EEPROM_ADRES        ; user defined high byte address
MOV  EEAH, A
MOV  A, EEPROM_ADRES        ; user defined low byte address
MOV  EEAL, A
MOV  A, EEPROM_DATA         ; user defined data
MOV  EED, A
CLR  EMI
SET  IAR1.3                 ; set WREN bit, enable write operations
SET  IAR1.2                 ; start Write Cycle - set WR bit – executed immediately
                            ; after setting WREN bit
SET  EMI
BACK:
SZ   IAR1.2                 ; check for write cycle end
JMP  BACK
CLR  MP1H
```

Writing a Data Page to the EEPROM – polling method

```
MOV  A, 40H                 ; set memory pointer low byte MP1L
MOV  MP1L, A                ; MP1L points to EEC register
MOV  A, 01H                 ; set memory pointer high byte MP1H
MOV  MP1H, A
SET  IAR1.4                 ; set MODE bit, select page operation mode
MOV  A, EEPROM_ADRES_H      ; user defined high byte address
MOV  EEAH, A
MOV  A, EEPROM_ADRES_L      ; user defined low byte address
MOV  EEAL, A
; ~~~~ The data length can be up to 16 bytes (Start) ~~~~
CALL WRITE_BUF
CALL WRITE_BUF
:
:
JMP WRITE_START
; ~~~~ The data length can be up to 16 bytes (End) ~~~~
WRITE_BUF:
MOV  A, EEPROM_DATA         ; user define data
MOV  EED, A
RET
:
WRITE_START:
CLR  EMI
SET  IAR1.3                 ; set WREN bit, enable write operations
SET  IAR1.2                 ; start Write Cycle - set WR bit – executed immediately
                            ; after setting WREN bit
SET  EMI
BACK:
SZ   IAR1.2                 ; check for write cycle end
JMP  BACK
CLR  MP1H
```

Oscillators

Various oscillator types offer the user a wide range of functions according to their various application requirements. The flexible features of the oscillator functions ensure that the best optimisation can be achieved in terms of speed and power saving. Oscillator selections and operation are selected through the relevant control registers.

Oscillator Overview

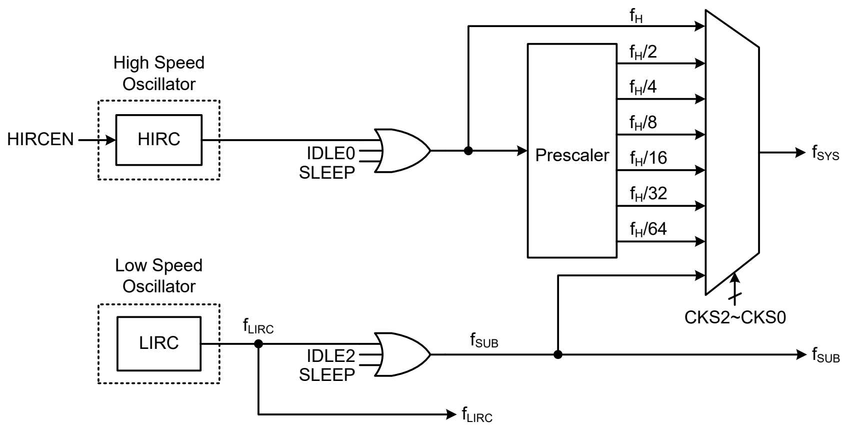
In addition to being the source of the main system clock the oscillators also provide clock sources for the Watchdog Timer and Time Base Interrupts. Two fully integrated internal oscillators, requiring no external components, are provided to form a wide range of both fast and slow system oscillators. The higher frequency oscillator provides higher performance but carry with it the disadvantage of higher power requirements, while the opposite is of course true for the lower frequency oscillator. With the capability of dynamically switching between fast and slow system clock, the device has the flexibility to optimize the performance/power ratio, a feature especially important in power sensitive portable applications.

Type	Name	Frequency
Internal High Speed RC	HIRC	8MHz
Internal Low Speed RC	LIRC	32kHz

Oscillator Types

System Clock Configurations

There are two oscillator sources, a high speed oscillator and a low speed oscillator. The high system clock is sourced from the internal 8MHz RC oscillator, HIRC. The low speed oscillator is the internal 32kHz RC oscillator, LIRC. Selecting whether the low or high speed oscillator is used as the system oscillator is implemented using the CKS2~CKS0 bits in the SCC register and as the system clock can be dynamically selected.



System Clock Configurations

Internal High Speed RC Oscillator – HIRC

The internal RC oscillator is a fully integrated system oscillator requiring no external components. The internal RC oscillator has a fixed frequency of 8MHz. Device trimming during the manufacturing process and the inclusion of internal frequency compensation circuits are used to ensure that the influence of the power supply voltage, temperature and process variations on the oscillation frequency are minimised.

Internal 32kHz Oscillator – LIRC

The Internal 32kHz System Oscillator is the low frequency oscillator. It is a fully integrated RC oscillator with a typical frequency of 32kHz, requiring no external components for its implementation. Device trimming during the manufacturing process and the inclusion of internal frequency compensation circuits are used to ensure that the influence of the power supply voltage, temperature and process variations on the oscillation frequency are minimised.

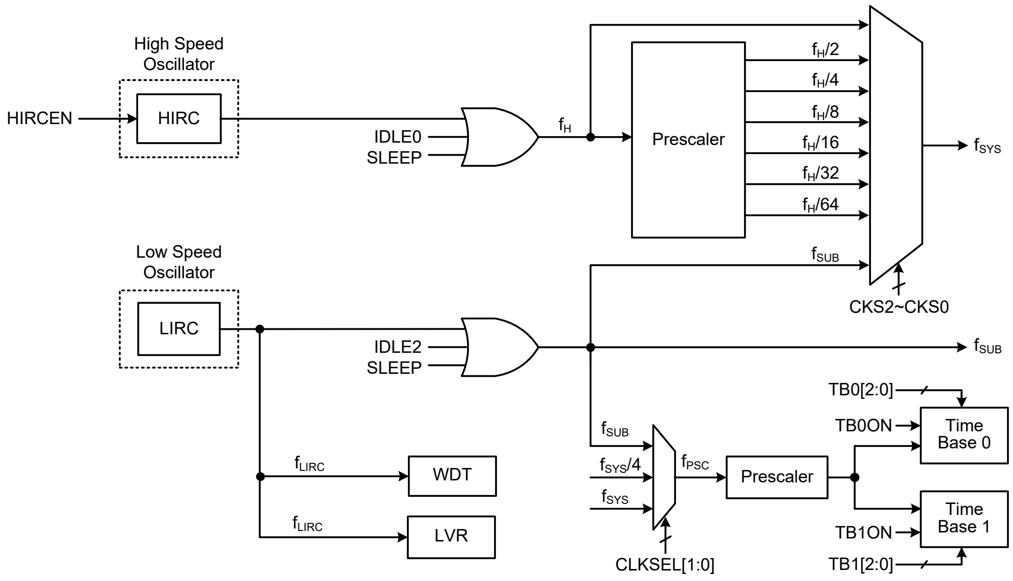
Operating Modes and System Clocks

Present day applications require that their microcontrollers have high performance but often still demand that they consume as little power as possible, conflicting requirements that are especially true in battery powered portable applications. The fast clocks required for high performance will by their nature increase current consumption and of course vice-versa lower speed clocks reduce current consumption. As Holtek has provided the device with both high and low speed clock sources and the means to switch between them dynamically, the user can optimise the operation of their microcontroller to achieve the best performance/power ratio.

System Clocks

The device has many different clock sources for both the CPU and peripheral function operation. By providing the user with a wide range of clock selections using register programming, a clock system can be configured to obtain maximum application performance.

The main system clock, can come from either a high frequency, f_H, or low frequency, f_{SUB}, source, and is selected using the CKS2~CKS0 bits in the SCC register. The high speed system clock is sourced from the HIRC oscillator. The low speed system clock source is sourced from the LIRC oscillator. The other choice, which is a divided version of the high speed system oscillator has a range of $f_H/2$~$f_H/64$.



Device Clock Configurations

Note: When the system clock source f_{SYS} is switched to f_{SUB} from f_H, the high speed oscillation can be stopped to conserve the power or continue to oscillate to provide the clock source, f_H~$f_H/64$, for peripheral circuits to use, which is determined by configuring the corresponding high speed oscillator enable control bit.

System Operation Modes

There are six different modes of operation for the microcontroller, each one with its own special characteristics and which can be chosen according to the specific performance and power requirements of the application. There are two modes allowing normal operation of the microcontroller, the FAST Mode and SLOW Mode. The remaining four modes, the SLEEP, IDLE0, IDLE1 and IDLE2 Mode are used when the microcontroller CPU is switched off to conserve power.

Operation Mode	CPU	Register Setting			f_{SYS}	f_H	f_{SUB}	f_{LIRC}
		FHIDEN	FSIDEN	CKS2~CKS0				
FAST	On	x	x	000~110	f_H~f_H/64	On	On	On
SLOW	On	x	x	111	f_{SUB}	On/Off (1)	On	On
IDLE0	Off	0	1	000~110	Off	Off	On	On
				111	On			
IDLE1	Off	1	1	xxx	On	On	On	On
IDLE2	Off	1	0	000~110	On	On	Off	On
				111	Off			
SLEEP	Off	0	0	xxx	Off	Off	Off	On/Off (2)

"x": don't care

Note: 1. The f_H clock will be switched on or off by configuring the corresponding oscillator enable bit in the SLOW mode.

2. The f_{LIRC} clock will be switched on or off which is controlled by the WDT function being enabled or disabled in the SLEEP mode.

FAST Mode

This is one of the main operating modes where the microcontroller has all of its functions operational and where the system clock is provided by the internal high speed oscillator. This mode operates allowing the microcontroller to operate normally with a clock source which will come from the HIRC oscillator. The high speed oscillator will however first be divided by a ratio ranging from 1 to 64, the actual ratio being selected by the CKS2~CKS0 bits in the SCC register. Although a high speed oscillator is used, running the microcontroller at a divided clock ratio reduces the operating current.

SLOW Mode

This is also a mode where the microcontroller operates normally although now with a slower speed clock source. The clock source used will be from f_{SUB}. The f_{SUB} clock is derived from the LIRC oscillator.

SLEEP Mode

The SLEEP Mode is entered when a HALT instruction is executed and when the FHIDEN and FSIDEN bits in the SCC register are both low. In the SLEEP mode the CPU will be stopped. The f_{SUB} clock provided to the peripheral function will also be stopped. However the f_{LIRC} clock will continue to operate if the WDT function is enabled.

IDLE0 Mode

The IDLE0 Mode is entered when a HALT instruction is executed and when the FHIDEN bit in the SCC register is low and the FSIDEN bit in the SCC register is high. In the IDLE0 Mode the CPU will be switched off but the low speed oscillator will be on to drive some peripheral functions.

IDLE1 Mode

The IDLE1 Mode is entered when a HALT instruction is executed and when the FHIDEN and FSIDEN bits in the SCC register are both high. In the IDLE1 Mode the CPU will be switched off but both the high and low speed oscillators will be on to provide a clock source to keep some peripheral functions operational.

IDLE2 Mode

The IDLE2 Mode is entered when a HALT instruction is executed and when the FHIDEN bit in the SCC register is high and the FSIDEN bit in the SCC register is low. In the IDLE2 Mode the CPU will be switched off but the high speed oscillator will be on to provide a clock source to keep some peripheral functions operational.

Control Registers

The SCC and HIRCC registers are used to control the system clock and the HIRC oscillator configurations.

Register Name	Bit							
	7	6	5	4	3	2	1	0
SCC	CKS2	CKS1	CKS0	—	—	—	FHIDEN	FSIDEN
HIRCC	—	—	—	—	—	—	HIRCF	HIRCEN

System Operating Mode Control Register List

• SCC Register

Bit	7	6	5	4	3	2	1	0
Name	CKS2	CKS1	CKS0	—	—	—	FHIDEN	FSIDEN
R/W	R/W	R/W	R/W	—	—	—	R/W	R/W
POR	0	0	0	—	—	—	0	0

Bit 7~5 **CKS2~CKS0**: System clock selection
000: f_H
001: $f_H/2$
010: $f_H/4$
011: $f_H/8$
100: $f_H/16$
101: $f_H/32$
110: $f_H/64$
111: f_{SUB}
These three bits are used to select which clock is used as the system clock source. In addition to the system clock source directly derived from f_H or f_{SUB}, a divided version of the high speed system oscillator can also be chosen as the system clock source.

Bit 4~2 Unimplemented, read as "0"

Bit 1 **FHIDEN**: High Frequency oscillator control when CPU is switched off
0: Disable
1: Enable
This bit is used to control whether the high speed oscillator is activated or stopped when the CPU is switched off by executing a "HALT" instruction.

Bit 0 **FSIDEN**: Low Frequency oscillator control when CPU is switched off
0: Disable
1: Enable
This bit is used to control whether the low speed oscillator is activated or stopped when the CPU is switched off by executing a "HALT" instruction.

Note: A certain delay is required before the relevant clock is successfully switched to the target clock source after any clock switching setup using the CKS2~CKS0 bits. A proper delay time must be arranged before executing the following operations which require immediate reaction with the target clock source.

Clock switching delay time=4×t_{SYS}+[0~(1.5×$t_{Curr.}$+0.5×$t_{Tar.}$)], where $t_{Curr.}$ indicates the current clock period, t_{Tar}. Indicates the target clock period and t_{SYS} indicates the current sytem clock period.

• HIRCC Register

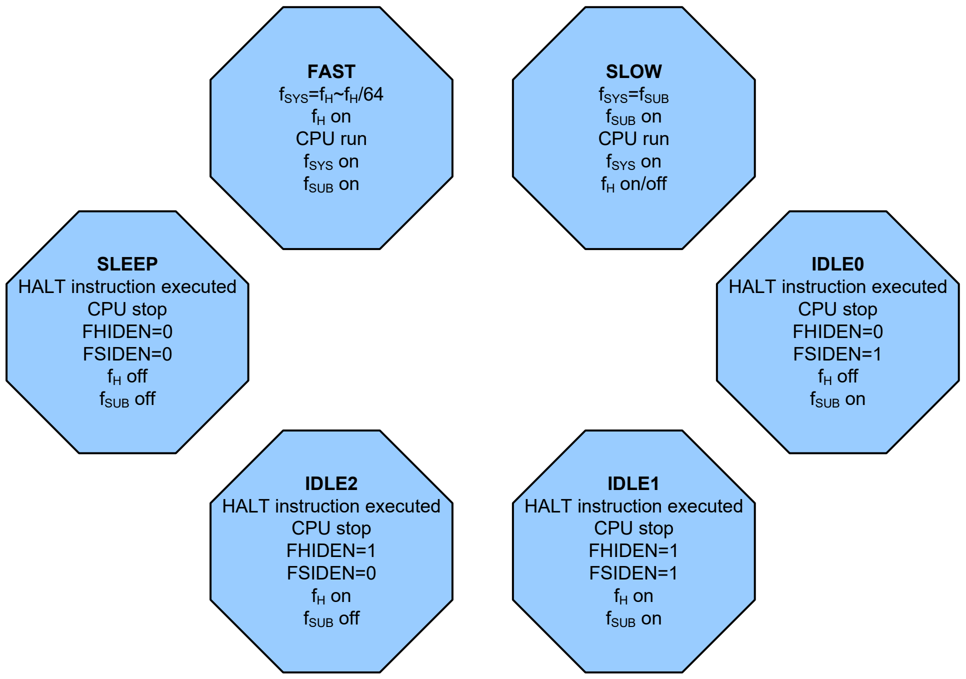
Bit	7	6	5	4	3	2	1	0
Name	—	—	—	—	—	—	HIRCF	HIRCEN
R/W	—	—	—	—	—	—	R	R/W
POR	—	—	—	—	—	—	0	1

Bit 7~2	Unimplemented, read as “0”
Bit 1	**HIRCF**: HIRC oscillator stable flag 0: Unstable 1: Stable This bit is used to indicate whether the HIRC oscillator is stable or not. When the HIRCEN bit is set to 1 to enable the HIRC oscillator, the HIRCF bit will first be cleared to 0 and then set to 1 after the HIRC oscillator is stable.
Bit 0	**HIRCEN**: HIRC oscillator enable control 0: Disable 1: Enable

Operating Mode Switching

The device can switch between operating modes dynamically allowing the user to select the best performance/power ratio for the present task in hand. In this way microcontroller operations that do not require high performance can be executed using slower clocks thus requiring less operating current and prolonging battery life in portable applications.

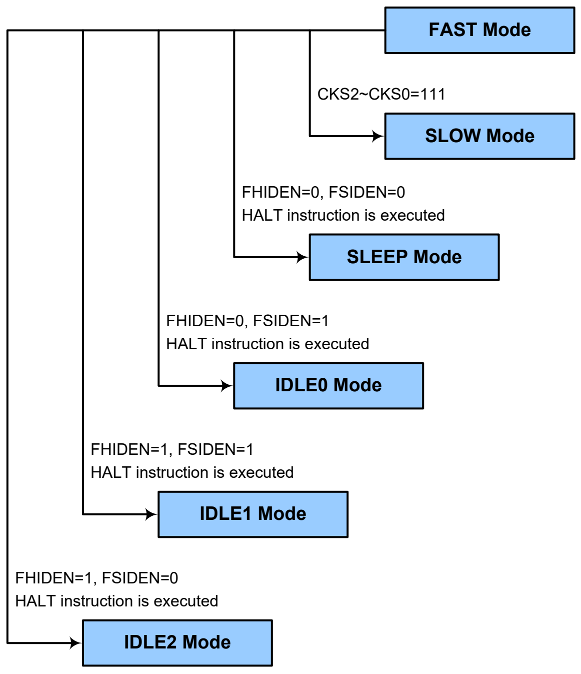
In simple terms, Mode Switching between the FAST Mode and SLOW Mode is executed using the CKS2~CKS0 bits in the SCC register while Mode Switching from the FAST/SLOW Modes to the SLEEP/IDLE Modes is executed via the HALT instruction. When a HALT instruction is executed, whether the device enters the IDLE Mode or the SLEEP Mode is determined by the condition of the FHIDEN and FSIDEN bits in the SCC register.



FAST Mode to SLOW Mode Switching

When running in the FAST Mode, which uses the high speed system oscillator, and therefore consumes more power, the system clock can switch to run in the SLOW Mode by setting the CKS2~CKS0 bits to "111" in the SCC register. This will then use the low speed system oscillator which will consume less power. Users may decide to do this for certain operations which do not require high performance and can subsequently reduce power consumption.

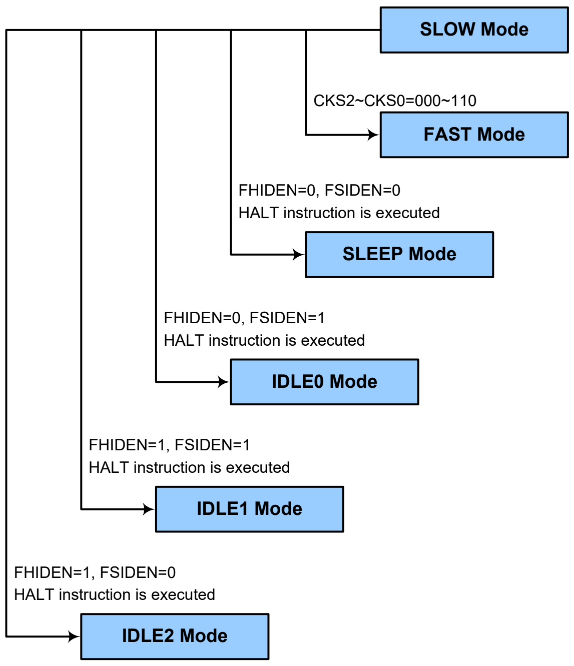
The SLOW Mode is sourced from the LIRC oscillator and therefore requires this oscillator to be stable before full mode switching occurs.



SLOW Mode to FAST Mode Switching

In SLOW mode the system clock is derived from f_{SUB}. When system clock is switched back to the FAST mode from f_{SUB}, the CKS2~CKS0 bits should be set to "000"~"110" and then the system clock will respectively be switched to f_H~$f_H/64$.

However, if f_H is not used in SLOW mode and thus switched off, it will take some time to re-oscillate and stabilise when switching to the FAST mode from the SLOW Mode. This is monitored using the HIRCF bit in the HIRCC register. The time duration required for the high speed system oscillator stabilization is specified in the System Start Up Time Characteristics.



Entering the SLEEP Mode

There is only one way for the device to enter the SLEEP Mode and that is to execute the "HALT" instruction in the application program with both the FHIDEN and FSIDEN bits in the SCC register equal to "0". When this instruction is executed under the conditions described above, the following will occur:

- The system clock will be stopped and the application program will stop at the "HALT" instruction.
- The Data Memory contents and registers will maintain their present condition.
- The I/O ports will maintain their present conditions.
- In the status register, the Power Down flag PDF will be set, and WDT timeout flag TO will be cleared.
- The WDT will be cleared and resume counting if the WDT function is enabled. If the WDT function is disabled, the WDT will be cleared and then stopped.

Entering the IDLE0 Mode

There is only one way for the device to enter the IDLE0 Mode and that is to execute the "HALT" instruction in the application program with the FHIDEN bit in the SCC register equal to "0" and the FSIDEN bit in the SCC register equal to "1". When this instruction is executed under the conditions described above, the following will occur:

- The f_H clock will be stopped and the application program will stop at the "HALT" instruction, but the f_{SUB} clock will be on.
- The Data Memory contents and registers will maintain their present condition.
- The I/O ports will maintain their present conditions.
- In the status register, the Power Down flag PDF will be set, and WDT timeout flag TO will be cleared.
- The WDT will be cleared and resume counting if the WDT function is enabled. If the WDT function is disabled, the WDT will be cleared and then stopped.

Entering the IDLE1 Mode

There is only one way for the device to enter the IDLE1 Mode and that is to execute the "HALT" instruction in the application program with both the FHIDEN and FSIDEN bits in the SCC register equal to "1". When this instruction is executed under the conditions described above, the following will occur:

- The f_H and f_{SUB} clocks will be on but the application program will stop at the "HALT" instruction.
- The Data Memory contents and registers will maintain their present condition.
- The I/O ports will maintain their present conditions.
- In the status register, the Power Down flag PDF will be set, and WDT timeout flag TO will be cleared.
- The WDT will be cleared and resume counting if the WDT function is enabled. If the WDT function is disabled, the WDT will be cleared and then stopped.

Entering the IDLE2 Mode

There is only one way for the device to enter the IDLE2 Mode and that is to execute the "HALT" instruction in the application program with the FHIDEN bit in the SCC register equal to "1" and the FSIDEN bit in the SCC register equal to "0". When this instruction is executed under the conditions described above, the following will occur:

- The f_H clock will be on but the f_{SUB} clock will be off and the application program will stop at the "HALT" instruction.
- The Data Memory contents and registers will maintain their present condition.
- The I/O ports will maintain their present conditions.
- In the status register, the Power Down flag PDF will be set, and WDT timeout flag TO will be cleared.
- The WDT will be cleared and resume counting if the WDT function is enabled. If the WDT function is disabled, the WDT will be cleared and then stopped.

Standby Current Considerations

As the main reason for entering the SLEEP or IDLE Mode is to keep the current consumption of the device to as low a value as possible, perhaps only in the order of several micro-amps except in the IDLE1 and IDLE2 Mode, there are other considerations which must also be taken into account by the circuit designer if the power consumption is to be minimised. Special attention must be made to the I/O pins on the device. All high-impedance input pins must be connected to either a fixed high or low level as any floating input pins could create internal oscillations and result in increased current consumption. This also applies to the device which has different package types, as there may be unbonded pins. These must either be setup as outputs or if setup as inputs must have pull-high resistors connected.

Care must also be taken with the loads, which are connected to I/O pins, which are setup as outputs. These should be placed in a condition in which minimum current is drawn or connected only to external circuits that do not draw current, such as other CMOS inputs. Also note that additional standby current will also be required if the LIRC oscillator has been enabled.

In the IDLE1 and IDLE2 Mode the high speed oscillator is on, if the peripheral function clock source is derived from the high speed oscillator, the additional standby current will also be perhaps in the order of several hundred micro-amps.

Wake-up

To minimise power consumption the device can enter the SLEEP or any IDLE Mode, where the CPU will be switched off. However, when the device is woken up again, it will take a considerable time for the original system oscillator to restart, stabilise and allow normal operation to resume.

After the system enters the SLEEP or IDLE Mode, it can be woken up from one of various sources listed as follows:

- An external pin reset
- An external falling edge on Port A (except the PA2 pin)
- A system interrupt
- A WDT overflow

If the system is woken up by an external $\overline{\text{RES}}$ pin reset, the device will experience a full system reset, however, if the device is woken up by a WDT overflow, a Watchdog Timer reset will be initiated. Although both of these wake-up methods will initiate a reset operation, the actual source of the wake-up can be determined by examining the TO and PDF flags. The PDF flag is cleared by a system power-up or executing the clear Watchdog Timer instructions and is set when executing the "HALT" instruction. The TO flag is set if a WDT time-out occurs, and causes a wake-up that only resets the Program Counter and Stack Pointer, the other flags remain in their original status.

Each pin on Port A can be setup using the PAWU register to permit a negative transition on the pin to wake up the system. When a Port A pin wake-up occurs, the program will resume execution at the instruction following the "HALT" instruction.If the system is woken up by an interrupt, then two possible situations may occur. The first is where the related interrupt is disabled or the interrupt is enabled but the stack is full, in which case the program will resume execution at the instruction following the "HALT" instruction. In this situation, the interrupt which woke up the device will not be immediately serviced, but will rather be serviced later when the related interrupt is finally enabled or when a stack level becomes free. The other situation is where the related interrupt is enabled and the stack is not full, in which case the regular interrupt response takes place. If an interrupt request flag is set high before entering the SLEEP or IDLE Mode, the wake-up function of the related interrupt will be disabled.

Watchdog Timer

The Watchdog Timer is provided to prevent program malfunctions or sequences from jumping to unknown locations, due to certain uncontrollable external events such as electrical noise.

Watchdog Timer Clock Source

The Watchdog Timer clock source is provided by the internal clock, f_{LIRC}, which is sourced from the LIRC oscillator. The LIRC internal oscillator has an approximate frequency of 32kHz and this specified internal clock period can vary with V_{DD}, temperature and process variations. The Watchdog Timer source clock is then subdivided by a ratio of 2^8 to 2^{18} to give longer timeouts, the actual value being chosen using the WS2~WS0 bits in the WDTC register.

Watchdog Timer Control Register

A single register, WDTC, controls the required time-out period as well as Watchdog Timer the enable/disable and the MCU reset operation.

• WDTC Register

Bit	7	6	5	4	3	2	1	0
Name	WE4	WE3	WE2	WE1	WE0	WS2	WS1	WS0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	1	0	1	0	0	1	1

Bit 7~3 **WE4~WE0**: WDT function enable control
10101: Disable
01010: Enable
Other values: Reset MCU
When these bits are changed to any other values due to environmental noise the microcontroller will be reset; this reset operation will be activated after a delay time, t_{SRESET}, and the WRF bit in the RSTFC register will be set high.

Bit 2~0 **WS2~WS0**: WDT time-out period selection
000: $2^8/f_{LIRC}$
001: $2^{10}/f_{LIRC}$
010: $2^{12}/f_{LIRC}$
011: $2^{14}/f_{LIRC}$
100: $2^{15}/f_{LIRC}$
101: $2^{16}/f_{LIRC}$
110: $2^{17}/f_{LIRC}$
111: $2^{18}/f_{LIRC}$
These three bits determine the division ratio of the watchdog timer source clock, which in turn determines the time-out period.

• RSTFC Register

Bit	7	6	5	4	3	2	1	0
Name	—	—	—	—	RSTF	LVRF	LRF	WRF
R/W	—	—	—	—	R/W	R/W	R/W	R/W
POR	—	—	—	—	0	x	0	0

"x": unknown

Bit 7~4 Unimplemented, read as "0"
Bit 3 **RSTF**: Reset control register software reset flag
Refer to the $\overline{RES}$ Pin Reset section.
Bit 2 **LVRF**: LVR function reset flag
Refer to the Low Voltage Reset section.

Bit 1 **LRF**: LVRC register software reset flag
Refer to the Low Voltage Reset section.

Bit 0 **WRF**: WDT control register software reset flag
0: Not occurred
1: Occurred
This bit is set to 1 by the WDT control register software reset and cleared by the application program. This bit can only be cleared to zero by application program.

Watchdog Timer Operation

The Watchdog Timer operates by providing a device reset when its timer overflows. This means that in the application program and during normal operation the user has to strategically clear the Watchdog Timer before it overflows to prevent the Watchdog Timer from executing a reset. This is done using the clear watchdog instruction. If the program malfunctions for whatever reason, jumps to an unknown location, or enters an endless loop, the clear instruction will not be executed in the correct manner, in which case the Watchdog Timer will overflow and reset the device. There are five bits, WE4~WE0, in the WDTC register to offer the Watchdog Timer enable/disable control and the MCU reset. The WDT function will be enabled when the WE4~WE0 bits are set to a value of 01010B while the WDT function will be disabled if the WE4~WE0 bits are equal to 10101B. If the WE4~WE0 bits are set to any other values rather than 01010B and 10101B, it will reset the device after a delay time, t_{SRESET}. After power on these bits will have a value of 01010B.

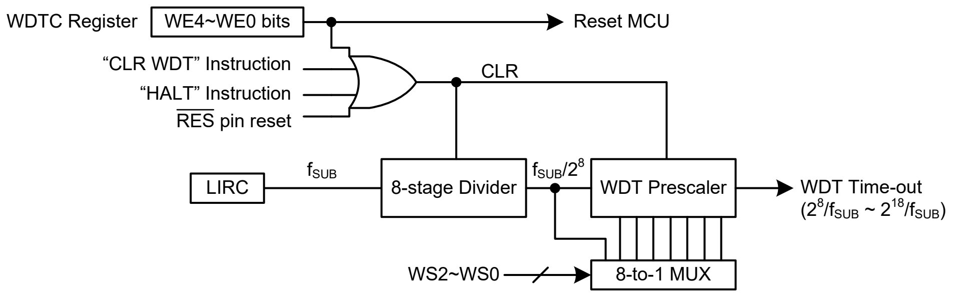
WE4~WE0 Bits	WDT Function
10101B	Disable
01010B	Enable
Any other value	Reset MCU

Watchdog Timer Function Control

Under normal program operation, a Watchdog Timer time-out will initialise a device reset and set the status bit TO. However, if the system is in the SLEEP or IDLE Mode, when a Watchdog Timer time-out occurs, the TO bit in the status register will be set and only the Program Counter and Stack Pointer will be reset. Four methods can be adopted to clear the contents of the Watchdog Timer. The first is a WDTC register software reset, which means a certain value except 01010B and 10101B written into the WE4~WE0 bits, the second is using the Watchdog Timer software clear instruction and the third is via a HALT instruction. The last is an external hardware reset, which means a low level on the external reset pin if the external reset pin is selected by the RSTC register.

There is only one method of using software instruction to clear the Watchdog Timer. That is to use the single "CLR WDT" instruction to clear the WDT contents.

The maximum time out period is when the 2^{18} division ratio is selected. As an example, with a 32kHz LIRC oscillator as its source clock, this will give a maximum watchdog period of around 8 seconds for the 2^{18} division ratio and a minimum timeout of 8ms for the 2^{8} division ration.



Watchdog Timer

Reset and Initialisation

A reset function is a fundamental part of any microcontroller ensuring that the device can be set to some predetermined condition irrespective of outside parameters. The most important reset condition is after power is first applied to the microcontroller. In this case, internal circuitry will ensure that the microcontroller, after a short delay, will be in a well defined state and ready to execute the first program instruction. After this power-on reset, certain important internal registers will be set to defined states before the program commences. One of these registers is the Program Counter, which will be reset to zero forcing the microcontroller to begin program execution from the lowest Program Memory address.

In addition to the power-on reset, situations may arise where it is necessary to forcefully apply a reset condition when the microcontroller is already running, the $\overline{\text{RES}}$ line is forcefully pulled low. In such a case, known as a normal operation reset, some of the microcontroller registers remain unchanged allowing the microcontroller to preceed with normal operation after the reset line is allowed to return high.

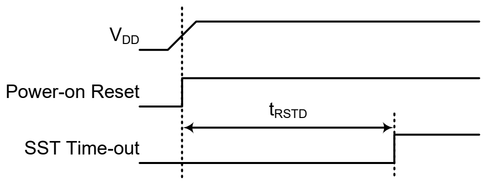
Another reset exists in the form of a Low Voltage Reset, LVR, where a full reset, similar to the $\overline{\text{RES}}$ reset is implemented in situations where the power supply voltage falls below a certain threshold. Another type of reset is when the Watchdog Timer overflows and resets the microcontroller. All types of reset operations result in different register conditions being setup.

Reset Functions

There are several ways in which a microcontroller reset can occur, through events occurring both externally and internally.

Power-on Reset

The most fundamental and unavoidable reset is the one that occurs after power is first applied to the microcontroller. As well as ensuring that the Program Memory begins execution from the first memory address, a power-on reset also ensures that certain other registers are preset to known conditions. All the I/O port and port control registers will power up in a high condition ensuring that all pins will be first set to inputs.



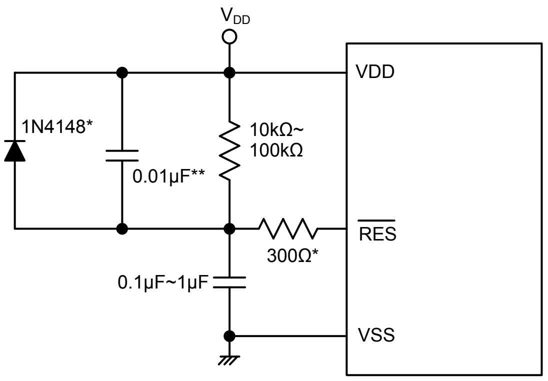
Power-On Reset Timing Chart

$\overline{\text{RES}}$ Pin Reset

The external reset pin is a dedicated pin of which a high to low transition will reset the microcontroller. As the reset pin is shared with I/O pins, the reset function must be selected using the control register, RSTC. Although the microcontroller has an internal RC reset function, if the V_{DD} power supply rise time is not fast enough or does not stabilise quickly at power-on, the internal reset function may be incapable of providing proper reset operation. For this reason it is recommended that an external RC network is connected to the $\overline{\text{RES}}$ pin, whose additional time delay will ensure that the $\overline{\text{RES}}$ pin remains low for an extended period to allow the power supply to stabilise. During this time delay, normal operation of the microcontroller will be inhibited. After the $\overline{\text{RES}}$ line reaches a certain voltage value, the reset delay time, t_{RSTD}, is invoked to provide an extea delay time after which the microcontroller will begin normal operation. The abbreviation SST in the figures stands for System Start-up Time.

For most applications a resistor connected between VDD and the $\overline{\text{RES}}$ line and a capacitor connected betweeb VSS and the $\overline{\text{RES}}$ pin will provide a suitable external reset circuit. Any wiring connected to the $\overline{\text{RES}}$ pin should be kept as short as possible to minimise any stray noise interference.

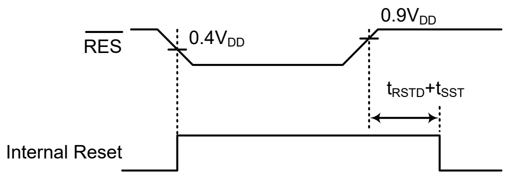
For applications that operate within an environment where more noise is present the Enhanced Reset Circuit shown is recommended.



Note: "*" It is recommended that this component is added for added ESD protection.
"**" It is recommended that this component is added in environments where power line noise is significant.

External $\overline{\text{RES}}$ Circuit

Pulling the $\overline{\text{RES}}$ pin low using external hardware will also execute a device reset. In this case, as in the case of other resets, the Program Counter will reset to zero and program execution initiated from this point.



$\overline{\text{RES}}$ Reset Timing Chart

There is an internal reset control register, RSTC, which is used to select the external $\overline{\text{RES}}$ pin function and provide a reset when the device operates abnormally due to the environmental noise interference. If the content of the RSTC register is set to any value other than 01010101B or 10101010B, it will reset the device after a delay time, t_{SRESET}. After power on the register will have a value of 10101010B.

RSTC7~RSTC0 Bits	Reset Function
01010101B	General purpose output pin function
10101010B	$\overline{\text{RES}}$ pin
Any other value	Reset MCU

Internal Reset Function Control

• **RSTC Register**

Bit	7	6	5	4	3	2	1	0
Name	RSTC7	RSTC6	RSTC5	RSTC4	RSTC3	RSTC2	RSTC1	RSTC0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	1	0	1	0	1	0	1	0

Bit 7~0 **RSTC7~RSTC0**: Reset function control
01010101: General purpose output pin function, PA2OEN=1
10101010: $\overline{\text{RES}}$ pin, PA2OEN=0
Other values: Reset MCU

If these bits are changed due to adverse environmental conditions, the microcontroller will be reset. The reset operation will be activated after a delay time, t_{SRESET} and the RSTF bit in the RSTFC register will be set to 1.

All resets will reset this register to POR value except the WDT time-out hardware warm reset. Note that if the register is set to 10101010 to set the $\overline{\text{RES}}$ pin, this configuration has higher priority than other related pin-shared controls.

• **RSTFC Register**

Bit	7	6	5	4	3	2	1	0
Name	—	—	—	—	RSTF	LVRF	LRF	WRF
R/W	—	—	—	—	R/W	R/W	R/W	R/W
POR	—	—	—	—	0	x	0	0

"x": unknown

Bit 7~4 Unimplemented, read as "0"

Bit 3 **RSTF**: Reset control register software reset flag
0: Not occurred
1: Occurred

This bit is set to 1 by the RSTC control register software reset and cleared by the application program. This bit can only be cleared to zero by application program.

Bit 2 **LVRF**: LVR function reset flag
Refer to the Low Voltage Reset section.

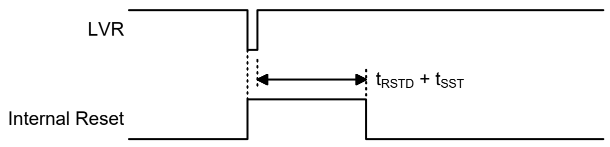
Bit 1 **LRF**: LVRC register software reset flag
Refer to the Low Voltage Reset section.

Bit 0 **WRF**: WDTC register software reset flag
Refer to the Watchdog Timer Control Register section.

Low Voltage Reset – LVR

The microcontroller contains a low voltage reset circuit in order to monitor the supply voltage of the device and provides an MCU reset when the value falls below a certain predefined level.

If the supply voltage of the device drops to within a range of 0.9V~V_{LVR} such as might occur when changing the battery in battery powered applications, the LVR will automatically reset the device internally and the LVRF bit in the RSTFC register will also be set to 1. For a valid LVR signal, a low supply voltage, i.e., a voltage in the range between 0.9V~V_{LVR} must exist for a time greater than that specified by t_{LVR} in the LVR Electrical Characteristics. If the low supply voltage state does not exceed this value, the LVR will ignore the low supply voltage and will not perform a reset function. The actual t_{LVR} value can be selected by the TLVR1~TLVR0 bits in the TLVRC register. If the LVS7~LVS0 bits are set to 01011010B, the LVR function is enabled with a fixed V_{LVR} of 2.1V. If the LVS7~LVS0 bits are set to 10100101B, the LVR function is disabled. If the LVS7~LVS0 bits are changed to any other values by environmental noise, the LVR will reset the device after a delay time, t_{SRESET}. When this happens, the LRF bit in the RSTFC register will be set to 1. After power-on the register will have the value of 01011010B. Note that the LVR function will be automatically disabled when the device enters the SLEEP or IDLE mode.



Low Voltage Reset Timing Chart

- **LVRC Register**

Bit	7	6	5	4	3	2	1	0
Name	LVS7	LVS6	LVS5	LVS4	LVS3	LVS2	LVS1	LVS0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	1	0	1	1	0	1	0

Bit 7~0 **LVS7~LVS0**: LVR voltage select
01011010: 2.1V
10100101: LVR disable
Other values: Generates an MCU reset

When an actual low voltage condition as specified above occurs, an MCU reset will be generated. The reset operation will be activated after the low voltage condition keeps more than a t_{LVR} time. In this situation the register contents will remain the same after such a reset occurs.

Any register value, other than 01011010B and 10100101B values, will also result in the generation of an MCU reset. The reset operation will be activated after a delay time, t_{SRESET}. However in this situation the register contents will be reset to the POR value.

- **TLVRC Register**

Bit	7	6	5	4	3	2	1	0
Name	—	—	—	—	—	—	TLVR1	TLVR0
R/W	—	—	—	—	—	—	R/W	R/W
POR	—	—	—	—	—	—	0	1

Bit 7~2 Unimplemented, read as “0”

Bit 1~0 **TLVR1~TLVR0**: Minimum low voltage width to reset time (t_{LVR})
00: (7~8)×t_{LIRC}
01: (31~32)×t_{LIRC}
10: (63~64)×t_{LIRC}
11: (127~128)×t_{LIRC}

- **RSTFC Register**

Bit	7	6	5	4	3	2	1	0
Name	—	—	—	—	RSTF	LVRF	LRF	WRF
R/W	—	—	—	—	R/W	R/W	R/W	R/W
POR	—	—	—	—	0	x	0	0

“x”: unknown

Bit 7~4 Unimplemented, read as “0”

Bit 3 **RSTF**: Reset control register software reset flag
Refer to the $\overline{\text{RES}}$ Pin Reset section.

Bit 2 **LVRF**: LVR function reset flag
0: Not occurred
1: Occurred

This bit is set to 1 when a specific low voltage reset condition occurs. This bit can only be cleared to zero by application program.

Bit 1 **LRF**: LVRC register software reset flag
0: Not occurred
1: Occurred
This bit is set high if the LVRC register contains any non-defined LVR voltage register values. This in effect acts like a software reset function. This bit can only be cleared to zero by application program.

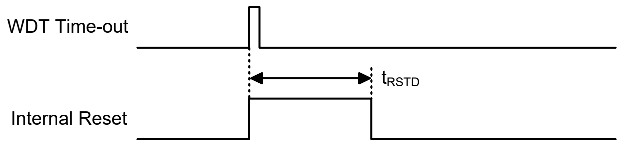
Bit 0 **WRF**: WDTC register software reset flag
Refer to the Watchdog Timer Control Register section.

IAP Reset

When a specific value of "55H" is written into the FC1 register, a reset signal will be generated to reset the whole device. Refer to the In Application Programming section for more associated details.

Watchdog Time-out Reset during Normal Operation

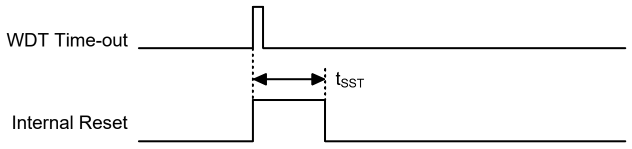
The Watchdog time-out Reset during normal operation is the same as the LVR Reset except that the Watchdog time-out flag TO will be set to "1".



WDT Time-out Reset during Normal Operation Timing Chart

Watchdog Time-out Reset during SLEEP or IDLE Mode

The Watchdog time-out Reset during SLEEP or IDLE Mode is a little different from other kinds of reset. Most of the conditions remain unchanged except that the Program Counter and the Stack Pointer will be cleared to "0" and the TO flag will be set to "1". Refer to the System Start Up Time Characteristics for t_{SST} details.



WDT Time-out Reset during SLEEP or IDLE Mode Timing Chart

Reset Initial Conditions

The different types of reset described affect the reset flags in different ways. These flags, known as PDF and TO are located in the status register and are controlled by various microcontroller operations, such as the SLEEP or IDLE Mode function or Watchdog Timer. The reset flags are shown in the table:

TO	PDF	Reset Conditions
0	0	Power-on reset
u	u	$\overline{\text{RES}}$ or LVR reset during FAST or SLOW Mode operation
1	u	WDT time-out reset during FAST or SLOW Mode operation
1	1	WDT time-out reset during IDLE or SLEEP Mode operation

"u": unchanged

The following table indicates the way in which the various components of the microcontroller are affected after a power-on reset occurs.

Item	Condition After Reset
Program Counter	Reset to zero
Interrupts	All interrupts will be disabled
WDT, Time Bases	Cleared after reset, WDT begins counting
Timer Modules	Timer Modules will be turned off
Input/Output Ports	I/O ports will be setup as inputs
Stack Pointer	Stack Pointer will point to the top of the stack

The different kinds of resets all affect the internal registers of the microcontroller in different ways. To ensure reliable continuation of normal program execution after a reset occurs, it is important to know what condition the microcontroller is in after a particular reset occurs. The following table describes how each type of reset affects the microcontroller internal registers. Note that where more than one package type exists the table will reflect the situation for the larger package type.

Register	Power-On Reset	$\overline{\text{RES}}$ Reset (Normal Operation)	$\overline{\text{RES}}$ Reset (IDLE/SLEEP)	WDT Time-out (Normal Operation)	WDT Time-out (IDLE/SLEEP)
IAR0	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
MP0	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
IAR1	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
MP1L	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
MP1H	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
ACC	xxxx xxxx	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu
PCL	0000 0000	0000 0000	0000 0000	0000 0000	0000 0000
TBLP	xxxx xxxx	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu
TBLH	xxxx xxxx	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu
TBHP	---x xxxx	---u uuuu	---u uuuu	---u uuuu	-uuu uuuu
STATUS	xx00 xxxx	uuuu uuuu	uu01 uuuu	uu1u uuuu	uu11 uuuu
IAR2	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
MP2L	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
MP2H	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
RSTFC	---- 0x00	---- uuuu	---- uuuu	---- uuuu	---- uuuu
LVRC	0101 1010	0101 1010	0101 1010	0101 1010	uuuu uuuu
TLVRC	---- --01	---- --01	---- --01	---- --01	---- --uu
SCC	000- --00	000- --00	000- --00	000- --00	uuu- --uu
HIRCC	---- --01	---- --01	---- --01	---- --01	---- --uu
PA	1111 1111	1111 1111	1111 1111	1111 1111	uuuu uuuu
PAC	1111 1-11	1111 1-11	1111 1-11	1111 1-11	uuuu u-uu
PAPU	0000 0-00	0000 0-00	0000 0-00	0000 0-00	uuuu u-uu
PAWU	0000 0-00	0000 0-00	0000 0-00	0000 0-00	uuuu u-uu
PB	1111 1111	1111 1111	1111 1111	1111 1111	uuuu uuuu
PBC	1111 1111	1111 1111	1111 1111	1111 1111	uuuu uuuu
PBPU	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
PC	1111 1111	1111 1111	1111 1111	1111 1111	uuuu uuuu
PCC	1111 1111	1111 1111	1111 1111	1111 1111	uuuu uuuu
PCPU	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
PD	---- -111	---- -111	---- -111	---- -111	---- -uuu
PDC	---- -111	---- -111	---- -111	---- -111	---- -uuu
PDPU	---- -000	---- -000	---- -000	---- -000	---- -uuu
PAS0	00-- 00--	00-- 00--	00-- 00--	00-- 00--	uu-- uu--
PAS1	--00 0000	--00 0000	--00 0000	--00 0000	--uu uuuu

Register	Power-On Reset	$\overline{\text{RES}}$ Reset (Normal Operation)	$\overline{\text{RES}}$ Reset (IDLE/SLEEP)	WDT Time-out (Normal Operation)	WDT Time-out (IDLE/SLEEP)
PBS0	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
PBS1	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
PCS1	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
IECC	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
WDTC	0101 0011	0101 0011	0101 0011	0101 0011	uuuu uuuu
TB0C	0--- -000	0--- -000	0--- -000	0--- -000	u--- -uuu
TB1C	0--- -000	0--- -000	0--- -000	0--- -000	u--- -uuu
SADC0	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
SADC1	0000 -000	0000 -000	0000 -000	0000 -000	uuuu -uuu
SADC2	---- 00--	---- 00--	---- 00--	---- 00--	---- uu--
SADOL	xxxx ----	xxxx ----	xxxx ----	xxxx ----	uuuu ---- (ADRFS=0)
					uuuu uuuu (ADRFS=1)
SADOH	xxxx xxxx	xxxx xxxx	xxxx xxxx	xxxx xxxx	uuuu uuuu (ADRFS=0)
					---- uuuu (ADRFS=1)
INTC0	-000 0000	-000 0000	-000 0000	-000 0000	-uuu uuuu
INTC1	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
INTC2	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
MFI0	--00 --00	--00 --00	--00 --00	--00 --00	--uu --uu
MFI1	--00 --00	--00 --00	--00 --00	--00 --00	--uu --uu
MFI2	--00 --00	--00 --00	--00 --00	--00 --00	--uu --uu
INTEG	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
CRCCR	---- ---0	---- ---0	---- ---0	---- ---0	---- ---u
CRCIN	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
CRCDL	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
CRCDH	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
EEAL	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
EEAH	---- ---0	---- ---0	---- ---0	---- ---0	---- ---u
EED	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
FC0	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
FC1	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
FC2	---- --00	---- --00	---- --00	---- --00	---- --uu
FARL	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
FARH	---0 0000	---0 0000	---0 0000	---0 0000	---u uuuu
FD0L	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
FD0H	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
FD1L	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
FD1H	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
FD2L	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
FD2H	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
FD3L	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
FD3H	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
CTMC0	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
CTMC1	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
CTMDL	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
CTMDH	---- --00	---- --00	---- --00	---- --00	---- --uu
CTMAL	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu

Register	Power-On Reset	$\overline{RES}$ Reset (Normal Operation)	$\overline{RES}$ Reset (IDLE/SLEEP)	WDT Time-out (Normal Operation)	WDT Time-out (IDLE/SLEEP)
CTMAH	---- --00	---- --00	---- --00	---- --00	---- --uu
STM0C0	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
STM0C1	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
STM0DL	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
STM0DH	---- --00	---- --00	---- --00	---- --00	---- --uu
STM0AL	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
STM0AH	---- --00	---- --00	---- --00	---- --00	---- --uu
STM1C0	0000 0---	0000 0---	0000 0---	0000 0---	uuuu u---
STM1C1	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
STM1DL	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
STM1DH	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
STM1AL	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
STM1AH	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
STM1RP	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
STKPTR	0--- -000	0--- -000	0--- -000	0--- -000	u--- -uuu
PCRL	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
PCRH	---0 0000	---0 0000	---0 0000	---0 0000	---u uuuu
DA0L	1000 0010	1000 0010	1000 0010	1000 0010	uuuu uuuu
DA0H	--00 0000	--00 0000	--00 0000	--00 0000	--uu uuuu
DA1L	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
DA1H	--10 0000	--10 0000	--10 0000	--10 0000	--uu uuuu
DAOPC	1100 ---0	1100 ---0	1100 ---0	1100 ---0	uuuu ---u
OPVOS	0-10 0000	0-10 0000	0-10 0000	0-10 0000	u-uu uuuu
PSCR	---- --00	---- --00	---- --00	---- --00	---- --uu
RSTC	1010 1010	1010 1010	1010 1010	1010 1010	uuuu uuuu
EEC	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu

Note: "u" stands for unchanged
"x" stands for unknown
"-" stands for unimplemented

Input/Output Ports

Holtek microcontrollers offer considerable flexibility on their I/O ports. With the input or output designation of every pin fully under user program control, pull-high selections for all ports and wake-up selections on certain pins, the user is provided with an I/O structure to meet the needs of a wide range of application possibilities.

The device provides bidirectional input/output lines labeled with port names PA~PD. These I/O ports are mapped to the RAM Data Memory with specific addresses as shown in the Special Purpose Data Memory table. All of these I/O ports can be used for input and output operations. For input operation, these ports are non-latching, which means the inputs must be ready at the T2 rising edge of instruction "MOV A, [m]", where m denotes the port address. For output operation, all the data is latched and remains unchanged until the output latch is rewritten.

Register Name	Bit							
	7	6	5	4	3	2	1	0
PA	PA7	PA6	PA5	PA4	PA3	PA2	PA1	PA0
PAC	PAC7	PAC6	PAC5	PAC4	PAC3	—	PAC1	PAC0
PAPU	PAPU7	PAPU6	PAPU5	PAPU4	PAPU3	—	PAPU1	PAPU0
PAWU	PAWU7	PAWU6	PAWU5	PAWU4	PAWU3	—	PAWU1	PAWU0

Register Name	Bit							
	7	6	5	4	3	2	1	0
PB	PB7	PB6	PB5	PB4	PB3	PB2	PB1	D0
PBC	PBC7	PBC6	PBC5	PBC4	PBC3	PBC2	PBC1	D0
PBPU	PBPU7	PBPU6	PBPU5	PBPU4	PBPU3	PBPU2	PBPU1	D0
PC	PC7	PC6	PC5	D4	D3	D2	D1	PC0
PCC	PCC7	PCC6	PCC5	D4	D3	D2	D1	PCC0
PCPU	PCPU7	PCPU6	PCPU5	D4	D3	D2	D1	PCPU0
PD	—	—	—	—	—	PD2	PD1	PD0
PDC	—	—	—	—	—	PDC2	PDC1	PDC0
PDPU	—	—	—	—	—	PDPU2	PDPU1	PDPU0

"—": Unimplemented, read as "0"

I/O Logic Function Register List

Pull-high Resistors

Many product applications require pull-high resistors for their switch inputs usually requiring the use of an external resistor. To eliminate the need for these external resistors, all I/O pins, when configured as an input have the capability of being connected to an internal pull-high resistor. These pull-high resistors are selected using the relevant pull-high control registers and are implemented using weak PMOS transistors.

Note that the pull-high resistor can be controlled by the relevant pull-high control register only when the pin-shared functional pin is selected as a digital input or NMOS output. Otherwise, the pull-high resistors cannot be enabled.

• PxPU Register

Bit	7	6	5	4	3	2	1	0
Name	PxPU7	PxPU6	PxPU5	PxPU4	PxPU3	PxPU2	PxPU1	PxPU0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

PxPUn: I/O Port x Pin pull-high function control
0: Disable
1: Enable
The PxPUn bit is used to control the pin pull-high function. Here the "x" is the Port name which can be A, B, C or D. However, the actual available bits for each I/O Port may be different.
Note: 1. The PA2 pin is without pull-high function.
2. The PA7, PC0, PC5~PC6 and PD0~PD1 lines which are internally connected to the CAN Bus, the corresponding bits should remain unchanged after power-on reset. In addition, the pull-high control bits denoted as "Dn" for ports B and C should also remain unchanged after power-on reset.

Port A Wake-up

The HALT instruction forces the microcontroller into the SLEEP or IDLE Mode which preserves power, a feature that is important for battery and other low-power applications. Various methods exist to wake-up the microcontroller, one of which is to change the logic condition on one of the Port A pins from high to low. This function is especially suitable for applications that can be woken up via external switches. Each pin on Port A can be selected individually to have this wake-up feature using the PAWU register.

Note that the wake-up function can be controlled by the wake-up control register only when the pin is selected as a general purpose input and the MCU enters the IDLE or SLEEP mode.

• **PAWU Register**

Bit	7	6	5	4	3	2	1	0
Name	PAWU7	PAWU6	PAWU5	PAWU4	PAWU3	—	PAWU1	PAWU0
R/W	R/W	R/W	R/W	R/W	R/W	—	R/W	R/W
POR	0	0	0	0	0	—	0	0

Bit 7~0 **PAWU7~PAWU0**: Port A pin Wake-up function control
0: Disable
1: Enable
Note: 1. The PA2 pin is without wake-up function.
2. The PA7 line, which is internally connected to the CAN Bus, the corresponding bit should remain unchanged after power-on reset.

I/O Port Control Registers

Each I/O port has its own control register which controls the input/output configuration. With this control register, each CMOS output or input can be reconfigured dynamically under software control. Each pin of the I/O ports is directly mapped to a bit in its associated port control register. For the I/O pin to function as an input, the corresponding bit of the control register must be written as a "1". This will then allow the logic state of the input pin to be directly read by instructions. When the corresponding bit of the control register is written as a "0", the I/O pin will be setup as a CMOS output. If the pin is currently setup as an output, instructions can still be used to read the output register. However, it should be noted that the program will in fact only read the status of the output data latch and not the actual logic status of the output pin when the IECM is set to "0".

• **PxC Register**

Bit	7	6	5	4	3	2	1	0
Name	PxC7	PxC6	PxC5	PxC4	PxC3	PxC2	PxC1	PxC0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	1	1	1	1	1	1	1	1

PxCn: I/O Port x Pin type selection
0: Output
1: Input

The PxCn bit is used to control the pin type selection. Here the "x" is the Port name which can be A, B, C or D. However, the actual available bits for each I/O Port may be different.

Note that for the PA7, PC0, PC5~PC6 and PD0~PD1 lines which are internally connected to the CAN Bus, the corresponding bits should be properly configured after power-on reset to implement correct interconnection. The PA7 and PC0 lines should be set as inputs and the PC5~PC6 and PD0~PD1 line should be set as outputs. In addition, the port control bits denoted as "Dn" for ports B and C should be cleared to 0 to set the corresponding pin as an output after power-on reset. This can prevent the device from consuming power due to input floating states for any unbonded pins.

Pin-shared Functions

The flexibility of the microcontroller range is greatly enhanced by the use of pins that have more than one function. Limited numbers of pins can force serious design constraints on designers but by supplying pins with multi-functions, many of these difficulties can be overcome. For these pins, the desired function of the multi-function I/O pins is selected by a series of registers via the application program control.

Pin-shared Function Selection Registers

The limited number of supplied pins in a package can impose restrictions on the amount of functions a certain device can contain. However by allowing the same pins to share several different functions and providing a means of function selection, a wide range of different functions can be incorporated into even relatively small package sizes. The device includes Port "x" output function selection register "n", labeled as PxSn, which can select the desired functions of the multi-function pin-shared pins.

The most important point to note is to make sure that the desired pin-shared function is properly selected and also deselected. For most pin-shared functions, to select the desired pin-shared function, the pin-shared function should first be correctly selected using the corresponding pin-shared control register. After that the corresponding peripheral functional setting should be configured and then the peripheral function can be enabled. However, a special point must be noted for some digital input pins, such as INTn, xTCKn, STPnI, etc., which share the same pin-shared control configuration with their corresponding general purpose I/O functions when setting the relevant pin-shared control bit fields. To select these pin functions, in addition to the necessary pin-shared control and peripheral functional setup aforementioned, they must also be setup as input by setting the corresponding bit in the I/O port control register. To correctly deselect the pin-shared function, the peripheral function should first be disabled and then the corresponding pin-shared function control register can be modified to select other pin-shared functions.

Register Name	Bit							
	7	6	5	4	3	2	1	0
PAS0	PAS07	PAS06	—	—	PAS03	PAS02	—	—
PAS1	—	—	PAS15	PAS14	PAS13	PAS12	PAS11	PAS10
PBS0	PBS07	PBS06	PBS05	PBS04	PBS03	PBS02	D1	D0
PBS1	PBS17	PBS16	PBS15	PBS14	PBS13	PBS12	PBS11	PBS10
PCS1	PCS17	PCS16	D5	D4	D3	D2	D1	D0

Pin-shared Function Selection Register List

• PAS0 Register

Bit	7	6	5	4	3	2	1	0
Name	PAS07	PAS06	—	—	PAS03	PAS02	—	—
R/W	R/W	R/W	—	—	R/W	R/W	—	—
POR	0	0	—	—	0	0	—	—

Bit 7~6 **PAS07~PAS06**: PA3 pin-shared function selection
00: PA3/INT0/CTCK
01: PA3/INT0/CTCK
10: PA3/INT0/CTCK
11: CTP

Bit 5~4 Unimplemented, read as "0"

Bit 3~2 **PAS03~PAS02**: PA1 pin-shared function selection
00: PA1
01: PA1
10: AN10
11: OPA2P

Bit 1~0 Unimplemented, read as "0"

- **PAS1 Register**

Bit	7	6	5	4	3	2	1	0
Name	—	—	PAS15	PAS14	PAS13	PAS12	PAS11	PAS10
R/W	—	—	R/W	R/W	R/W	R/W	R/W	R/W
POR	—	—	0	0	0	0	0	0

Bit 7~6 Unimplemented, read as "0"

Bit 5~4 **PAS15~PAS14**: PA6 pin-shared function selection
00: PA6
01: PA6
10: PA6
11: Reserved

Bit 3~2 **PAS13~PAS12**: PA5 pin-shared function selection
00: PA5
01: CTPB
10: OPA0P
11: AN6

Bit 1~0 **PAS11~PAS10**: PA4 pin-shared function selection
00: PA4/STCK0
01: PA4/STCK0
10: STP0
11: AN9

- **PBS0 Register**

Bit	7	6	5	4	3	2	1	0
Name	PBS07	PBS06	PBS05	PBS04	PBS03	PBS02	D1	D0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7~6 **PBS07~PBS06**: PB3 pin-shared function selection
00: PB3/STCK1
01: PB3/STCK1
10: PB3/STCK1
11: AN3

Bit 5~4 **PBS05~PBS04**: PB2 pin-shared function selection
00: PB2/STP1I
01: PB2/STP1I
10: PB2/STP1I
11: AN2

Bit 3~2 **PBS03~PBS02**: PB1 pin-shared function selection
00: PB1/STP0I
01: PB1/STP0I
10: PB1/STP0I
11: AN1

Bit 1~0 **D1~D0**: Reserved bits, should remain unchanged after power-on reset

• PBS1 Register

Bit	7	6	5	4	3	2	1	0
Name	PBS17	PBS16	PBS15	PBS14	PBS13	PBS12	PBS11	PBS10
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7~6 **PBS17~PBS16**: PB7 pin-shared function selection
00: PB7
01: PB7
10: PB7
11: OPA1P

Bit 5~4 **PBS15~PBS14**: PB6 pin-shared function selection
00: PB6/INT1
01: PB6/INT1
10: Reserved
11: AN7

Bit 3~2 **PBS13~PBS12**: PB5 pin-shared function selection
00: PB5
01: VREF
10: STP1B
11: AN5

Bit 1~0 **PBS11~PBS10**: PB4 pin-shared function selection
00: PB4
01: PB4
10: STP1
11: AN4

• PCS1 Register

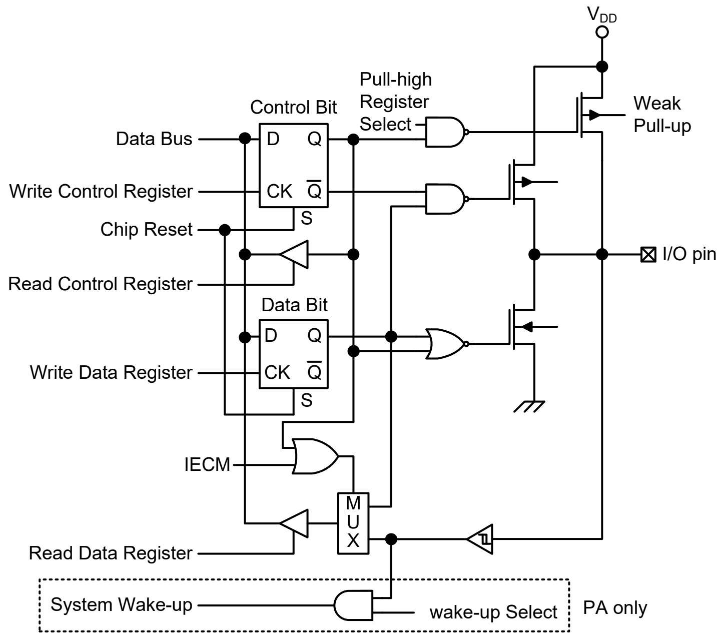
Bit	7	6	5	4	3	2	1	0
Name	PCS17	PCS16	D5	D4	D3	D2	D1	D0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7~6 **PCS17~PCS16**: PC7 pin-shared function selection
00: PC7
01: PC7
10: Reserved
11: STP0B

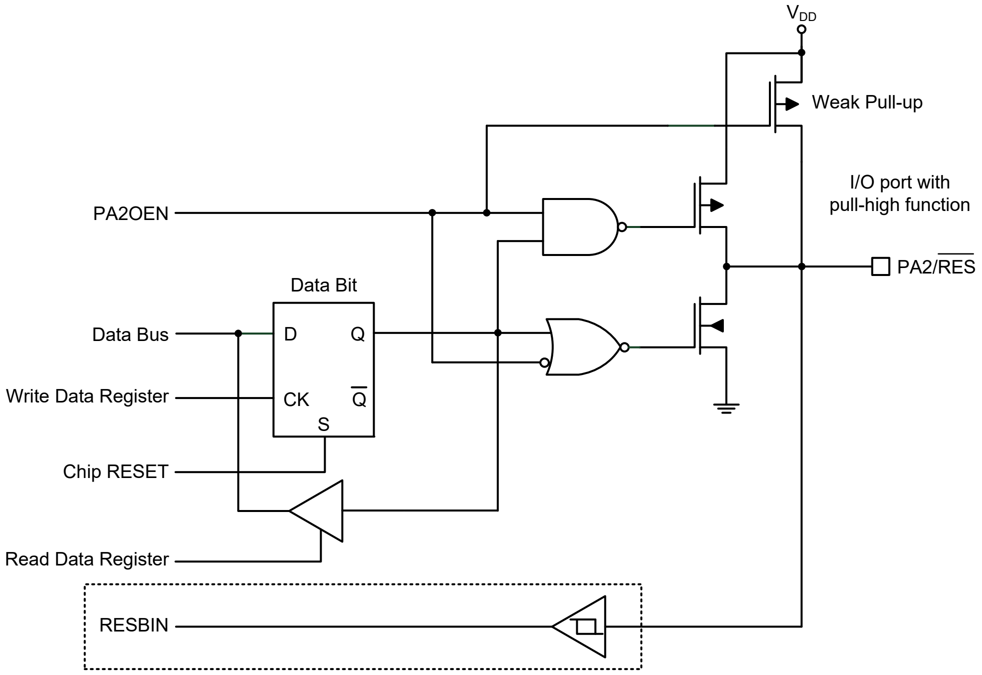
Bit 5~0 **D5~D0**: Reserved bits, should remain unchanged after power-on reset

I/O Pin Structures

The accompanying diagram illustrates the internal structures of the I/O logic function. As the exact logical construction of the I/O pin will differ from this drawing, it is supplied as a guide only to assist with the functional understanding of the logic function I/O pins. The wide range of pin-shared structures does not permit all types to be shown.



Logic Function Input/Output Structure



Note: The PA2OEN value is determined by the RSTC register value. Please refer to the RSTC register description for details.

PA2 Port Output Structure

READ PORT Function

The READ PORT function is used to read data from I/O pins, which is specially designed for the IEC 60730 self-diagnostic test on the I/O function and A/D paths. After the self-diagnostic test on the I/O function and A/D paths are completed, users must disable the READ PORT function immediately. In cases other than the I/O function and A/D path self-diagnostic test, it is strongly recommended to disable the READ PORT function to avoid affecting other peripheral functions and causing unexpected consequences.

There is a register, IECC, which is used to control the READ PORT function. When a specific data pattern, "11001010", is written into the IECC register, the internal signal named IECM will be set high to enable the READ PORT function. If the READ PORT function is enabled, the reading path is from the I/O pins. The value on the corresponding pins will be passed to the accumulator ACC when the read port instruction "mov a, Px" is executed, where the "x" stands for the corresponding I/O port name. However, when the IECC register content is set to any other values rather than "11001010", the IECM internal signal will be cleared to 0 to disable the READ PORT function, and the reading path will be from the data latch or I/O pins. If the READ PORT function is disabled, the pin function will operate as the selected pin-shared function.

• IECC Register

Bit	7	6	5	4	3	2	1	0
Name	IECS7	IECS6	IECS5	IECS4	IECS3	IECS2	IECS1	IECS0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

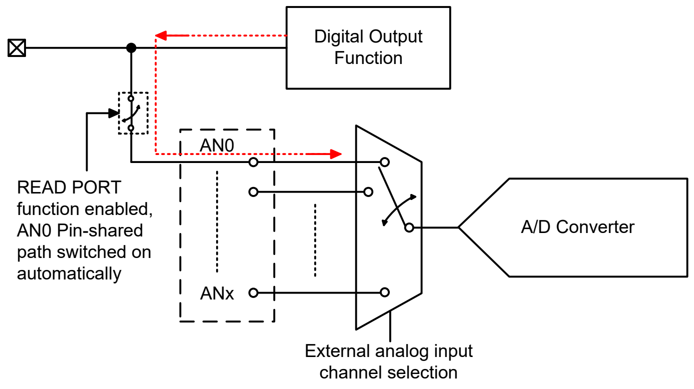
Bit 7~0 **IECS7~IECS0**: READ PORT function enable control bit 7~ bit 0
11001010: IECM=1 – READ PORT function is enabled
Others: IECM=0 – READ PORT function is disabled

READ PORT Function	Disabled		Enabled	
Port Control Register Bit – PxC.n	1	0	1	0
I/O Function	Pin value	Data latch value	Pin value	
A/D Function	0			

Note: The value on the above table is the content of the ACC register after "mov a, Px" instruction is executed where "x" means the relevant port name.

The additional function of the READ PORT mode is to check the A/D path. When the READ PORT function is disabled, the A/D path from the external pin to the internal analog input will be switched off if the A/D input pin function is not selected by the corresponding selection bits. For the MCU with A/D converter channels, the desired A/D channel can be switched on by properly configuring the external analog input channel selection bits in the A/D Control Register together with the corresponding analog input pin function is selected. However, the additional function of the READ PORT mode is to force the A/D path to be switched on. As shown in the following example, when the AN0 is selected as the analog input channel as the READ PORT function is enabled, the AN0 analog input path will be switched on even if the AN0 analog input pin function is not selected. In this way, the AN0 analog input path can be examined by internally connecting the digital output on this shared pin with the AN0 analog input pin switch and then converting the corresponding digital data without any external analog input voltage connected.

Note that the A/D converter reference voltage should be equal to the I/O power supply voltage when examining the A/D path using the READ PORT function.


Digital Output Function
AN0
ANx
READ PORT function enabled, AN0 Pin-shared path switched on automatically
A/D Converter
External analog input channel selection

A/D Channel Input Path Internal Connection

Programming Considerations

Within the user program, one of the things first to consider is port initialisation. After a reset, all of the I/O data and port control registers will be set high. This means that all I/O pins will default to an input state, the level of which depends on the other connected circuitry and whether pull-high selections have been chosen. If the port control registers are then programmed to setup some pins as outputs, these output pins will have an initial high output value unless the associated port data registers are first programmed. Selecting which pins are inputs and which are outputs can be achieved byte-wide by loading the correct values into the appropriate port control register or by programming individual bits in the port control register using the “SET [m].i” and “CLR [m].i” instructions. Note that when using these bit control instructions, a read-modify-write operation takes place. The microcontroller must first read in the data on the entire port, modify it to the required new bit values and then rewrite this data back to the output ports.

Port A has the additional capability of providing wake-up functions. When the device is in the SLEEP or IDLE Mode, various methods are available to wake the device up. One of these is a high to low transition of any of the Port A pins. Single or multiple pins on Port A can be setup to have this function.

Timer Modules – TM

One of the most fundamental functions in any microcontroller device is the ability to control and measure time. To implement time related functions the device includes several Timer Modules, generally abbreviated to the name TM. The TMs are multi-purpose timing units and serve to provide operations such as Timer/Counter, Input Capture, Compare Match Output and Single Pulse Output as well as being the functional unit for the generation of PWM signals. Each of the TMs has two interrupts. The addition of input and output pins for each TM ensures that users are provided with timing units with a wide and flexible range of features.

The common features of the different TM types are described here with more detailed information provided in the individual Compact and Standard Type TM sections.

Introduction

The device contains three TMs and each individual TM is categorised as a certain type, namely Compact Type TM or Standard Type TM. Although similar in nature, the different TM types vary in their feature complexity. The common features to all of the Compact and Standard TMs will be described in this section and the detailed operation regarding each of the TM types will be described in separate sections. The main features and differences between the two types of TMs are summarised in the accompanying table.

TM Function	CTM	STM
Timer/Counter	√	√
Input Capture	—	√
Compare Match Output	√	√
PWM Output	√	√
Single Pulse Output	—	√
PWM Alignment	Edge	Edge
PWM Adjustment Period & Duty	Duty or Period	Duty or Period

TM Function Summary

TM Operation

The different types of TM offer a diverse range of functions, from simple timing operations to PWM signal generation. The key to understanding how the TM operates is to see it in terms of a free running counter whose value is then compared with the value of pre-programmed internal comparators. When the free running counter has the same value as the pre-programmed comparator, known as a compare match situation, a TM interrupt signal will be generated which can clear the counter and perhaps also change the condition of the TM output pin. The internal TM counter is driven by a user selectable clock source, which can be an internal clock or an external pin.

TM Clock Source

The clock source which drives the main counter in each TM can originate from various sources. The selection of the required clock source is implemented using the xTnCK2~xTnCK0 bits in the xTMn control registers, where “x” stands for C or S type TM and “n” stands for the specific TM serial number. For the CTM there is no serial number “n” in the relevant pin or control register bits since there is only one CTM in the device. The clock source can be a ratio of the system clock, f_{SYS}, or the internal high clock, f_H, the f_{SUB} clock source or the external xTCKn pin. The xTCKn pin clock source is used to allow an external signal to drive the TM as an external clock source for event counting.

TM Interrupts

Each Compact or Standard type TMs has two internal interrupts, one for each of the internal comparator A or comparator P, which generate a TM interrupt when a compare match condition occurs. When a TM interrupt is generated, it can be used to clear the counter and also to change the state of the TM output pins.

TM External Pins

Each of the TMs, irrespective of what type, has one TM input pin, with the label xTCKn while the STMn has another input pin with the label STPnI. The xTMn input pin, xTCKn, is essentially a clock source for the xTMn and is selected using the xTnCK2~xTnCK0 bits in the xTMnC0 register. This external TM input pin allows an external clock source to drive the internal TM. The xTCKn input pin can be chosen to have either a rising or falling active edge. The STCKn pin is also used as the external trigger input pin in single pulse output mode for the STMn.

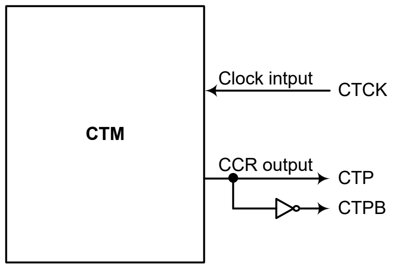
The STMn has another input pin, STPnI, which is the capture input whose active edge can be a rising edge, a falling edge or both rising and falling edges and the active edge transition type is selected using the STnIO1~STnIO0 bits in the STMnC1 register.

The TMs each have two output pins with the label xTPn and the xTPnB. When the TM is in the Compare Match Output Mode, this xTPn pin can be controlled by the TM to switch to a high or low level or to toggle when a compare match situation occurs. The xTPnB pin output is the inverted signal of the xTPn. The external output pins are also the pins where the TM generates the PWM output waveform.

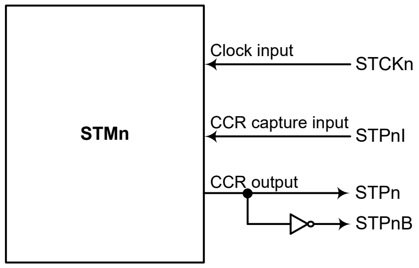
As the TM input and output pins are pin-shared with other functions, the TM input and output functions must first be selected using relevant pin-shared function selection. The details of the pin-shared function selection are described in the pin-shared function section.

CTM		STM0		STM1	
Input	Output	Input	Output	Input	Output
CTCK	CTP, CTPB	STCK0, STP0I	STP0, STP0B	STCK1, STP1I	STP1, STP1B

TM External Pins



CTM Function Pin Block Diagram

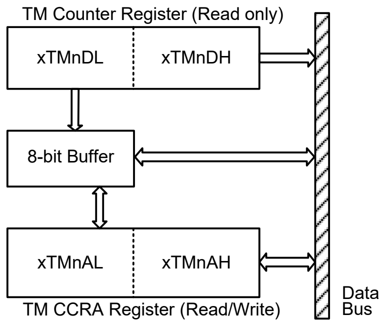


STM Function Pin Block Diagram (n=0~1)

Programming Considerations

The TM Counter Registers and the Capture/Compare CCRA registers all have a low and high byte structure. The high bytes can be directly accessed, but as the low bytes can only be accessed via an internal 8-bit buffer, reading or writing to these register pairs must be carried out in a specific way. The important point to note is that data transfer to and from the 8-bit buffer and its related low byte only takes place when a write or read operation to its corresponding high byte is executed.

As the CCRA registers are implemented in the way shown in the following diagram and accessing these register pairs is carried out in a specific way as described above, it is recommended to use the “MOV” instruction to access the CCRA low byte registers, named xTMnAL, using the following access procedures. Accessing the CCRA low byte registers without following these access procedures will result in unpredictable values.

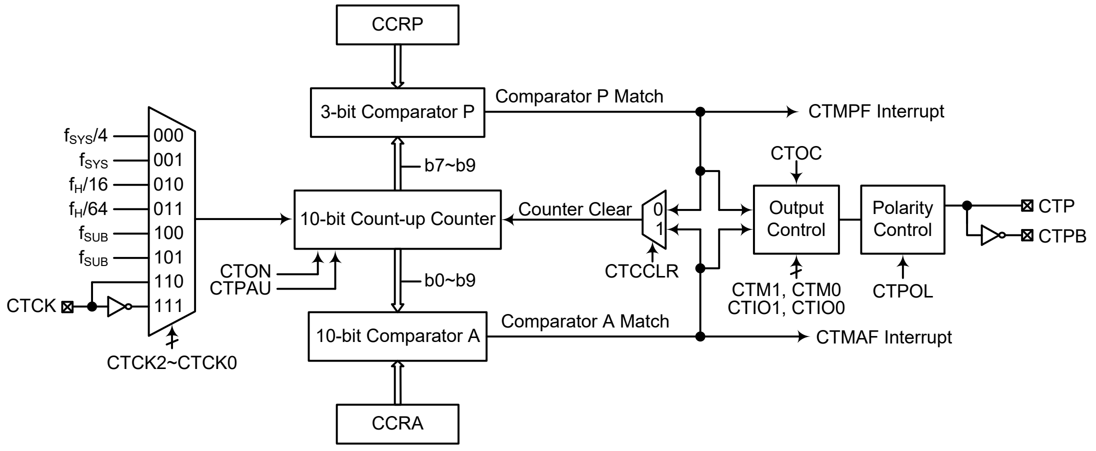

TM Counter Register (Read only)
xTMnDL
xTMnDH
8-bit Buffer
xTMnAL
xTMnAH
TM CCRA Register (Read/Write)
Data Bus

The following steps show the read and write procedures:

- Writing Data to CCRA
  - ♦ Step 1. Write data to Low Byte xTMnAL
    - – Note that here data is only written to the 8-bit buffer.
  - ♦ Step 2. Write data to High Byte xTMnAH
    - – Here data is written directly to the high byte registers and simultaneously data is latched from the 8-bit buffer to the Low Byte registers.
- Reading Data from the Counter Registers and CCRA
  - ♦ Step 1. Read data from the High Byte xTMnDH, xTMnAH
    - – Here data is read directly from the High Byte registers and simultaneously data is latched from the Low Byte register into the 8-bit buffer.
  - ♦ Step 2. Read data from the Low Byte xTMnDL, xTMnAL
    - – This step reads data from the 8-bit buffer.

Compact Type TM – CTM

The Compact TM type contains three operating modes, which are Compare Match Output, Timer/Event Counter and PWM Output modes. The Compact TM can also be controlled with an external input pin and can drive two external output pins.



Note: 1. As the CTM external pins are pin-shared with other functions, so before using the CTM function the relevant pin-shared function registers must be set properly to enable the CTM pin function. The CTCK pin, if used, must also be set as an input by setting the corresponding bits in the port control register.

2. The CTPB is the inverted signal of the CTP.

10-bit Compact Type TM Block Diagram

Compact Type TM Operation

The size of Compact TM is 10-bit wide and its core is a 10-bit count-up counter which is driven by a user selectable internal or external clock source. There are also two internal comparators with the names, Comparator A and Comparator P. These comparators will compare the value in the counter with CCRP and CCRA registers. The CCRP is three bits wide whose value is compared with the highest 3 bits in the counter while the CCRA is the 10 bits and therefore compares with all counter bits.

The only way of changing the value of the 10-bit counter using the application program, is to clear the counter by changing the CTON bit from low to high. The counter will also be cleared automatically by a counter overflow or a compare match with one of its associated comparators. When these conditions occur, a CTM interrupt signal will also usually be generated. The Compact Type TM can operate in a number of different operational modes, can be driven by different clock sources including an input pin and can also control two output pins. All operating setup conditions are selected using relevant internal registers.

Compact Type TM Register Description

Overall operation of the Compact TM is controlled using several registers. A read only register pair exists to store the internal counter 10-bit value, while a read/write register pair exists to store the internal 10-bit CCRA value. The remaining two registers are control registers which setup the different operating and control modes as well as the 3 CCRP bits.

Register Name	Bit							
	7	6	5	4	3	2	1	0
CTMC0	CTPAU	CTCK2	CTCK1	CTCK0	CTON	CTRP2	CTRP1	CTRP0
CTMC1	CTM1	CTM0	CTIO1	CTIO0	CTOC	CTPOL	CTDPX	CTCCLR
CTMDL	D7	D6	D5	D4	D3	D2	D1	D0
CTMDH	—	—	—	—	—	—	D9	D8
CTMAL	D7	D6	D5	D4	D3	D2	D1	D0
CTMAH	—	—	—	—	—	—	D9	D8

10-bit Compact TM Register List

• CTMC0 Register

Bit	7	6	5	4	3	2	1	0
Name	CTPAU	CTCK2	CTCK1	CTCK0	CTON	CTRP2	CTRP1	CTRP0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7 **CTPAU**: CTM Counter Pause Control
0: Run
1: Pause

The counter can be paused by setting this bit high. Clearing the bit to zero restores normal counter operation. When in a Pause condition the CTM will remain powered up and continue to consume power. The counter will retain its residual value when this bit changes from low to high and resume counting from this value when the bit changes to a low value again

Bit 6~4 **CTCK2~CTCK0**: Select CTM Counter clock
000: $f_{SYS}/4$
001: f_{SYS}
010: $f_H/16$
011: $f_H/64$
100: f_{SUB}
101: f_{SUB}
110: CTCK rising edge clock
111: CTCK falling edge clock

These three bits are used to select the clock source for the CTM. The external pin clock source can be chosen to be active on the rising or falling edge. The clock source f_{SYS} is the system clock, while f_H and f_{SUB} are other internal clocks, the details of which can be found in the Operating Modes and System Clocks section.

Bit 3 **CTON**: CTM Counter On/Off Control
0: Off
1: On

This bit controls the overall on/off function of the CTM. Setting the bit high enables the counter to run, clearing the bit disables the CTM. Clearing this bit to zero will stop the counter from counting and turn off the CTM which will reduce its power consumption. When the bit changes state from low to high the internal counter value will be reset to zero, however when the bit changes from high to low, the internal counter will retain its residual value until the bit returns high again.

If the CTM is in the Compare Match Output Mode or the PWM Output Mode then the CTM output pin will be reset to its initial condition, as specified by the CTOC bit, when the CTON bit changes from low to high.

Bit 2~0 **CTRP2~CTRP0**: CTM CCRP 3-bit register, compared with the CTM Counter bit 9 ~ bit 7

Comparator P Match Period=
000: 1024 CTM clocks
001: 128 CTM clocks
010: 256 CTM clocks
011: 384 CTM clocks
100: 512 CTM clocks
101: 640 CTM clocks
110: 768 CTM clocks
111: 896 CTM clocks

These three bits are used to setup the value on the internal CCRP 3-bit register, which are then compared with the internal counter's highest three bits. The result of this comparison can be selected to clear the internal counter if the CTCCLR bit is set to zero. Setting the CTCCLR bit to zero ensures that a compare match with the CCRP values will reset the internal counter. As the CCRP bits are only compared with the highest three counter bits, the compare values exist in 128 clock cycle multiples. Clearing all three bits to zero is in effect allowing the counter to overflow at its maximum value.

• CTMC1 Register

Bit	7	6	5	4	3	2	1	0
Name	CTM1	CTM0	CTIO1	CTIO0	CTOC	CTPOL	CTDPX	CTCCLR
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7~6 **CTM1~CTM0**: Select CTM Operating Mode
00: Compare Match Output Mode
01: Undefined
10: PWM Output Mode
11: Timer/Counter Mode

These bits setup the required operating mode for the CTM. To ensure reliable operation the CTM should be switched off before any changes are made to the CTM1 and CTM0 bits. In the Timer/Counter Mode, the CTM output pin state is undefined.

Bit 5~4 **CTIO1~CTIO0**: Select CTM external pin function

Compare Match Output Mode
00: No change
01: Output low
10: Output high
11: Toggle output

PWM Output Mode
00: PWM output inactive state
01: PWM output active state
10: PWM output
11: Undefined

Timer/Counter Mode
Unused

These two bits are used to determine how the CTM external pin changes state when a certain condition is reached. The function that these bits select depends upon in which mode the CTM is running.

In the Compare Match Output Mode, the CTIO1 and CTIO0 bits determine how the CTM output pin changes state when a compare match occurs from the Comparator A. The CTM output pin can be setup to switch high, switch low or to toggle its present state when a compare match occurs from the Comparator A. When the bits are both zero, then no change will take place on the output. The initial value of the CTM output pin should be setup using the CTOC bit in the CTMC1 register. Note that the output level requested by the CTIO1 and CTIO0 bits must be different from the initial value setup using the CTOC bit otherwise no change will occur on the CTM output pin when a compare match occurs. After the CTM output pin changes state, it can be reset to its initial level by changing the level of the CTON bit from low to high.

In the PWM Output Mode, the CTIO1 and CTIO0 bits determine how the CTM output pin changes state when a certain compare match condition occurs. The PWM output function is modified by changing these two bits. It is necessary to change the values of the CTIO1 and CTIO0 bits only after the CTM has been switched off. Unpredictable PWM outputs will occur if the CTIO1 and CTIO0 bits are changed when the CTM is running.

Bit 3 **CTOC**: CTM CTP Output control

Compare Match Output Mode
0: Initial low
1: Initial high

PWM Output Mode
0: Active low
1: Active high

This is the output control bit for the CTM output pin. Its operation depends upon whether CTM is being used in the Compare Match Output Mode or in the PWM Output Mode. It has no effect if the CTM is in the Timer/Counter Mode. In the Compare Match Output Mode it determines the logic level of the CTM output pin before a compare match occurs. In the PWM Output Mode it determines if the PWM signal is active high or active low.

Bit 2 **CTPOL**: CTP Output polarity control
0: Non-invert
1: Invert

This bit controls the polarity of the CTP output pin. When the bit is set high the CTM output pin will be inverted and not inverted when the bit is zero. It has no effect if the CTM is in the Timer/Counter Mode.

Bit 1 **CTDPX**: CTM PWM duty/period control
0: CCRP – period; CCRA – duty
1: CCRP – duty; CCRA – period

This bit determines which of the CCRA and CCRP registers are used for period and duty control of the PWM waveform.

Bit 0 **CTCCLR**: Select CTM Counter Clear condition
0: Comparator P match
1: Comparator A match

This bit is used to select the method which clears the counter. Remember that the Compact TM contains two comparators, Comparator A and Comparator P, either of which can be selected to clear the internal counter. With the CTCCLR bit set high, the counter will be cleared when a compare match occurs from the Comparator A. When the bit is low, the counter will be cleared when a compare match occurs from the Comparator P or with a counter overflow. A counter overflow clearing method can only be implemented if the CCRP bits are all cleared to zero. The CTCCLR bit is not used in the PWM Output Mode.

- **CTMDL Register**

Bit	7	6	5	4	3	2	1	0
Name	D7	D6	D5	D4	D3	D2	D1	D0
R/W	R	R	R	R	R	R	R	R
POR	0	0	0	0	0	0	0	0

Bit 7~0 **D7~D0**: CTM Counter Low Byte Register bit 7 ~ bit 0
CTM 10-bit Counter bit 7 ~ bit 0

- **CTMDH Register**

Bit	7	6	5	4	3	2	1	0
Name	—	—	—	—	—	—	D9	D8
R/W	—	—	—	—	—	—	R	R
POR	—	—	—	—	—	—	0	0

Bit 7~2 Unimplemented, read as "0"
Bit 1~0 **D9~D8**: CTM Counter High Byte Register bit 1 ~ bit 0
CTM 10-bit Counter bit 9 ~ bit 8

- **CTMAL Register**

Bit	7	6	5	4	3	2	1	0
Name	D7	D6	D5	D4	D3	D2	D1	D0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7~0 **D7~D0**: CTM CCRA Low Byte Register bit 7 ~ bit 0
CTM 10-bit CCRA bit 7 ~ bit 0

- **CTMAH Register**

Bit	7	6	5	4	3	2	1	0
Name	—	—	—	—	—	—	D9	D8
R/W	—	—	—	—	—	—	R/W	R/W
POR	—	—	—	—	—	—	0	0

Bit 7~2 Unimplemented, read as "0"
Bit 1~0 **D9~D8**: CTM CCRA High Byte Register bit 1 ~ bit 0
CTM 10-bit CCRA bit 9 ~ bit 8

Compact Type TM Operating Modes

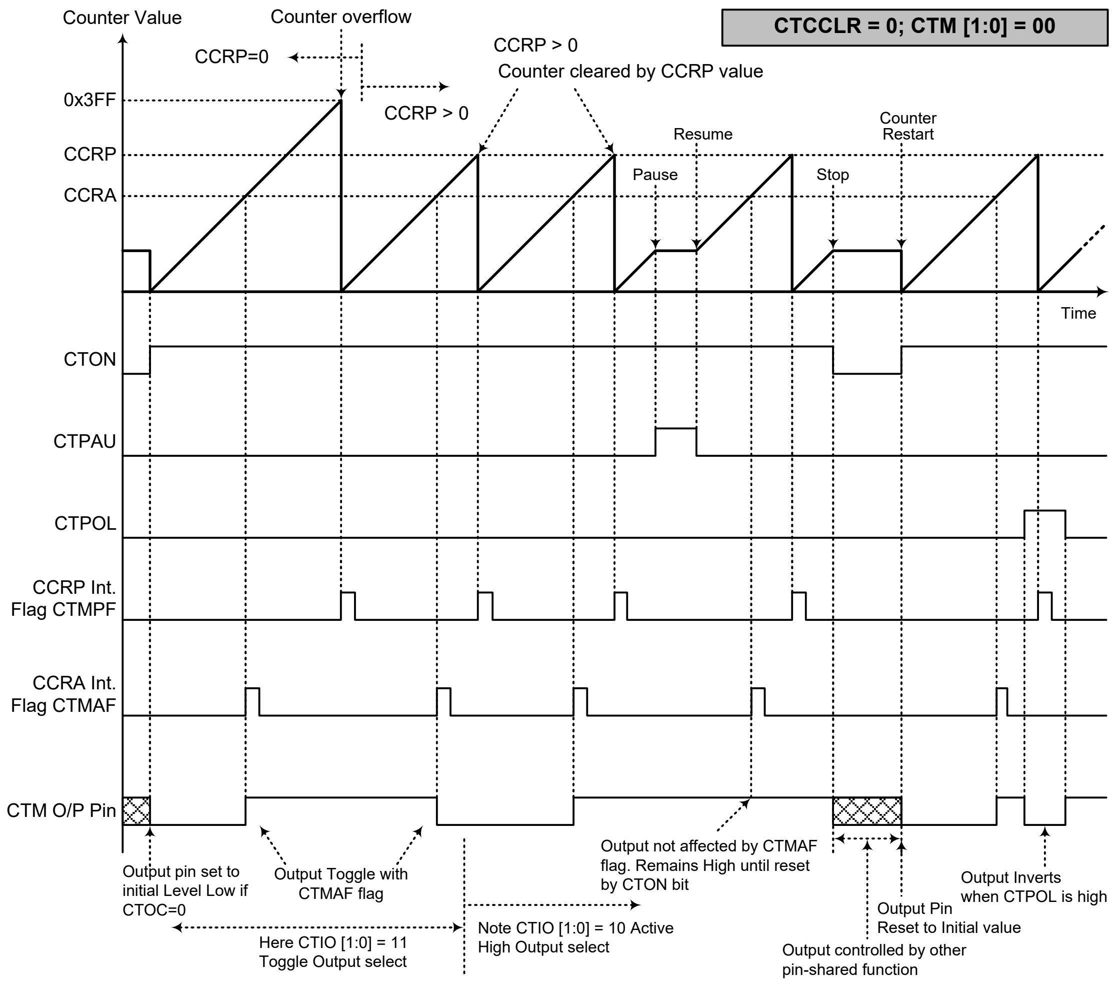
The Compact Type TM can operate in one of three operating modes, Compare Match Output Mode, PWM Output Mode or Timer/Counter Mode. The operating mode is selected using the CTM1 and CTM0 bits in the CTMC1 register.

Compare Match Output Mode

To select this mode, bits CTM1 and CTM0 in the CTMC1 register, should be set to 00 respectively. In this mode once the counter is enabled and running it can be cleared by three methods. These are a counter overflow, a compare match from Comparator A and a compare match from Comparator P. When the CTCCLR bit is low, there are two ways in which the counter can be cleared. One is when a compare match from Comparator P, the other is when the CCRP bits are all zero which allows the counter to overflow. Here both CTMAF and CTMPF interrupt request flags for Comparator A and Comparator P respectively, will both be generated.

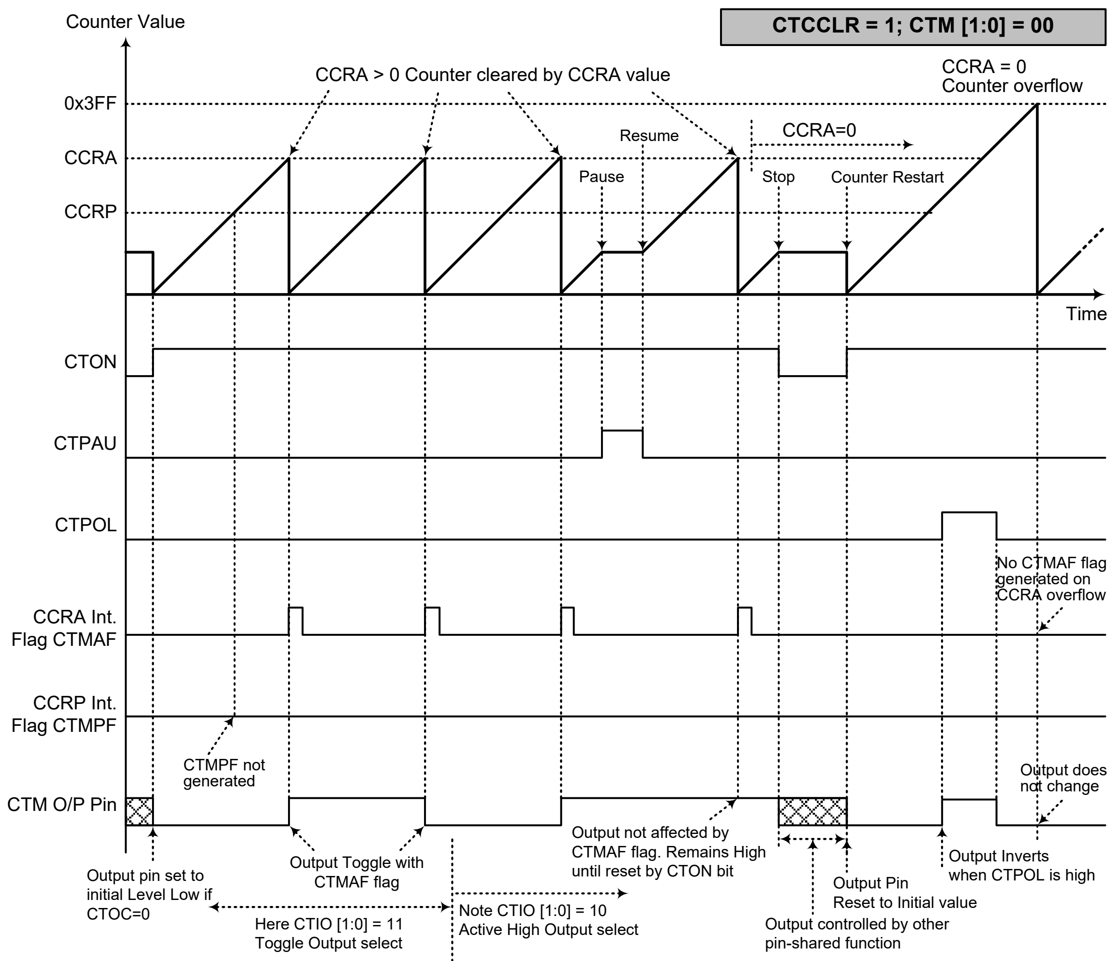
If the CTCCLR bit in the CTMC1 register is high then the counter will be cleared when a compare match occurs from Comparator A. However, here only the CTMAF interrupt request flag will be generated even if the value of the CCRP bits is less than that of the CCRA registers. Therefore when CTCCLR is high no CTMPF interrupt request flag will be generated. If the CCRA bits are all zero, the counter will overflow when it reaches its maximum 10-bit, 3FF Hex, value, however here the CTMAF interrupt request flag will not be generated.

As the name of the mode suggests, after a comparison is made, the CTM output pin will change state. The CTM output pin condition however only changes state when a CTMAF interrupt request flag is generated after a compare match occurs from Comparator A. The CTMPF interrupt request flag, generated from a compare match occurs from Comparator P, will have no effect on the CTM output pin. The way in which the CTM output pin changes state are determined by the condition of the CTIO1 and CTIO0 bits in the CTMC1 register. The CTM output pin can be selected using the CTIO1 and CTIO0 bits to go high, to go low or to toggle from its present condition when a compare match occurs from Comparator A. The initial condition of the CTM output pin, which is setup after the CTON bit changes from low to high, is setup using the CTOC bit. Note that if the CTIO1 and CTIO0 bits are zero then no pin change will take place.


CTCCLR = 0; CTM [1:0] = 00
Counter Value
Counter overflow
CCRP=0
CCRP > 0
CCRP > 0
Counter cleared by CCRP value
0x3FF
CCRP
CCRA
Pause
Resume
Stop
Counter Restart
Time
CTON
CTPAU
CTPOL
CCRP Int. Flag CTMPF
CCRA Int. Flag CTMAF
CTM O/P Pin
Output pin set to initial Level Low if CTOC=0
Output Toggle with CTMAF flag
Here CTIO [1:0] = 11 Toggle Output select
Note CTIO [1:0] = 10 Active High Output select
Output not affected by CTMAF flag. Remains High until reset by CTON bit
Output controlled by other pin-shared function
Output Pin Reset to Initial value
Output Inverts when CTPOL is high

Compare Match Output Mode – CTCCLR=0

Note: 1. With CTCCLR=0, a Comparator P match will clear the counter
2. The CTM output pin controlled only by the CTMAF flag
3. The output pin reset to initial state by a CTON bit rising edge


Counter Value
CTCCLR = 1; CTM [1:0] = 00
CCRA > 0 Counter cleared by CCRA value
CCRA = 0
Counter overflow
0x3FF
CCRA
CCRP
Resume
CCRA=0
Pause
Stop
Counter Restart
Time
CTON
CTPAU
CTPOL
No CTMAF flag generated on CCRA overflow
CCRA Int. Flag CTMAF
CCRP Int. Flag CTMPF
CTMPF not generated
Output does not change
CTM O/P Pin
Output not affected by CTMAF flag. Remains High until reset by CTON bit
Output Toggle with CTMAF flag
Output Inverts when CTPOL is high
Output Pin Reset to Initial value
Output pin set to initial Level Low if CTOC=0
Here CTIO [1:0] = 11 Toggle Output select
Note CTIO [1:0] = 10 Active High Output select
Output controlled by other pin-shared function

Compare Match Output Mode – CTCCLR=1

Note: 1. With CTCCLR=1, a Comparator A match will clear the counter
2. The CTM output pin controlled only by the CTMAF flag
3. The output pin reset to initial state by a CTON rising edge
4. The CTMPF flags is not generated when CTCCLR=1

Timer/Counter Mode

To select this mode, bits CTM1 and CTM0 in the CTMC1 register should be set to 11 respectively. The Timer/Counter Mode operates in an identical way to the Compare Match Output Mode generating the same interrupt flags. The exception is that in the Timer/Counter Mode the CTM output pin is not used. Therefore the above description and Timing Diagrams for the Compare Match Output Mode can be used to understand its function. As the CTM output pin is not used in this mode, the pin can be used as a normal I/O pin or other pin-shared function.

PWM Output Mode

To select this mode, bits CTM1 and CTM0 in the CTMC1 register should be set to 10 respectively. The PWM function within the CTM is useful for applications which require functions such as motor control, heating control, illumination control etc. By providing a signal of fixed frequency but of varying duty cycle on the CTM output pin, a square wave AC waveform can be generated with varying equivalent DC RMS values.

As both the period and duty cycle of the PWM waveform can be controlled, the choice of generated waveform is extremely flexible. In the PWM Output Mode, the CTCCLR bit has no effect on the PWM operation. Both of the CCRA and CCRP registers are used to generate the PWM waveform, one register is used to clear the internal counter and thus control the PWM waveform frequency, while the other one is used to control the duty cycle. Which register is used to control either frequency or duty cycle is determined using the CTDPX bit in the CTMC1 register. The PWM waveform frequency and duty cycle can therefore be controlled by the values in the CCRA and CCRP registers.

An interrupt flag, one for each of the CCRA and CCRP, will be generated when a compare match occurs from either Comparator A or Comparator P. The CTOC bit In the CTMC1 register is used to select the required polarity of the PWM waveform while the two CTIO1 and CTIO0 bits are used to enable the PWM output or to force the CTM output pin to a fixed high or low level. The CTPOL bit is used to reverse the polarity of the PWM output waveform.

- **CTM, PWM Output Mode, Edge-aligned Mode, CTDPX=0**

CCRP	001b	010b	011b	100b	101b	110b	111b	000b
Period	128	256	384	512	640	768	896	1024
Duty	CCRA							

If f_{SYS}=8MHz, CTM clock source is f_{SYS}/4, CCRP=100b, CCRA=128,

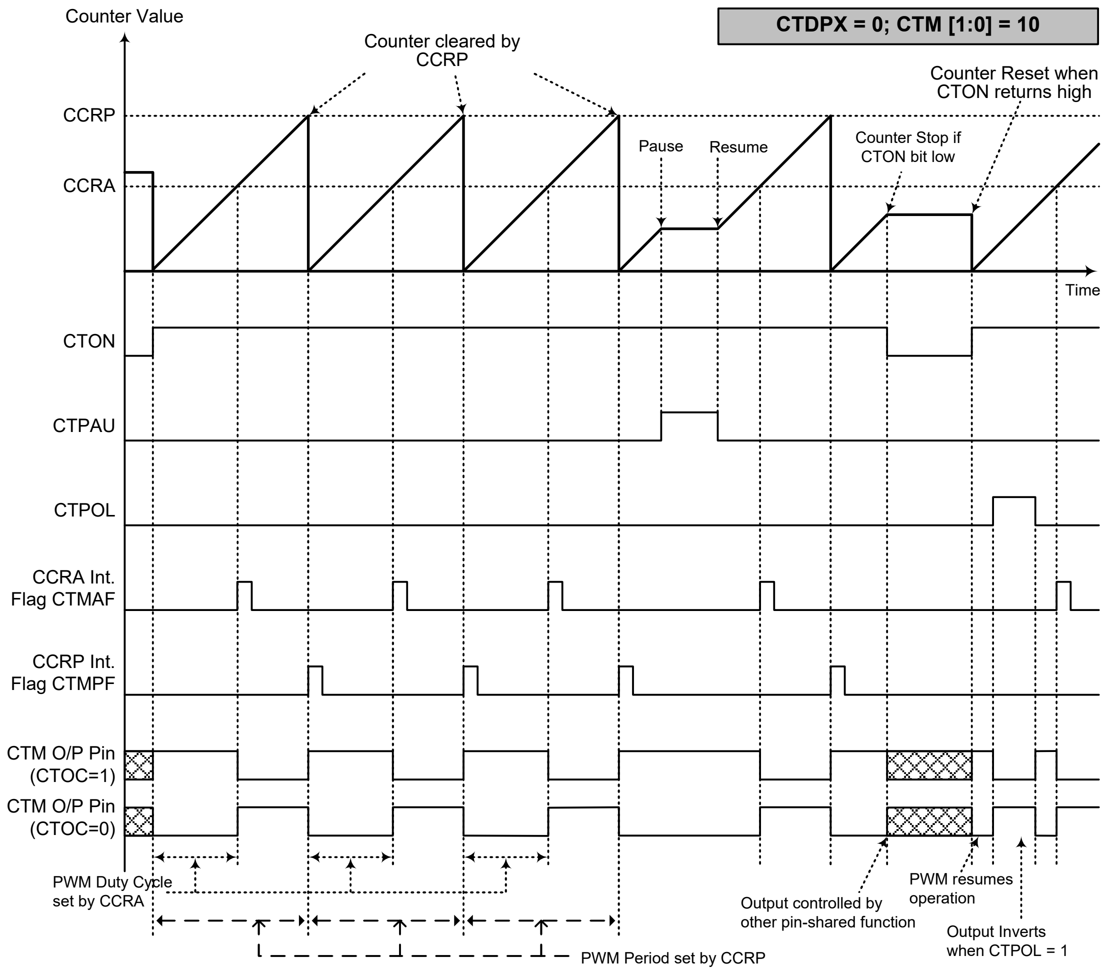
The CTM PWM output frequency=(f_{SYS}/4)/512=f_{SYS}/2048=3.9063kHz, duty=128/512=25%.

If the Duty value defined by the CCRA register is equal to or greater than the Period value, then the PWM output duty is 100%.

- **CTM, PWM Output Mode, Edge-aligned Mode, CTDPX=1**

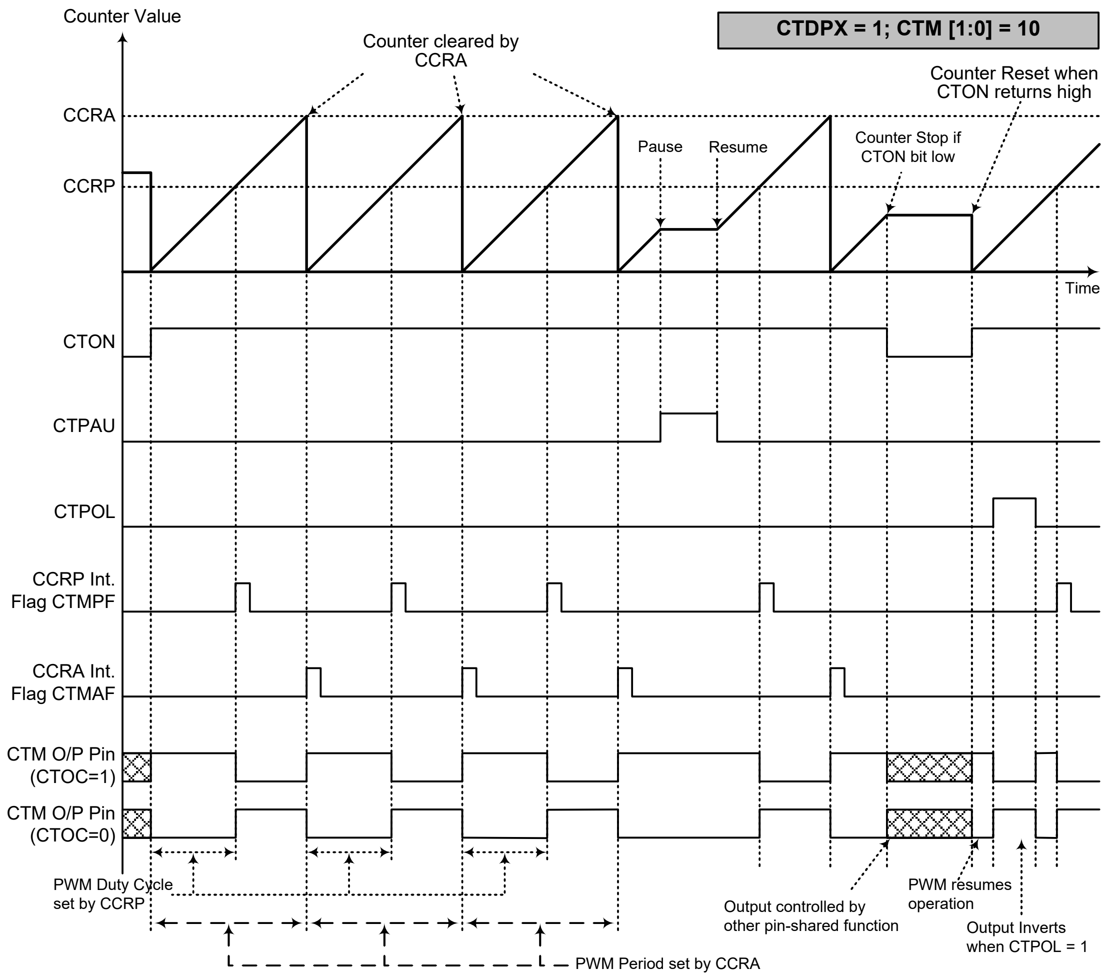
CCRP	001b	010b	011b	100b	101b	110b	111b	000b
Period	CCRA							
Duty	128	256	384	512	640	768	896	1024

The PWM output period is determined by the CCRA register value together with the CTM clock while the PWM duty cycle is defined by the CCRP register value.



PWM Output Mode – CTDPX=0

Note: 1. Here CTDPX=0 – Counter cleared by CCRP
2. A counter clear sets PWM Period
3. The internal PWM function continues running even when CTIO[1:0]=00 or 01
4. The CTCCLR bit has no influence on PWM operation



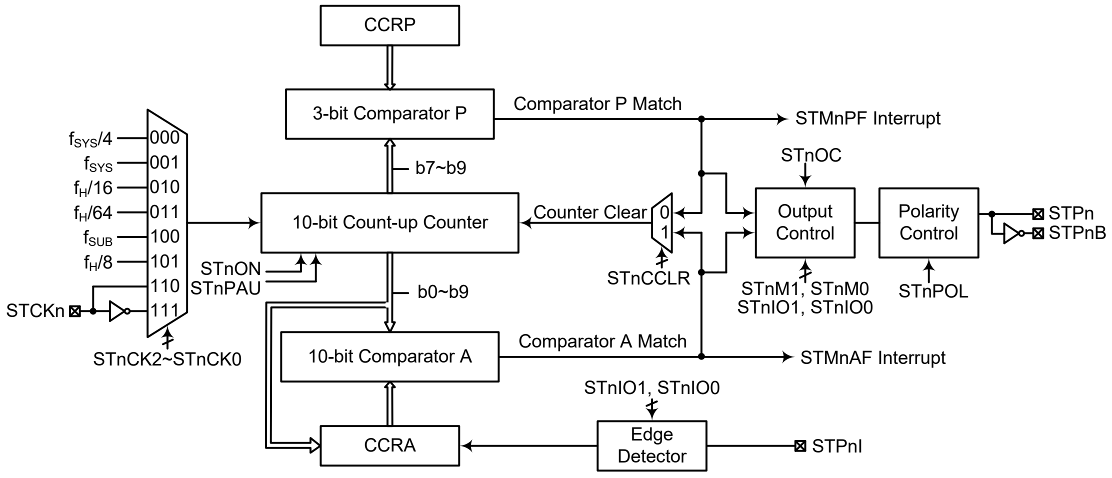
PWM Output Mode – CTDPX=1

Note: 1. Here CTDPX=1 – Counter cleared by CCRA
2. A counter clear sets PWM Period
3. The internal PWM function continues even when CTIO[1:0]=00 or 01
4. The CTCCLR bit has no influence on PWM operation

Standard Type TM – STM

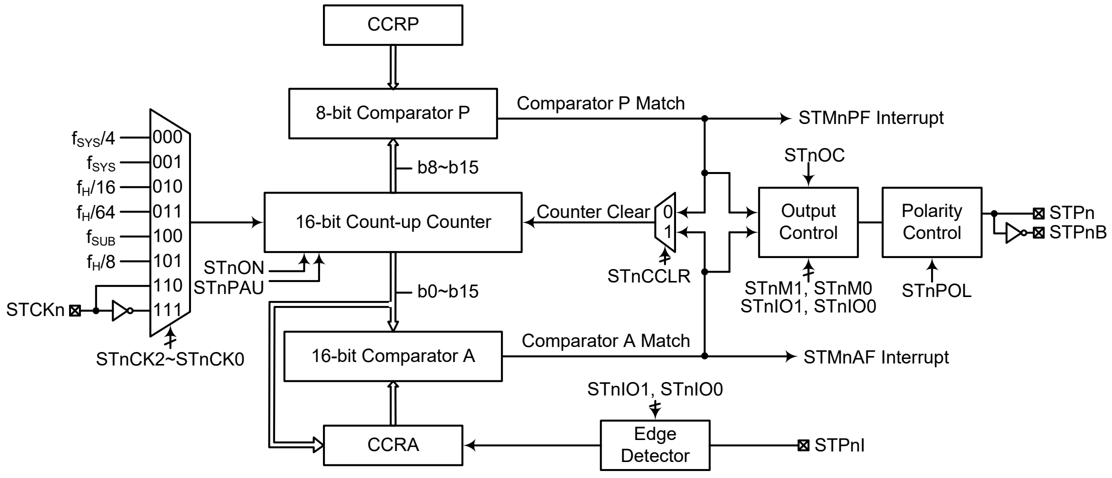
The Standard Type TM contains five operating modes, which are Compare Match Output, Timer/Event Counter, Capture Input, Single Pulse Output and PWM Output modes. The Standard TM can also be controlled with two external input pins and can drive two external output pins.

STM Core	STM Input Pins	STM Output Pins
10-bit STM (STM0)	STCK0, STP0I	STP0, STP0B
16-bit STM (STM1)	STCK1, STP1I	STP1, STP1B



Note: 1. The STMn external pins are pin-shared with other functions, so before using the STMn function, ensure that the relevant pin-shared function registers have been set properly to enable the STMn pin function. The STCKn and STPnI pin, if used, must also be set as an input by setting the corresponding bits in the port control register.
2. The STPnB is the inverted signal of the STPn.

10-bit Standard Type TM Block Diagram (n=0)



Note: 1. The STMn external pins are pin-shared with other functions, so before using the STMn function, ensure that the relevant pin-shared function registers have been set properly to enable the STMn pin function. The STCKn and STPnI pin, if used, must also be set as an input by setting the corresponding bits in the port control register.
2. The STPnB is the inverted signal of the STPn.

16-bit Standard Type TM Block Diagram (n=1)

Standard TM Operation

The size of Standard Type TM is 10/16-bit wide and its core is a 10/16-bit count-up counter which is driven by a user selectable internal or external clock source. There are also two internal comparators with the names, Comparator A and Comparator P. These comparators will compare the value in the counter with CCRP and CCRA registers. The CCRP comparator is 3/8-bit wide whose value is compared with the highest 3 or 8 bits in the counter while the CCRA is the 10/16 bits and therefore compares all counter bits.

The only way of changing the value of the 10/16-bit counter using the application program, is to clear the counter by changing the STnON bit from low to high. The counter will also be cleared automatically by a counter overflow or a compare match with one of its associated comparators. When these conditions occur, an STMn interrupt signal will also usually be generated. The Standard Type TM can operate in a number of different operational modes, can be driven by different clock sources including an input pin and can also control two output pins. All operating setup conditions are selected using relevant internal registers.

Standard Type TM Register Description

Overall operation of the Standard TM is controlled using a series of registers. A read only register pair exists to store the internal counter 10/16-bit value, while a read/write register pair exists to store the internal 10/16-bit CCRA value. The STMnRP register for the 16-bit STM is used to store the 8-bit CCRP value. The remaining two registers are control registers which setup the different operating and control modes.

Register Name	Bit							
	7	6	5	4	3	2	1	0
STMnC0	STnPAU	STnCK2	STnCK1	STnCK0	STnON	STnRP2	STnRP1	STnRP0
STMnC1	STnM1	STnM0	STnIO1	STnIO0	STnOC	STnPOL	STnDPX	STnCCLR
STMnDL	D7	D6	D5	D4	D3	D2	D1	D0
STMnDH	—	—	—	—	—	—	D9	D8
STMnAL	D7	D6	D5	D4	D3	D2	D1	D0
STMnAH	—	—	—	—	—	—	D9	D8

10-bit Standard TM Register List (n=0)

Register Name	Bit							
	7	6	5	4	3	2	1	0
STMnC0	STnPAU	STnCK2	STnCK1	STnCK0	STnON	—	—	—
STMnC1	STnM1	STnM0	STnIO1	STnIO0	STnOC	STnPOL	STnDPX	STnCCLR
STMnDL	D7	D6	D5	D4	D3	D2	D1	D0
STMnDH	D15	D14	D13	D12	D11	D10	D9	D8
STMnAL	D7	D6	D5	D4	D3	D2	D1	D0
STMnAH	D15	D14	D13	D12	D11	D10	D9	D8
STMnRP	D7	D6	D5	D4	D3	D2	D1	D0

16-bit Standard TM Register List (n=1)

• **STMnC0 Register (n=0)**

Bit	7	6	5	4	3	2	1	0
Name	STnPAU	STnCK2	STnCK1	STnCK0	STnON	STnRP2	STnRP1	STnRP0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7 **STnPAU**: STMn Counter Pause control
0: Run
1: Pause

The counter can be paused by setting this bit high. Clearing the bit to zero restores normal counter operation. When in a Pause condition the STMn will remain powered up and continue to consume power. The counter will retain its residual value when this bit changes from low to high and resume counting from this value when the bit changes to a low value again.

Bit 6~4 **STnCK2~STnCK0**: Select STMn Counter clock
000: $f_{SYS}/4$
001: f_{SYS}
010: $f_H/16$
011: $f_H/64$
100: f_{SUB}
101: f_{SUB}
110: STCKn rising edge clock
111: STCKn falling edge clock

These three bits are used to select the clock source for the STMn. The external pin clock source can be chosen to be active on the rising or falling edge. The clock source f_{SYS} is the system clock, while f_H and f_{SUB} are other internal clocks, the details of which can be found in the Operating Modes and System Clocks section.

Bit 3 **STnON**: STMn Counter On/Off control
0: Off
1: On

This bit controls the overall on/off function of the STMn. Setting the bit high enables the counter to run while clearing the bit disables the STMn. Clearing this bit to zero will stop the counter from counting and turn off the STMn which will reduce its power consumption. When the bit changes state from low to high the internal counter value will be reset to zero, however when the bit changes from high to low, the internal counter will retain its residual value until the bit returns high again. If the STMn is in the Compare Match Output Mode, PWM Output Mode or Single Pulse Mode then the STMn output pin will be reset to its initial condition, as specified by the STnOC bit, when the STnON bit changes from low to high.

Bit 2~0 **STnRP2~STnRP0**: STMn CCRP 3-bit register, compared with the STMn Counter bit 9 ~ bit 7

Comparator P Match Period=
000: 1024 STMn clocks
001: 128 STMn clocks
010: 256 STMn clocks
011: 384 STMn clocks
100: 512 STMn clocks
101: 640 STMn clocks
110: 768 STMn clocks
111: 896 STMn clocks

These three bits are used to setup the value on the internal CCRP 3-bit register, which are then compared with the internal counter's highest three bits. The result of this comparison can be selected to clear the internal counter if the STnCCLR bit is set to zero. Setting the STnCCLR bit to zero ensures that a compare match with the CCRP values will reset the internal counter. As the CCRP bits are only compared with the highest three counter bits, the compare values exist in 128 clock cycle multiples. Clearing all three bits to zero is in effect allowing the counter to overflow at its maximum value.

• **STMnC0 Register (n=1)**

Bit	7	6	5	4	3	2	1	0
Name	STnPAU	STnCK2	STnCK1	STnCK0	STnON	—	—	—
R/W	R/W	R/W	R/W	R/W	R/W	—	—	—
POR	0	0	0	0	0	—	—	—

Bit 7 **STnPAU**: STMn Counter Pause control
0: Run
1: Pause
The counter can be paused by setting this bit high. Clearing the bit to zero restores normal counter operation. When in a Pause condition the STMn will remain powered up and continue to consume power. The counter will retain its residual value when this bit changes from low to high and resume counting from this value when the bit changes to a low value again.

Bit 6~4 **STnCK2~STnCK0**: Select STMn Counter clock
000: $f_{SYS}/4$
001: f_{SYS}
010: $f_H/16$
011: $f_H/64$
100: f_{SUB}
101: f_{SUB}
110: STCKn rising edge clock
111: STCKn falling edge clock
These three bits are used to select the clock source for the STMn. The external pin clock source can be chosen to be active on the rising or falling edge. The clock source f_{SYS} is the system clock, while f_H and f_{SUB} are other internal clocks, the details of which can be found in the Operating Modes and System Clocks section.

Bit 3 **STnON**: STMn Counter On/Off control
0: Off
1: On
This bit controls the overall on/off function of the STMn. Setting the bit high enables the counter to run while clearing the bit disables the STMn. Clearing this bit to zero will stop the counter from counting and turn off the STMn which will reduce its power consumption. When the bit changes state from low to high the internal counter value will be reset to zero, however when the bit changes from high to low, the internal counter will retain its residual value until the bit returns high again. If the STMn is in the Compare Match Output Mode, PWM Output Mode or Single Pulse Mode then the STMn output pin will be reset to its initial condition, as specified by the STnOC bit, when the STnON bit changes from low to high.

Bit 2~0 Unimplemented, read as "0"

• **STMnC1 Register (n=0~1)**

Bit	7	6	5	4	3	2	1	0
Name	STnM1	STnM0	STnIO1	STnIO0	STnOC	STnPOL	STnDPX	STnCCLR
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7~6 **STnM1~STnM0**: Select STMn Operating Mode
00: Compare Match Output Mode
01: Capture Input Mode
10: PWM Output Mode or Single Pulse Output Mode
11: Timer/Counter Mode
These bits setup the required operating mode for the STMn. To ensure reliable operation the STMn should be switched off before any changes are made to the STnM1 and STnM0 bits. In the Timer/Counter Mode, the STMn output pin state is undefined.

Bit 5~4 **STnIO1~STnIO0**: Select STMn external pin function

Compare Match Output Mode
- 00: No change
- 01: Output low
- 10: Output high
- 11: Toggle output

PWM Output Mode/Single Pulse Output Mode
- 00: PWM output inactive state
- 01: PWM output active state
- 10: PWM output
- 11: Single Pulse Output

Capture Input Mode
- 00: Input capture at rising edge of STPnI
- 01: Input capture at falling edge of STPnI
- 10: Input capture at both rising/falling edge of STPnI
- 11: Input capture disabled

Timer/Counter Mode
- Unused

These two bits are used to determine how the STMn external pin changes state when a certain condition is reached. The function that these bits select depends upon in which mode the STMn is running.

In the Compare Match Output Mode, the STnIO1 and STnIO0 bits determine how the STMn output pin changes state when a compare match occurs from the Comparator A. The STMn output pin can be setup to switch high, switch low or to toggle its present state when a compare match occurs from the Comparator A. When the bits are both zero, then no change will take place on the output. The initial value of the STMn output pin should be setup using the STnOC bit in the STMnC1 register. Note that the output level requested by the STnIO1 and STnIO0 bits must be different from the initial value setup using the STnOC bit otherwise no change will occur on the STMn output pin when a compare match occurs. After the STMn output pin changes state, it can be reset to its initial level by changing the level of the STnON bit from low to high.

In the PWM Output Mode, the STnIO1 and STnIO0 bits determine how the STMn output pin changes state when a certain compare match condition occurs. The PWM output function is modified by changing these two bits. It is necessary to change the values of the STnIO1 and STnIO0 bits only after the STMn has been switched off. Unpredictable PWM outputs will occur if the STnIO1 and STnIO0 bits are changed when the STMn is running.

Bit 3 **STnOC**: STMn STPn Output control

Compare Match Output Mode
- 0: Initial low
- 1: Initial high

PWM Output Mode/Single Pulse Output Mode
- 0: Active low
- 1: Active high

This is the output control bit for the STMn output pin. Its operation depends upon whether STMn is being used in the Compare Match Output Mode or in the PWM Output Mode/Single Pulse Output Mode. It has no effect if the STMn is in the Timer/Counter Mode. In the Compare Match Output Mode it determines the logic level of the STMn output pin before a compare match occurs. In the PWM output Mode it determines if the PWM signal is active high or active low. In the Single Pulse Output Mode it determines the logic level of the STMn output pin when the STnON bit changes from low to high.

Bit 2 **STnPOL**: STMn STPn Output polarity control
0: Non-invert
1: Invert
This bit controls the polarity of the STPn output pin. When the bit is set high the STMn output pin will be inverted and not inverted when the bit is zero. It has no effect if the STMn is in the Timer/Counter Mode.

Bit 1 **STnDPX**: STMn PWM duty/period control
0: CCRP – period; CCRA – duty
1: CCRP – duty; CCRA – period
This bit determines which of the CCRA and CCRP registers are used for period and duty control of the PWM waveform.

Bit 0 **STnCCLR**: STMn Counter Clear condition selection
0: Comparator P match
1: Comparator A match
This bit is used to select the method which clears the counter. Remember that the Standard TM contains two comparators, Comparator A and Comparator P, either of which can be selected to clear the internal counter. With the STnCCLR bit set high, the counter will be cleared when a compare match occurs from the Comparator A. When the bit is low, the counter will be cleared when a compare match occurs from the Comparator P or with a counter overflow. A counter overflow clearing method can only be implemented if the CCRP bits are all cleared to zero. The STnCCLR bit is not used in the PWM Output, Single Pulse Output or Capture Input Mode.

• STMnDL Register (n=0~1)

Bit	7	6	5	4	3	2	1	0
Name	D7	D6	D5	D4	D3	D2	D1	D0
R/W	R	R	R	R	R	R	R	R
POR	0	0	0	0	0	0	0	0

Bit 7~0 **D7~D0**: STMn Counter Low Byte Register bit 7 ~ bit 0
STMn 10/16-bit Counter bit 7 ~ bit 0

• STMnDH Register (n=0)

Bit	7	6	5	4	3	2	1	0
Name	—	—	—	—	—	—	D9	D8
R/W	—	—	—	—	—	—	R	R
POR	—	—	—	—	—	—	0	0

Bit 7~2 Unimplemented, read as "0"
Bit 1~0 **D9~D8**: STMn Counter High Byte Register bit 1 ~ bit 0
STMn 10-bit Counter bit 9 ~ bit 8

• STMnDH Register (n=1)

Bit	7	6	5	4	3	2	1	0
Name	D15	D14	D13	D12	D11	D10	D9	D8
R/W	R	R	R	R	R	R	R	R
POR	0	0	0	0	0	0	0	0

Bit 7~0 **D15~D8**: STMn Counter High Byte Register bit 7 ~ bit 0
STMn 16-bit Counter bit 15 ~ bit 8

• **STMnAL Register (n=0~1)**

Bit	7	6	5	4	3	2	1	0
Name	D7	D6	D5	D4	D3	D2	D1	D0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7~0 **D7~D0**: STMn CCRA Low Byte Register bit 7 ~ bit 0
STMn 10/16-bit CCRA bit 7 ~ bit 0

• **STMnAH Register (n=0)**

Bit	7	6	5	4	3	2	1	0
Name	—	—	—	—	—	—	D9	D8
R/W	—	—	—	—	—	—	R/W	R/W
POR	—	—	—	—	—	—	0	0

Bit 7~2 Unimplemented, read as "0"
Bit 1~0 **D9~D8**: STMn CCRA High Byte Register bit 1 ~ bit 0
STMn 10-bit CCRA bit 9 ~ bit 8

• **STMnAH Register (n=1)**

Bit	7	6	5	4	3	2	1	0
Name	D15	D14	D13	D12	D11	D10	D9	D8
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7~0 **D15~D8**: STMn CCRA High Byte Register bit 7 ~ bit 0
STMn 16-bit CCRA bit 15 ~ bit 8

• **STMnRP Register (n=1)**

Bit	7	6	5	4	3	2	1	0
Name	D7	D6	D5	D4	D3	D2	D1	D0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7~0 **D7~D0**: STMn CCRP 8-bit register, compared with the STMn counter bit 15 ~ bit 8
Comparator P match period=
0: 65536 STMn clocks
1~255: (1~255)×256 STMn clocks
These eight bits are used to setup the value on the internal CCRP 8-bit register, which are then compared with the internal counter's highest eight bits. The result of this comparison can be selected to clear the internal counter if the STnCCLR bit is set to zero. Setting the STnCCLR bit to zero ensures that a compare match with the CCRP values will reset the internal counter. As the CCRP bits are only compared with the highest eight counter bits, the compare values exist in 256 clock cycle multiples. Clearing all eight bits to zero is in effect allowing the counter to overflow at its maximum value.

Standard Type TM Operation Modes

The Standard Type TM can operate in one of five operating modes, Compare Match Output Mode, PWM Output Mode, Single Pulse Output Mode, Capture Input Mode or Timer/Counter Mode. The operating mode is selected using the STnM1 and STnM0 bits in the STMnC1 register.

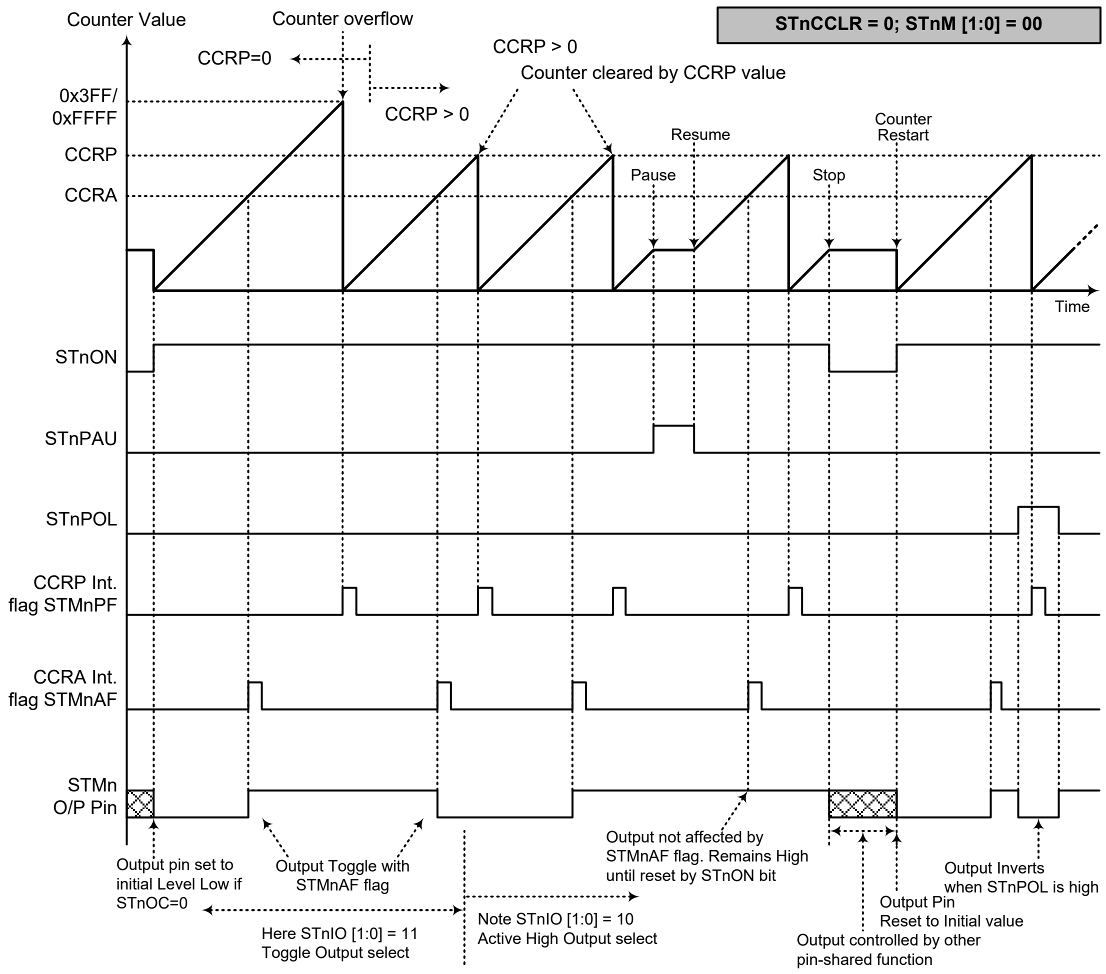
Compare Match Output Mode

To select this mode, bits STnM1 and STnM0 in the STMnC1 register, should be set to 00 respectively. In this mode once the counter is enabled and running it can be cleared by three methods. These are a counter overflow, a compare match from Comparator A and a compare match from Comparator P. When the STnCCLR bit is low, there are two ways in which the counter can be cleared. One is when a compare match from Comparator P, the other is when the CCRP bits are all zero which allows the counter to overflow. Here both STMnAF and STMnPF interrupt request flags for Comparator A and Comparator P respectively, will both be generated.

If the STnCCLR bit in the STMnC1 register is high then the counter will be cleared when a compare match occurs from Comparator A. However, here only the STMnAF interrupt request flag will be generated even if the value of the CCRP bits is less than that of the CCRA registers. Therefore when STnCCLR is high no STMnPF interrupt request flag will be generated. In the Compare Match Output Mode, the CCRA can not be set to “0”.

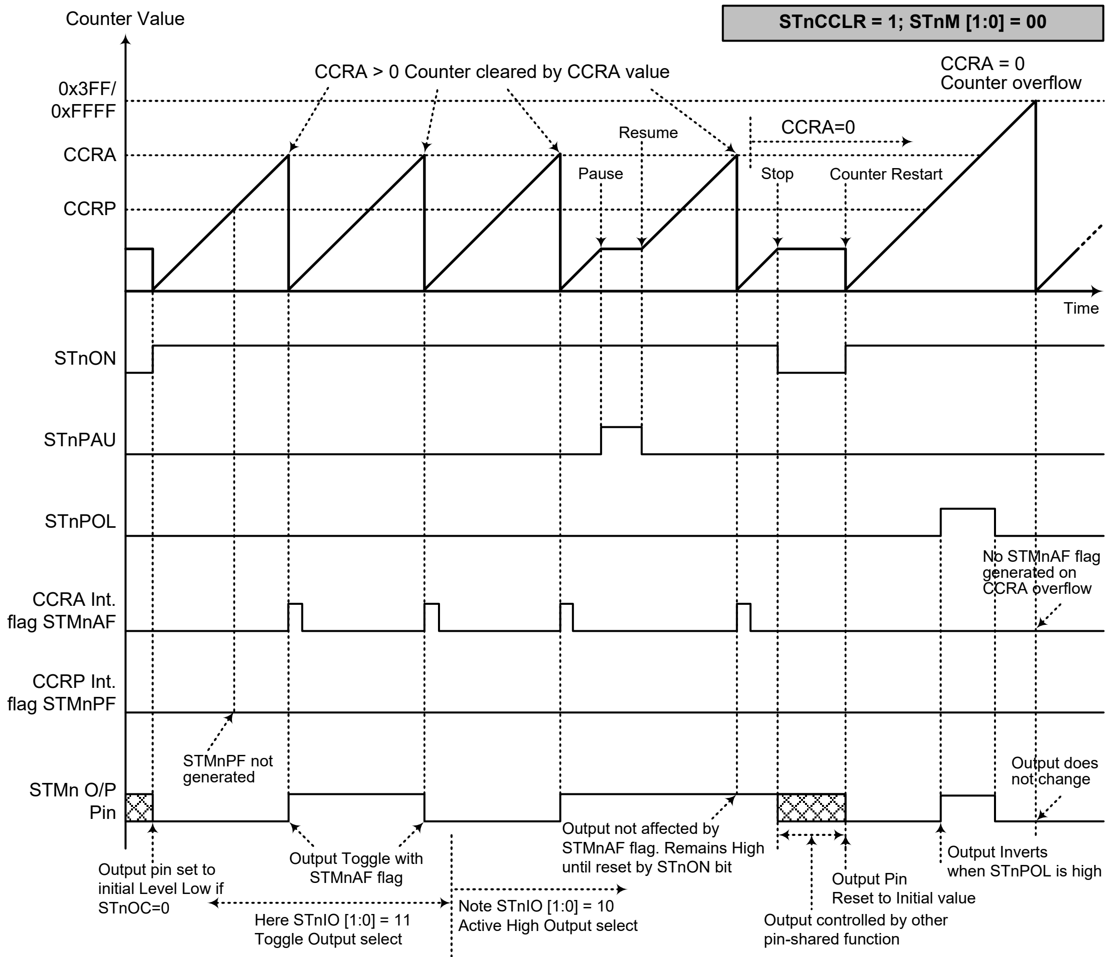
If the CCRA bits are all zero, the counter will overflow when it reaches its maximum 10-bit, 3FF Hex, or 16-bit FFFF Hex, value, however here the STMnAF interrupt request flag will not be generated.

As the name of the mode suggests, after a comparison is made, the STMn output pin, will change state. The STMn output pin condition however only changes state when an STMnAF interrupt request flag is generated after a compare match occurs from Comparator A. The STMnPF interrupt request flag, generated from a compare match occurs from Comparator P, will have no effect on the STMn output pin. The way in which the STMn output pin changes state are determined by the condition of the STnIO1 and STnIO0 bits in the STMnC1 register. The STMn output pin can be selected using the STnIO1 and STnIO0 bits to go high, to go low or to toggle from its present condition when a compare match occurs from Comparator A. The initial condition of the STMn output pin, which is setup after the STnON bit changes from low to high, is setup using the STnOC bit. Note that if the STnIO1 and STnIO0 bits are zero then no pin change will take place.


STnCCLR = 0; STnM [1:0] = 00
Counter Value
Counter overflow
CCRP=0
CCRP > 0
CCRP > 0
Counter cleared by CCRP value
0x3FF/
0xFFFF
CCRP
CCRA
Resume
Pause
Counter Restart
Stop
Time
STnON
STnPAU
STnPOL
CCRP Int. flag STMnPF
CCRA Int. flag STMnAF
STMn O/P Pin
Output pin set to initial Level Low if STnOC=0
Output Toggle with STMnAF flag
Here STnIO [1:0] = 11 Toggle Output select
Note STnIO [1:0] = 10 Active High Output select
Output not affected by STMnAF flag. Remains High until reset by STnON bit
Output controlled by other pin-shared function
Output Pin Reset to Initial value
Output Inverts when STnPOL is high

Compare Match Output Mode – STnCCLR=0

Note: 1. With STnCCLR=0, a Comparator P match will clear the counter
2. The STMn output pin is controlled only by the STMnAF flag
3. The output pin is reset to its initial state by an STnON bit rising edge
4. n=0 for 10-bit STM while n=1 for 16-bit STM


STnCCLR = 1; STnM [1:0] = 00
Counter Value
0x3FF/
0xFFFF
CCRA
CCRP
CCRA > 0 Counter cleared by CCRA value
CCRA = 0
Counter overflow
Resume
Pause
CCRA=0
Stop
Counter Restart
Time
STnON
STnPAU
STnPOL
CCRA Int.
flag STMnAF
CCRP Int.
flag STMnPF
STMn O/P
Pin
No STMnAF flag generated on CCRA overflow
STMnPF not generated
Output does not change
Output not affected by STMnAF flag. Remains High until reset by STnON bit
Output pin set to initial Level Low if STnOC=0
Output Toggle with STMnAF flag
Here STnIO [1:0] = 11 Toggle Output select
Note STnIO [1:0] = 10 Active High Output select
Output Pin Reset to Initial value
Output controlled by other pin-shared function
Output Inverts when STnPOL is high

Compare Match Output Mode – STnCCLR=1

Note: 1. With STnCCLR=1, a Comparator A match will clear the counter
2. The STMn output pin is controlled only by the STMnAF flag
3. The output pin is reset to its initial state by an STnON bit rising edge
4. An STMnPF flag is not generated when STnCCLR=1
5. n=0 for 10-bit STM while n=1 for 16-bit STM

Timer/Counter Mode

To select this mode, bits STnM1 and STnM0 in the STMnC1 register should be set to 11 respectively. The Timer/Counter Mode operates in an identical way to the Compare Match Output Mode generating the same interrupt flags. The exception is that in the Timer/Counter Mode the STMn output pin is not used. Therefore the above description and Timing Diagrams for the Compare Match Output Mode can be used to understand its function. As the STMn output pin is not used in this mode, the pin can be used as a normal I/O pin or other pin-shared function.

PWM Output Mode

To select this mode, bits STnM1 and STnM0 in the STMnC1 register should be set to 10 respectively. The PWM function within the STMn is useful for applications which require functions such as motor control, heating control, illumination control etc. By providing a signal of fixed frequency but of varying duty cycle on the STMn output pin, a square wave AC waveform can be generated with varying equivalent DC RMS values.

As both the period and duty cycle of the PWM waveform can be controlled, the choice of generated waveform is extremely flexible. In the PWM Output Mode, the STnCCLR bit has no effect as the PWM period. Both of the CCRA and CCRP registers are used to generate the PWM waveform, one register is used to clear the internal counter and thus control the PWM waveform frequency, while the other one is used to control the duty cycle. Which register is used to control either frequency or duty cycle is determined using the STnDPX bit in the STMnC1 register. The PWM waveform frequency and duty cycle can therefore be controlled by the values in the CCRA and CCRP registers.

An interrupt flag, one for each of the CCRA and CCRP, will be generated when a compare match occurs from either Comparator A or Comparator P. The STnOC bit in the STMnC1 register is used to select the required polarity of the PWM waveform while the two STnIO1 and STnIO0 bits are used to enable the PWM output or to force the STMn output pin to a fixed high or low level. The STnPOL bit is used to reverse the polarity of the PWM output waveform.

- **10-bit STM, PWM Output Mode, Edge-aligned Mode, STnDPX=0**

CCRP	1~7	0
Period	CCRP×128	1024
Duty	CCRA	

If f_{SYS}=4MHz, STM clock source is f_{SYS}/4, CCRP=2 and CCRA=128,

The STM PWM output frequency=(f_{SYS}/4)/(2×128)=f_{SYS}/1024=4kHz, duty=128/(2×128)=50%.

If the Duty value defined by the CCRA register is equal to or greater than the Period value, then the PWM output duty is 100%.

- **10-bit STM, PWM Output Mode, Edge-al igned Mode, STnDPX=1**

CCRP	1~7	0
Period	CCRA	
Duty	CCRP×128	1024

The PWM output period is determined by the CCRA register value together with the STMn clock while the PWM duty cycle is defined by the CCRP register value.

- **16-bit STM, PWM Output Mode, Edge-aligned Mode, STnDPX=0**

CCRP	1~255	0
Period	CCRP×256	65536
Duty	CCRA	

If f_{SYS}=4MHz, STM clock source is f_{SYS}/4, CCRP=2 and CCRA=128,

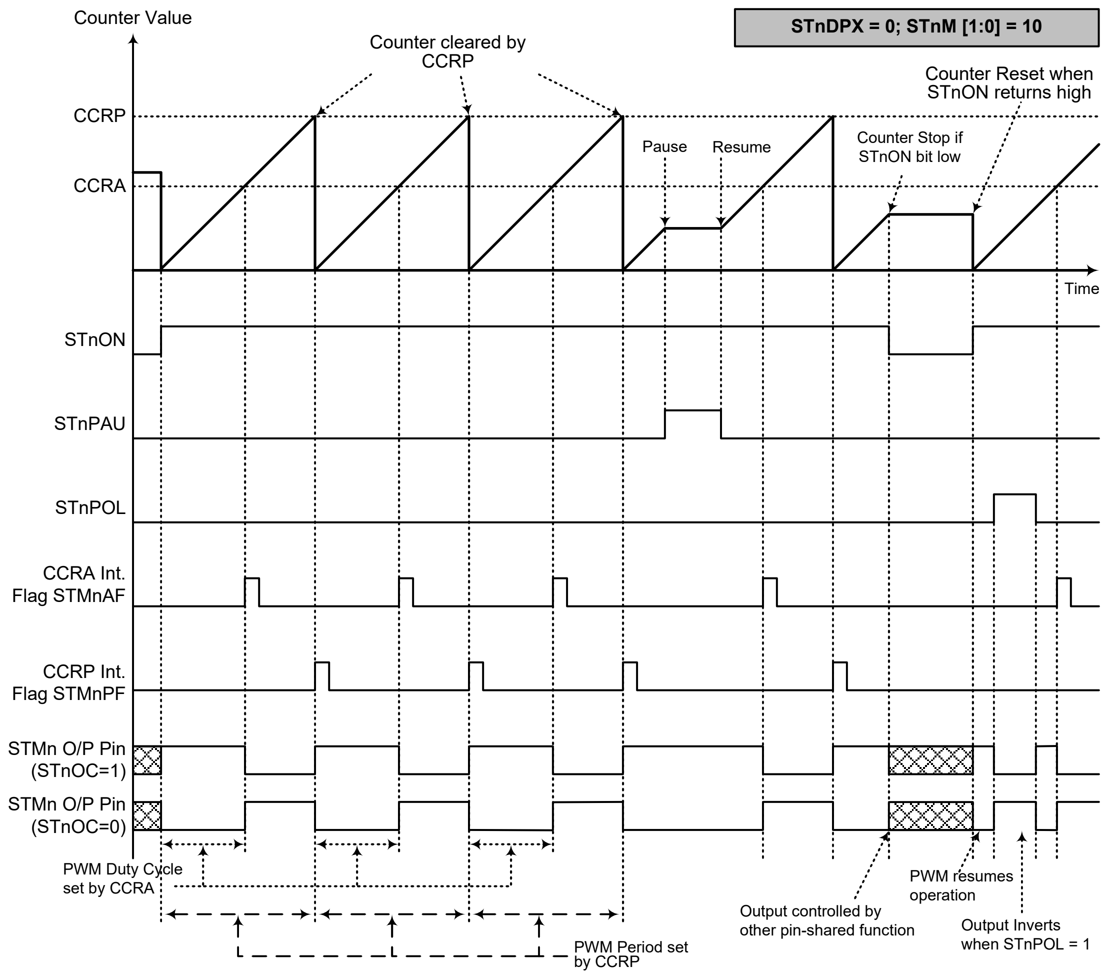
The STM PWM output frequency=(f_{SYS}/4)/(2×256)=f_{SYS}/2048=2kHz, duty=128/(2×256)=25%.

If the Duty value defined by the CCRA register is equal to or greater than the Period value, then the PWM output duty is 100%.

- **16-bit STM, PWM Output Mode, Edge-aligned Mode, STnDPX=1**

CCRP	1~255	0
Period	CCRA	
Duty	CCRP×256	65536

The PWM output period is determined by the CCRA register value together with the STMn clock while the PWM duty cycle is defined by the CCRP register value.

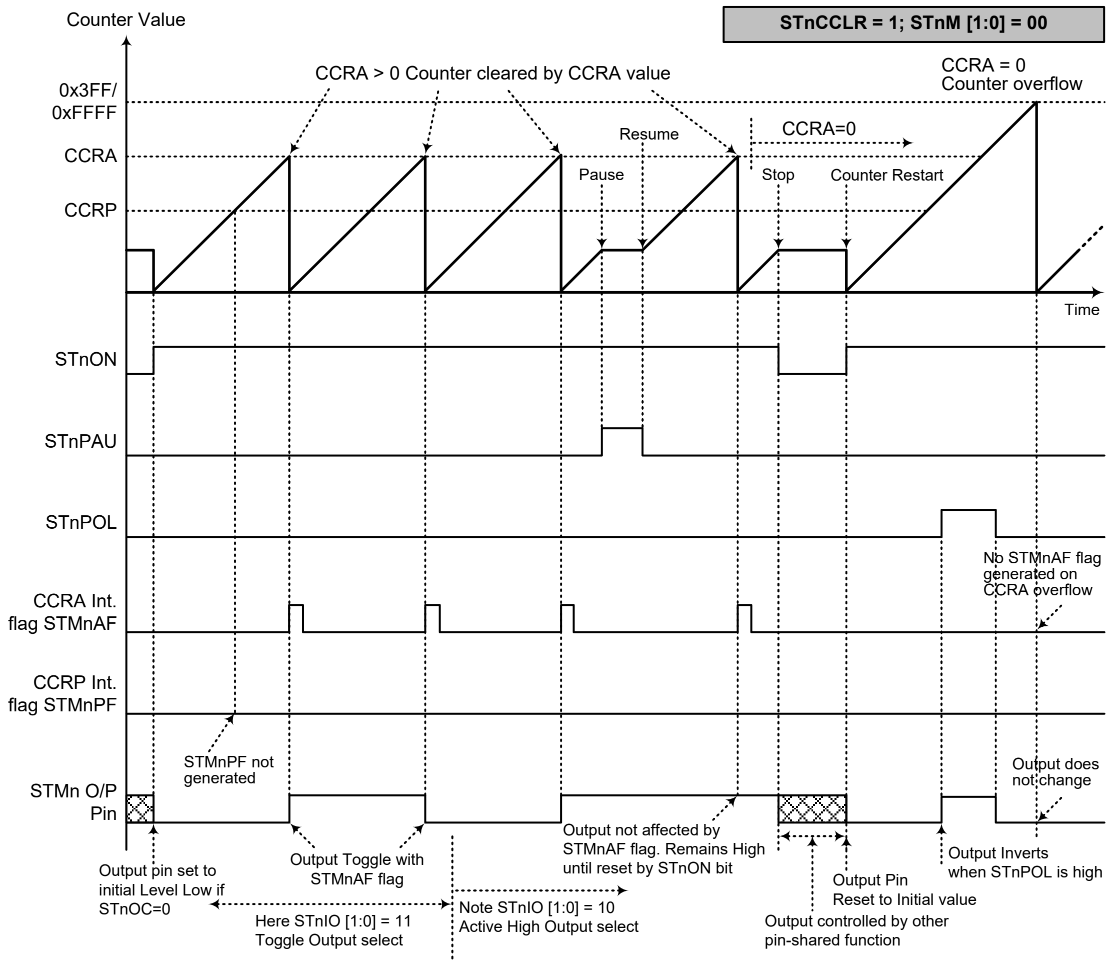


PWM Output Mode – STnDPX=0

Note: 1. Here STnDPX=0 – Counter cleared by CCRP

2. A counter clear sets the PWM Period

3. The internal PWM function continues running even when STnIO [1:0]=00 or 01
4. The STnCCLR bit has no influence on PWM operation
5. n=0 for 10-bit STM while n=1 for 16-bit STM


Counter Value
STnCCLR = 1; STnM [1:0] = 00
0x3FF/ 0xFFFF
CCRA
CCRP
CCRA > 0 Counter cleared by CCRA value
CCRA = 0 Counter overflow
Resume
CCRA=0
Pause
Stop
Counter Restart
Time
STnON
STnPAU
STnPOL
CCRA Int. flag STMnAF
CCRP Int. flag STMnPF
STMn O/P Pin
No STMnAF flag generated on CCRA overflow
STMnPF not generated
Output does not change
Output pin set to initial Level Low if STnOC=0
Output Toggle with STMnAF flag
Output not affected by STMnAF flag. Remains High until reset by STnON bit
Output Pin Reset to Initial value
Output Inverts when STnPOL is high
Here STnIO [1:0] = 11 Toggle Output select
Note STnIO [1:0] = 10 Active High Output select
Output controlled by other pin-shared function

PWM Output Mode – STnDPX=1

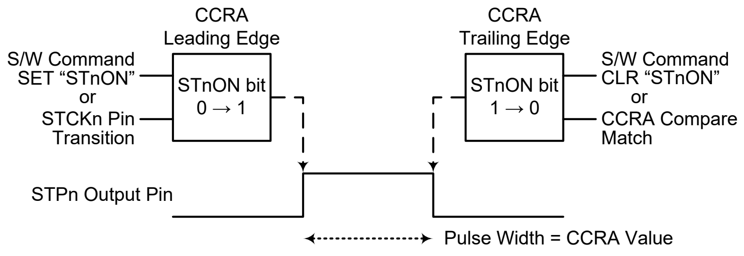
Note: 1. Here STnDPX=1 – Counter cleared by CCRA
2. A counter clear sets the PWM Period
3. The internal PWM function continues running even when STnIO [1:0]=00 or 01
4. The STnCCLR bit has no influence on PWM operation
5. n=0 for 10-bit STM while n=1 for 16-bit STM

Single Pulse Output Mode

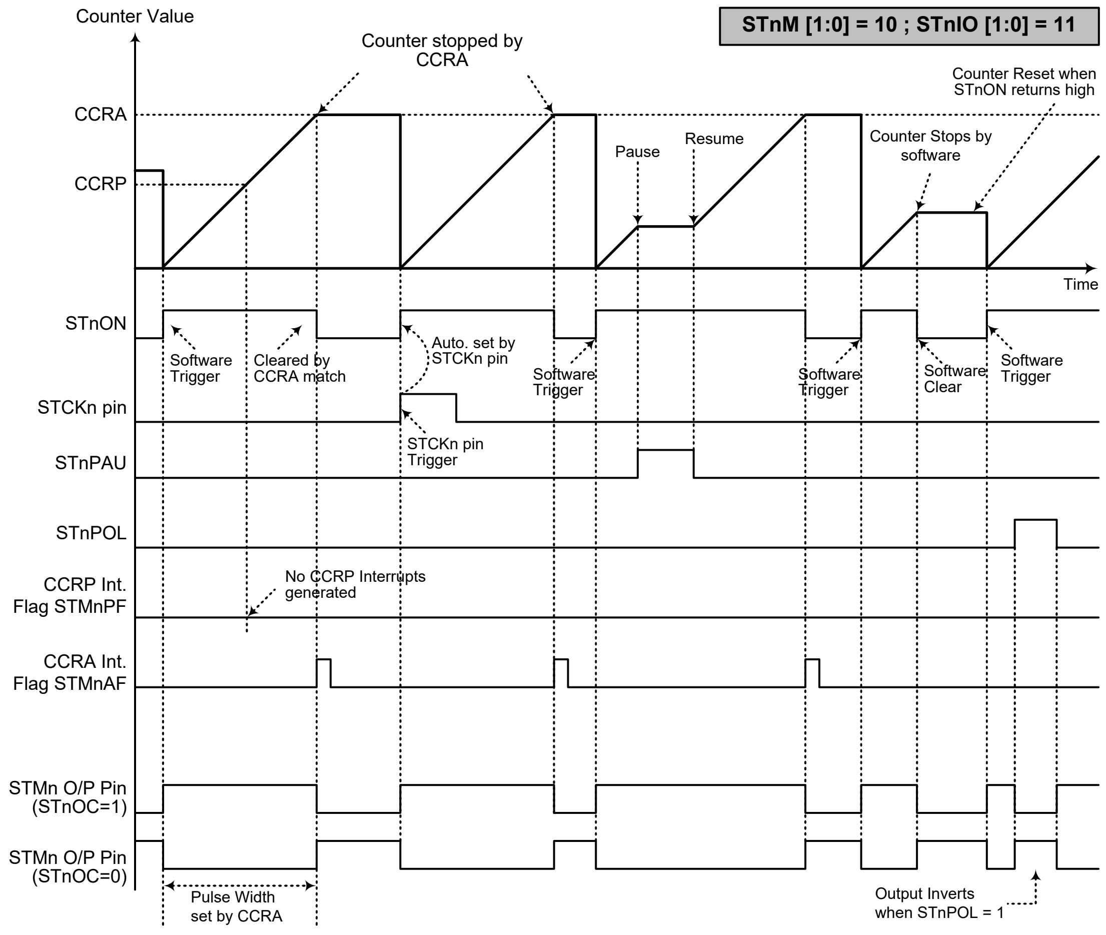
To select this mode, bits STnM1 and STnM0 in the STMnC1 register should be set to 10 respectively and also the STnIO1 and STnIO0 bits should be set to 11 respectively. The Single Pulse Output Mode, as the name suggests, will generate a single shot pulse on the STMn output pin.

The trigger for the pulse output leading edge is a low to high transition of the STnON bit, which can be implemented using the application program. However in the Single Pulse Output Mode, the STnON bit can also be made to automatically change from low to high using the external STCKn pin, which will in turn initiate the Single Pulse output. When the STnON bit transitions to a high level, the counter will start running and the pulse leading edge will be generated. The STnON bit should remain high when the pulse is in its active state. The generated pulse trailing edge will be generated when the STnON bit is cleared to zero, which can be implemented using the application program or when a compare match occurs from Comparator A.

However a compare match from Comparator A will also automatically clear the STnON bit and thus generate the Single Pulse output trailing edge. In this way the CCRA value can be used to control the pulse width. A compare match from Comparator A will also generate an STMn interrupt. The counter can only be reset back to zero when the STnON bit changes from low to high when the counter restarts. In the Single Pulse Output Mode CCRP is not used. The STnCCLR and STnDPX bits are not used in this mode.



Single Pulse Generation



Single Pulse Output Mode

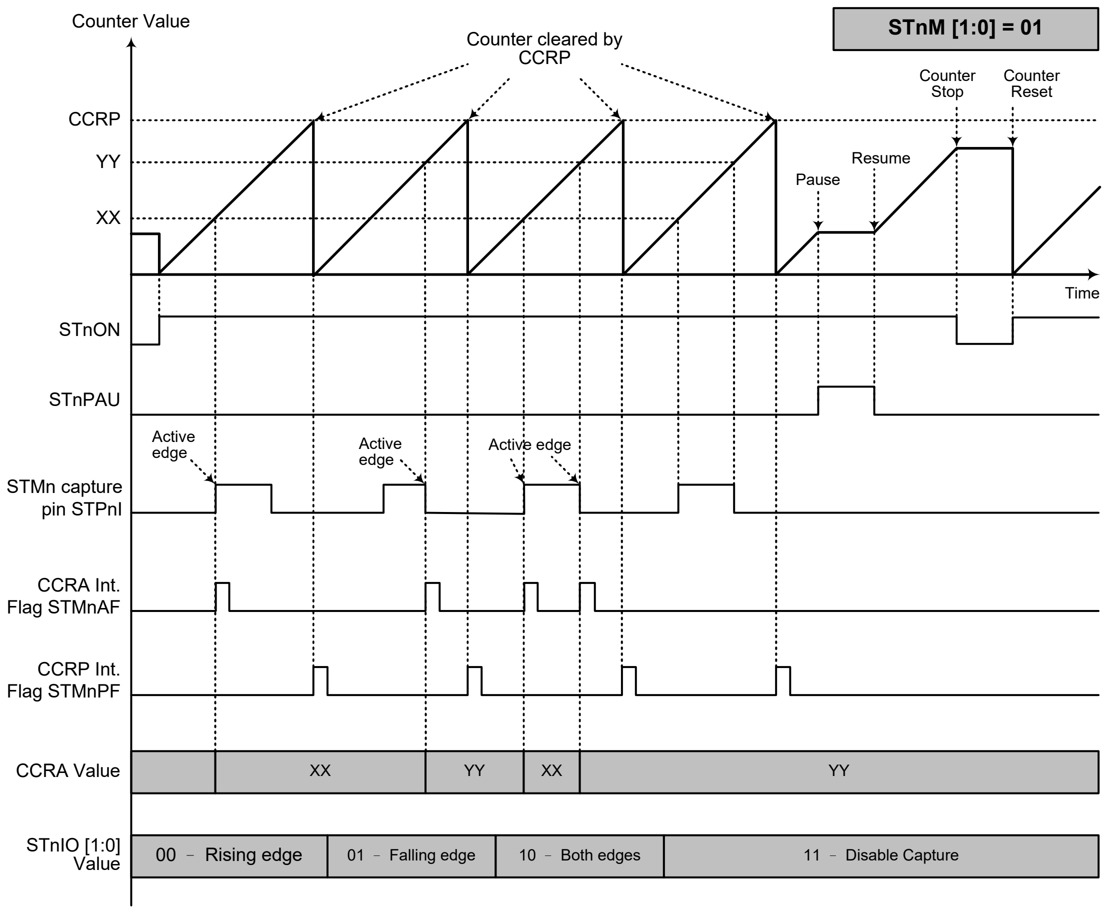
Note: 1. Counter stopped by CCRA
2. CCRP is not used
3. The pulse triggered by the STCKn pin or by setting the STnON bit high
4. An STCKn pin active edge will automatically set the STnON bit high
5. In the Single Pulse Output Mode, STnIO [1:0] must be set to “11” and cannot be changed
6. n=0 for 10-bit STM while n=1 for 16-bit STM

Capture Input Mode

To select this mode bits STnM1 and STnM0 in the STMnC1 register should be set to 01 respectively. This mode enables external signals to capture and store the present value of the internal counter and can therefore be used for applications such as pulse width measurements. The external signal is supplied on the STPnI pin, whose active edge can be a rising edge, a falling edge or both rising and falling edges; the active edge transition type is selected using the STnIO1 and STnIO0 bits in the STMnC1 register. The counter is started when the STnON bit changes from low to high which is initiated using the application program.

When the required edge transition appears on the STPnI pin the present value in the counter will be latched into the CCRA registers and an STMn interrupt generated. Irrespective of what events occur on the STPnI pin the counter will continue to free run until the STnON bit changes from high to low. When a CCRP compare match occurs the counter will reset back to zero; in this way the CCRP value can be used to control the maximum counter value. When a CCRP compare match occurs from Comparator P, an STMn interrupt will also be generated. Counting the number of overflow interrupt signals from the CCRP can be a useful method in measuring long pulse widths. The STnIO1 and STnIO0 bits can select the active trigger edge on the STPnI pin to be a rising edge, falling edge or both edge types. If the STnIO1 and STnIO0 bits are both set high, then no capture operation will take place irrespective of what happens on the STPnI pin, however it must be noted that the counter will continue to run. The STnCCLR and STnDPX bits are not used in this mode.

There are some considerations that should be noted. If the captured pulse width is less than 2 timer clock periods, it may be ignored by hardware. After the counter value is latched to the CCRA registers by an active capture edge, the STMnAF flag will be set high after 0.5 timer clock period. The delay time from the active capture edge received to the action of latching counter value to CCRA registers is less than 1.5 timer clock periods.



Capture Input Mode

Note: 1. STnM [1:0]=01 and active edge set by the STnIO[1:0] bits
2. An STMn Capture input pin active edge transfers the counter value to CCRA
3. STnCCLR bit not used
4. No output function – STnOC and STnPOL bits are not used
5. CCRP determines the counter value and the counter has a maximum count value when CCRP is equal to zero
6. The capture input mode cannot be used if the selected STM counter clock is not available
7. n=0 for 10-bit STM while n=1 for 16-bit STM

Analog to Digital Converter – ADC

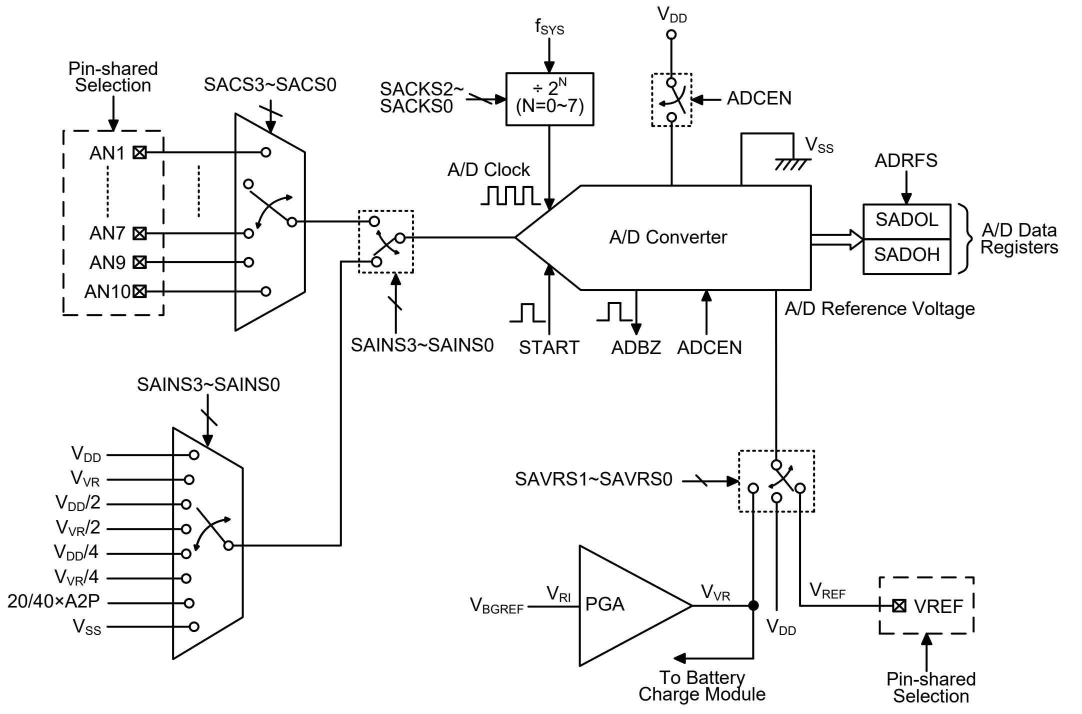
The need to interface to real world analog signals is a common requirement for many electronic systems. However, to properly process these signals by a microcontroller, they must first be converted into digital signals by A/D converters. By integrating the A/D conversion electronic circuitry into the microcontroller, the need for external components is reduced significantly with the corresponding follow-on benefits of lower costs and reduced component space requirements.

A/D Converter Overview

The device contains a multi-channel analog to digital converter which can directly interface to external analog signals, such as that from sensors or other control signals and convert these signals directly into a 12-bit digital value. It also can convert the internal signals, such as the OPA2 output voltage, internal A/D converter power supply and internal PGA output voltage, into a 12-bit digital value. The external or internal analog signal to be converted is determined by the SAINS3~SAINS0 bits and SACS3~SACS0 bits. Note that when the internal analog signal is selected to be converted using the SAINS3~SAINS0 bits, the external channel analog input will automatically be switched off. More detailed information about the A/D input signal selection will be described in the "A/D Converter Input Signals" section.

External Input Channels	Internal Analog Signals	A/D Signal Select
AN1~AN7, AN9~AN10	20/40×A2P, V_{DD}, V_{VR}, $V_{DD}/2$, $V_{VR}/2$, $V_{DD}/4$, $V_{VR}/4$, V_{SS}	SAINS3~SAINS0, SACS3~SACS0

The accompanying block diagram shows the internal structure of the A/D converter, together with its associated registers and control bits.



Note: 20/40×A2P is 20/40 times OPA2 positive input voltage signal. More details can be obtained in the Battery Charge Module section.

A/D Converter Structure

A/D Converter Register Description

Overall operation of the A/D converter is controlled using five registers. A read only register pair exists to store the A/D converter data 12-bit value. Theree registers, SADC0, SADC1 and SADC2, are the control registers which setup the operating and control function of the A/D converter.

Register Name	Bit							
	7	6	5	4	3	2	1	0
SADOL (ADRFS=0)	D3	D2	D1	D0	—	—	—	—
SADOL (ADRFS=1)	D7	D6	D5	D4	D3	D2	D1	D0
SADOH (ADRFS=0)	D11	D10	D9	D8	D7	D6	D5	D4
SADOH (ADRFS=1)	—	—	—	—	D11	D10	D9	D8
SADC0	START	ADBZ	ADCEN	ADRFS	SACS3	SACS2	SACS1	SACS0
SADC1	SAINS3	SAINS2	SAINS1	SAINS0	—	SACKS2	SACKS1	SACKS0
SADC2	—	—	—	—	SAVRS1	SAVRS0	—	—

A/D Converter Register List

A/D Converter Data Registers – SADOL, SADOH

As the device contains an internal 12-bit A/D converter, it requires two data registers to store the converted value. These are a high byte register, known as SADOH, and a low byte register, known as SADOL. After the conversion process takes place, these registers can be directly read by the microcontroller to obtain the digitised conversion value. As only 12 bits of the 16-bit register space is utilised, the format in which the data is stored is controlled by the ADRFS bit in the SADC0 register as shown in the accompanying table. D0~D11 are the A/D conversion result data bits. Any unused bits will be read as zero. Note that the A/D converter data registers contents will keep unchanged if the A/D converter is disabled.

ADRFS	SADOH								SADOL							
	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
0	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0	0	0	0	0
1	0	0	0	0	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0

A/D Converter Data Registers

A/D Converter Control Registers – SADC0, SADC1, SADC2

To control the function and operation of the A/D converter, three control registers known as SADC0, SADC1 and SADC2 are provided. These 8-bit registers define functions such as the selection of which analog signal is connected to the internal A/D converter, the digitised data format, the A/D clock source as well as controlling the start function and monitoring the A/D converter busy status. As the device contains only one actual analog to digital converter hardware circuit, each of the external and internal analog signals must be routed to the converter. The SAINS3~SAINS0 bits in the SADC1 register and SACS3~SACS0 bits in the SADC0 register are used to determine which analog signal derived from the external or internal signals will be connected to the A/D converter. The A/D converter also contains an amplifier, PGA, to generate the A/D converter internal reference voltage. The PGA is always enabled.

The relevant pin-shared function selection bits determine which pins on I/O Ports are used as analog inputs for the A/D converter input and which pins are not. When the pin is selected to be an A/D input, its original function whether it is an I/O or other pin-shared function will be removed. In addition, any internal pull-high resistor connected to the pin will be automatically removed if the pin is selected to be an A/D converter input.

• SADC0 Register

Bit	7	6	5	4	3	2	1	0
Name	START	ADBZ	ADCEN	ADRFS	SACS3	SACS2	SACS1	SACS0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7 **START**: Start the A/D Conversion
0→1→0: Start
This bit is used to initiate an A/D conversion process. The bit is normally low but if set high and then cleared low again, the A/D converter will initiate a conversion process.

Bit 6 **ADBZ**: A/D Converter busy flag
0: No A/D conversion is in progress
1: A/D conversion is in progress
This read only flag is used to indicate whether the A/D conversion is in progress or not. When the START bit is set from low to high and then to low again, the ADBZ flag will be set to 1 to indicate that the A/D conversion is initiated. The ADBZ flag will be cleared to 0 after the A/D conversion is complete.

Bit 5 **ADCEN**: A/D Converter function enable control
0: Disable
1: Enable
This bit controls the A/D internal function. This bit should be set to one to enable the A/D converter. If the bit is set low, then the A/D converter will be switched off reducing the device power consumption. When the A/D converter function is disabled, the contents of the A/D data register pair known as SADOL and SADOH will be unchanged.

Bit 4 **ADRFS**: A/D conversion data format selection
0: A/D converter data format → SADOH=D [11:4]; SADOL=D [3:0]
1: A/D converter data format → SADOH=D [11:8]; SADOL=D [7:0]
This bit controls the format of the 12-bit converted A/D value in the two A/D data registers. Details are provided in the A/D converter data register section.

Bit 3~0 **SACS3~SACS0**: A/D converter external analog input channel selection
0000: Reserved
0001: AN1
0010: AN2
0011: AN3
0100: AN4
0101: AN5
0110: AN6
0111: AN7
1000: Reserved
1001: AN9
1010: AN10
1011~1111: Non-existed channel, the input will be floating

• **SADC1 Register**

Bit	7	6	5	4	3	2	1	0
Name	SAINS3	SAINS2	SAINS1	SAINS0	—	SACKS2	SACKS1	SACKS0
R/W	R/W	R/W	R/W	R/W	—	R/W	R/W	R/W
POR	0	0	0	0	—	0	0	0

Bit 7~4 **SAINS3~SAINS0**: A/D converter input signal selection
0000: External input – External analog channel input, ANn
0001: Internal input – Internal signal derived from V_{DD}
0010: Internal input – Internal signal derived from $V_{DD}/2$
0011: Internal input – Internal signal derived from $V_{DD}/4$
0100: External input – External analog channel input, ANn
0101: Internal input – Internal signal derived from PGA output V_{VR}
0110: Internal input – Internal signal derived from PGA output $V_{VR}/2$
0111: Internal input – Internal signal derived from PGA output $V_{VR}/4$
1000: Internal input – Internal OPA2 output voltage, 20/40×A2P
1001~1011: Internal input – Ground, V_{SS}
1100~1111: External input – External analog channel input, ANn
Care must be taken if the SAINS3~SAINS0 bits are set to "0001~0011", "0101~1011" to select the internal analog signal to be converted. When the internal analog signal is selected to be converted, the external channel signal input will automatically be switched off regardless of the SACS3~SACS0 bits value. It will prevent the external channel input from being connected together with the internal analog signal.

Bit 3 Unimplemented, read as "0"

Bit 2~0 **SACKS2~SACKS0**: A/D conversion clock source select
000: f_{SYS}
001: $f_{SYS}/2$
010: $f_{SYS}/4$
011: $f_{SYS}/8$
100: $f_{SYS}/16$
101: $f_{SYS}/32$
110: $f_{SYS}/64$
111: $f_{SYS}/128$
These three bits are used to select the clock source for the A/D converter.

• **SADC2 Register**

Bit	7	6	5	4	3	2	1	0
Name	—	—	—	—	SAVRS1	SAVRS0	—	—
R/W	—	—	—	—	R/W	R/W	—	—
POR	—	—	—	—	0	0	—	—

Bit 7~4 Unimplemented, read as "0"

Bit 3~2 **SAVRS1~SAVRS0**: A/D converter reference voltage select
00: Internal A/D converter power, V_{DD}
01: External VREF pin
1x: Internal PGA output voltage, V_{VR}
These bits are used to select the A/D converter reference voltage source. When the internal reference voltage source is selected, the reference voltage derived from the external VREF pin will automatically be switched off.

Bit 1~0 Unimplemented, read as "0"

A/D Converter Operation

The START bit in the SADC0 register is used to start the AD conversion. When the microcontroller sets this bit from low to high and then low again, an analog to digital conversion cycle will be initiated.

The ADBZ bit in the SADC0 register is used to indicate whether the analog to digital conversion process is in progress or not. This bit will be automatically set to 1 by the microcontroller after an A/D conversion is successfully initiated. When the A/D conversion is complete, the ADBZ bit will be cleared to 0. In addition, the corresponding A/D interrupt request flag will be set in the interrupt control register, and if the interrupts are enabled, an internal interrupt signal will be generated. This A/D internal interrupt signal will direct the program flow to the associated A/D internal interrupt address for processing. If the A/D internal interrupt is disabled, the microcontroller can poll the ADBZ bit in the SADC0 register to check whether it has been cleared as an alternative method of detecting the end of an A/D conversion cycle.

The clock source for the A/D converter, which originates from the system clock f_{SYS}, can be chosen to be either f_{SYS} or a subdivided version of f_{SYS}. The division ratio value is determined by the SACKS2~SACKS0 bits in the SADC1 register. Although the A/D clock source is determined by the system clock f_{SYS} and by bits SACKS2~SACKS0, there are some limitations on the maximum A/D clock source speed that can be selected. As the recommended range of permissible A/D clock period, t_{ADCK}, is from 0.5μs to 10μs, care must be taken for system clock frequencies. For example, if the system clock operates at a frequency of 8MHz, the SACKS2~SACKS0 bits should not be set to 000, 001 or 111. Doing so will give A/D clock periods that are less than the minimum A/D clock period or greater than the maximum A/D clock period which may result in inaccurate A/D conversion values. Refer to the following table for examples, where values marked with an asterisk * show where, special care must be taken, as the values may be exceed the specified A/D Clock Period range.

f_{SYS}	A/D Clock Period (t_{ADCK})							
	SACKS[2:0] =000 (f_{SYS})	SACKS[2:0] =001 (f_{SYS}/2)	SACKS[2:0] =010 (f_{SYS}/4)	SACKS[2:0] =011 (f_{SYS}/8)	SACKS[2:0] =100 (f_{SYS}/16)	SACKS[2:0] =101 (f_{SYS}/32)	SACKS[2:0] =110 (f_{SYS}/64)	SACKS[2:0] =111 (f_{SYS}/128)
1MHz	1μs	2μs	4μs	8μs	16μs *	32μs *	64μs *	128μs *
2MHz	500ns	1μs	2μs	4μs	8μs	16μs *	32μs *	64μs *
4MHz	250ns *	500ns	1μs	2μs	4μs	8μs	16μs *	32μs *
8MHz	125ns *	250ns *	500ns	1μs	2μs	4μs	8μs	16μs *
12MHz	83ns *	167ns *	333ns *	667ns	1.33μs	2.67μs	5.33μs	10.67μs *
16MHz	62.5ns *	125ns *	250ns *	500ns	1μs	2μs	4μs	8μs

A/D Clock Period Examples

Controlling the power on/off function of the A/D converter circuitry is implemented using the ADCEN bit in the SADC0 register. This bit must be set high to power on the A/D converter. When the ADCEN bit is set high to power on the A/D converter internal circuitry, a certain delay as indicated in the timing diagram must be allowed before an A/D conversion is initiated. Even if no pins are selected for use as A/D inputs, if the ADCEN bit is high, then some power will still be consumed. In power conscious applications it is therefore recommended that the ADCEN is set low to reduce power consumption when the A/D converter function is not being used.

A/D Converter Reference Voltage

The actual reference voltage supply to the A/D Converter can be supplied from the positive power supply, V_{DD}, an external reference source supplied on pin VREF or an internal reference voltage V_{VR} determined by the SAVRS1~SAVRS0 bits in the SADC2 register. The internal reference voltage V_{VR} is an amplified output signal through an amplifier, PGA, which is always enabled. The PGA input comes from the internal Bandgap reference voltage, V_{BGREF}. As the VREF pin is pin-shared with other functions, when the VREF pin is selected as the reference voltage pin, the VREF pin-shared function selection bits should first be properly configured to disable other pin-shared functions. However, if the internal reference signal is selected as the reference source, the external reference input from the VREF pin will automatically be switched off by hardware.

Note that the internal Bandgap reference circuit is always enabled. A specific start-up time is necessary for the Bandgap circuit to become stable and accurate.

The analog input values must be allowed to exceed the value of the selected A/D reference voltage.

SAVRS1~SAVRS0	Reference	Description
00	V_{DD}	Internal A/D converter power supply voltage
01	VREF pin	External A/D converter reference pin VREF
10, 11	V_{VR}	Internal A/D converter PGA output voltage

A/D Converter Reference Voltage Selection

A/D Converter Input Signals

All of the external A/D analog input pins are pin-shared with the I/O pins as well as other functions. The corresponding pin-shared function selection bits in the PxS1 and PxS0 registers, determine whether the external input pins are setup as A/D converter analog channel inputs or whether they have other functions. If the corresponding pin is setup to be an A/D converter analog channel input, the original pin function will be disabled. In this way, pins can be changed under program control to change their function between A/D inputs and other functions. All pull-high resistors, which are setup through register programming, will be automatically disconnected if the pins are setup as A/D inputs. Note that it is not necessary to first setup the A/D pin as an input in the port control register to enable the A/D input as when the relevant A/D input function selection bits enable an A/D input, the status of the port control register will be overridden.

When the SAINS3~SAINS0 bits are set to the value of "0x01", "0x10", "0x11" or "10xx", the internal analog signal will be selected. If the internal analog signal is selected to be converted, the external channel signal input will automatically be switched off regardless of the SACS3~SACS0 bits value. It will prevent the external channel input from being connected together with the internal analog signal.

SAINS [3:0]	SACS [3:0]	Input Signals	Description
0000, 0100, 1100~1111	0001~0111, 1001~1010	AN1~AN7, AN9~AN10	External channel analog input ANn
	1011~1111	—	Floating, no external channel is selected
0001	xxxx	V_{DD}	Internal A/D converter power supply voltage V_{DD}
0010	xxxx	$V_{DD}/2$	Internal A/D converter power supply voltage $V_{DD}/2$
0011	xxxx	$V_{DD}/4$	Internal A/D converter power supply voltage $V_{DD}/4$
0101	xxxx	V_{VR}	Internal A/D converter PGA output V_{VR}
0110	xxxx	$V_{VR}/2$	Internal A/D converter PGA output $V_{VR}/2$
0111	xxxx	$V_{VR}/4$	Internal A/D converter PGA output $V_{VR}/4$
1000	xxxx	20/40×A2P	20/40 times OPA2 positive input voltage signal
1001~1011	xxxx	V_{SS}	Connected to ground

"x": Don't care

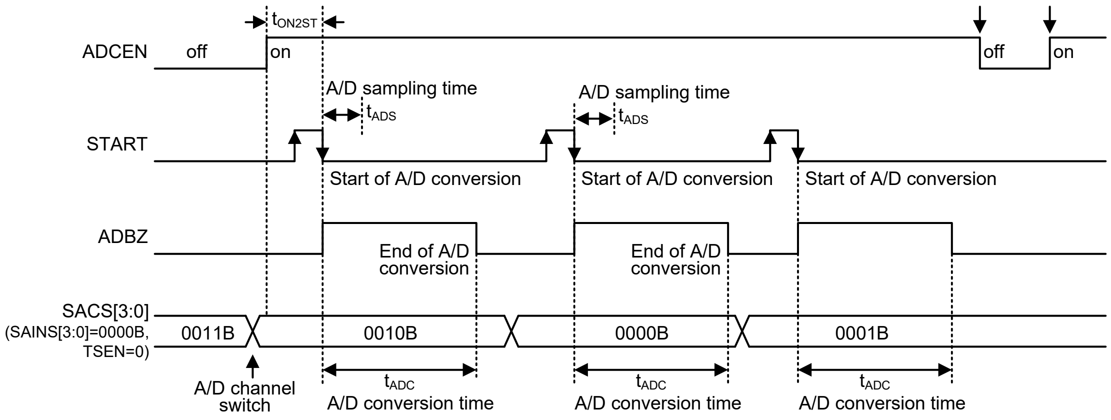
A/D Converter Input Signal Selection

Conversion Rate and Timing Diagram

A complete A/D conversion contains two parts, data sampling and data conversion. The data sampling which is defined as t_{ADS} takes 4 A/D clock periods and the data conversion takes 12 A/D clock periods. Therefore a total of 16 A/D clock periods for an analog signal A/D conversion which is defined as t_{ADC} are necessary.

Maximum single A/D conversion rate=1/(A/D clock period×16)

The accompanying diagram shows graphically the various stages involved in an analog to digital conversion process and its associated timing. After an A/D conversion process has been initiated by the application program, the microcontroller internal hardware will begin to carry out the conversion, during which time the program can continue with other functions. The time taken for the A/D conversion is 16 t_{ADCK} where t_{ADCK} is equal to the A/D clock period.



A/D Conversion Timing – External Channel Input

Summary of A/D Conversion Steps

The following summarises the individual steps that should be executed in order to implement an A/D conversion process.

- Step 1

  Select the required A/D conversion clock by properly programming the SACKS2~SACKS0 bits in the SADC1 register.

- Step 2

  Enable the A/D converter by setting the ADCEN bit in the SADC0 register to one.

- Step 3

  Select which signal is to be connected to the internal A/D converter by correctly configuring the SACS3~SACS0 bits and SAINS3~SAINS0 bits.

  Selecting the external channel input to be converted, go to Step 4.

  Selecting the internal analog signal to be converted, go to Step 5.

- Step 4

  If the SAINS3~SAINS0 bits are 0000, 0100 or 11xx, the external channel input can be selected. The desired external channel input is selected by configuring the SACS3~SACS0. When the A/D input signal comes from the external channel input, the corresponding pin should be configured as an A/D input function by selecting the relevant pin-shared function control bits. Then go to Step 6.

- Step 5

  If the SAINS3~SAINS0 bits are set to 0x01, 0x10, 0x11 or 1000, the relevant internal analog signal will be selected. When the internal analog signal is selected to be converted, the external channel analog input will automatically be disconnected. Then go to Step 6.

- Step 6

  Select the A/D converter reference voltage source by configuring the SAVRS1~SAVRS0 bits.

- Step 7

  Select the A/D converter output data format by configuring the ADRFS bit.

- Step 8

  If A/D conversion interrupt is used, the interrupt control registers must be correctly configured to ensure the A/D interrupt function is active. The master interrupt control bit, EMI, and the A/D conversion interrupt control bit, ADE, must both be set high in advance.

- Step 9

  The A/D conversion procedure can now be initialized by setting the START bit from low to high and then low again.

- Step 10

  If A/D conversion is in progress, the ADBZ flag will be set high. After the A/D conversion process is complete, the ADBZ flag will go low and then the output data can be read from SADOH and SADOL registers.

Note: When checking for the end of the conversion process, if the method of polling the ADBZ bit in the SADC0 register is used, the interrupt enable step above can be omitted.

Programming Considerations

During microcontroller operations where the A/D converter is not being used, the A/D internal circuitry can be switched off to reduce power consumption by clearing bit ADCEN to 0 in the SADC0 register. When this happens, the internal A/D converter circuits will not consume power irrespective of what analog voltage is applied to their input lines. If the A/D converter input lines are used as normal I/Os, then care must be taken as if the input voltage is not at a valid logic level, then this may lead to some increase in power consumption.

A/D Converter Transfer Function

As the device contains a 12-bit A/D converter, its full-scale converted digitised value is equal to FFFH. Since the full-scale analog input value is equal to the actual A/D converter reference voltage, V_{REF}, this gives a single bit analog input value of reference voltage value divided by 4096.

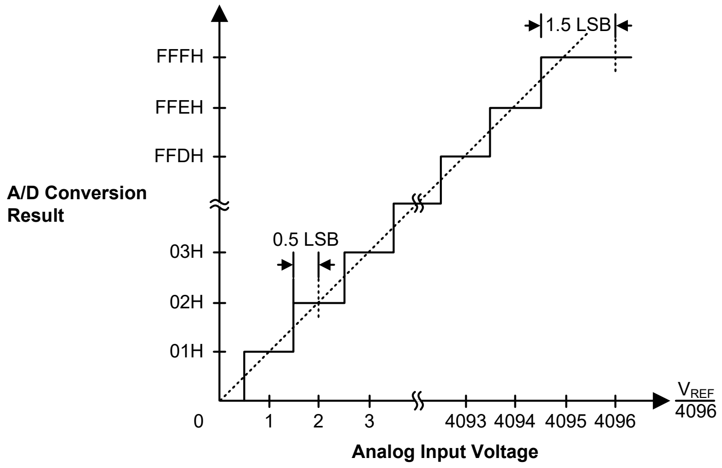
$$1\ \text{LSB}=V_{REF}/4096$$

The A/D Converter input voltage value can be calculated using the following equation:

$$\text{A/D input voltage}=\text{A/D output digital value}\times V_{REF}/4096$$

The diagram shows the ideal transfer function between the analog input value and the digitised output value for the A/D converter. Except for the digitised zero value, the subsequent digitised values will change at a point 0.5 LSB below where they would change without the offset, and the last full scale digitised value will change at a point 1.5 LSB below the V_{REF} level.

Note that here the V_{REF} voltage is the actual A/D converter reference voltage determined by the SAVRS1~SAVRS0 bits.



Ideal A/D Transfer Function

A/D Programming Examples

The following two programming examples illustrate how to setup and implement an A/D conversion. In the first example, the method of polling the ADBZ bit in the SADC0 register is used to detect when the conversion cycle is complete, whereas in the second example, the A/D interrupt is used to determine when the conversion is complete.

Example: using an ADBZ polling method to detect the end of conversion

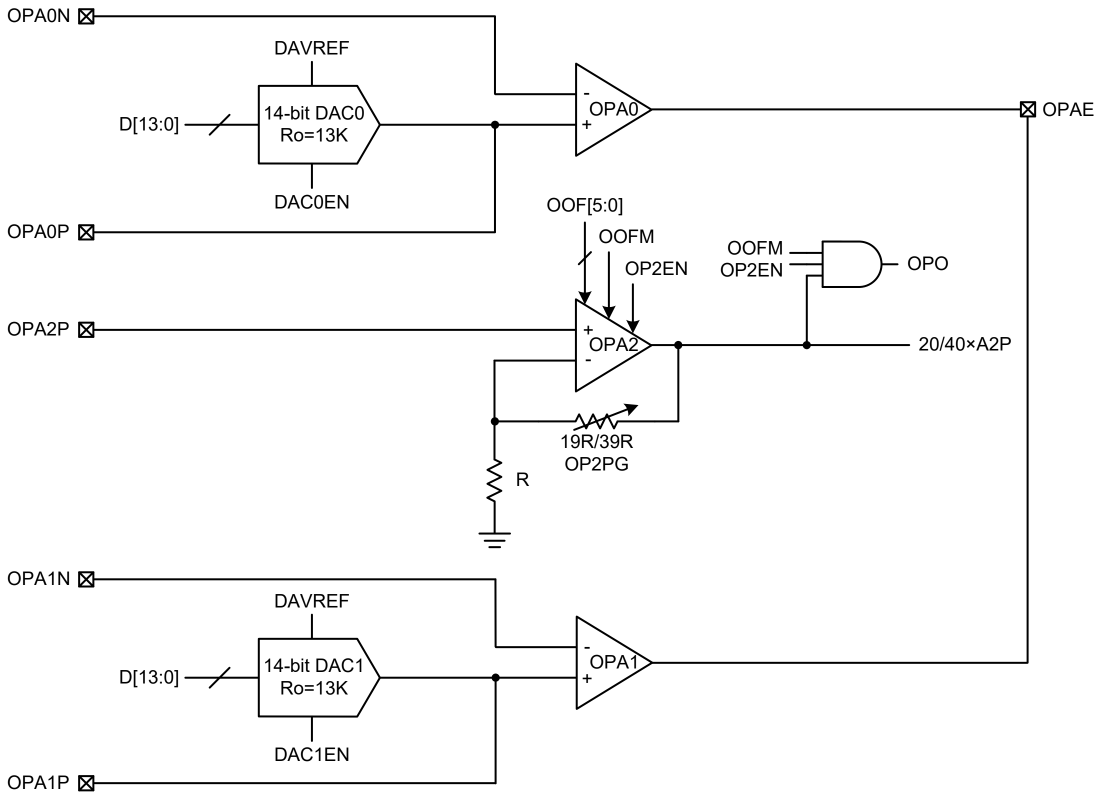
```
clr ADE              ; disable ADC interrupt
mov a,03H
mov SADC1,a          ; select A/D input signal from external channel, fSYS/8 as A/D
                     ; clock
mov a,00H            ; select VDD as the A/D reference voltage source
mov SADC2,a
mov a,0CH            ; setup PBS0 to configure pin AN1
mov PBS0,a
mov a,21H            ; enable A/D converter and select AN1 as the A/D external
                     ; channel input
mov SADC0,a
:
start_conversion:
clr START            ; high pulse on start bit to initiate conversion
set START            ; reset A/D
clr START            ; start A/D
:
polling_EOC:
sz  ADBZ             ; poll the SADC0 register ADBZ bit to detect end of A/D
                     ; conversion
jmp polling_EOC      ; continue polling
:
mov a,SADOL          ; read low byte conversion result value
mov SADOL_buffer,a   ; save result to user defined register
mov a,SADOH          ; read high byte conversion result value
mov SADOH_buffer,a   ; save result to user defined register
:
jmp start_conversion ; start next A/D conversion
```

Example: using the interrupt method to detect the end of conversion

```
clr ADE              ; disable ADC interrupt
mov a,03H
mov SADC1,a          ; select A/D input signal from external channel, fSYS/8 as A/D
                     ; clock
mov a,00H            ; select VDD as the A/D reference voltage source
mov SADC2,a
mov a,0CH            ; setup PBS0 to configure pin AN1
mov PBS0,a
mov a,21H
mov SADC0,a          ; enable A/D converter and select AN1 as the A/D external
                     ; channel input
:
Start_conversion:
clr START            ; high pulse on START bit to initiate conversion
set START            ; reset A/D
clr START            ; start A/D
clr ADF              ; clear ADC interrupt request flag
set ADE              ; enable ADC interrupt
set EMI              ; enable global interrupt
:
:
ADC_ISR:             ; ADC interrupt service routine
mov acc_stack,a      ; save ACC to user defined memory
mov a,STATUS
mov status_stack,a   ; save STATUS to user defined memory
:
mov a,SADOL          ; read low byte conversion result value
mov SADOL_buffer,a   ; save result to user defined register
mov a,SADOH          ; read high byte conversion result value
mov SADOH_buffer,a   ; save result to user defined register
:
EXIT_INT_ISR:
mov a,status_stack
mov STATUS,a         ; restore STATUS from user defined memory
mov a,acc_stack      ; restore ACC from user defined memory
reti
```

Battery Charge Module

The device contains a battery charge module which consists of three operational amplifies and two 14-bit D/A converters. The OPA0 together with DAC0 are used for battery charge constant current (CC) control and the OPA1 together with DAC1 are used for battery charge constant voltage (CV) control. The OPA2 is used for battery charge current amplification.



Battery Charge Module Structure

Note: 1. The OPA0 and OPA1 are always enabled, while the OPA2 is controlled by the OP2EN bit in the DAOPC register.
2. The OPA0 and OPA1 are open drain outputs.
3. The OPA0 and OPA1 do not need to calibrate the input offset.
4. The OPA2 needs to calibrate the input offset.
5. When the DAC0 or DAC1 is disabled, the output will be in a floating state.
6. The DAVREF voltage comes from A/D PGA output voltage, V_{VR}.

Battery Charge Module Registers

Overall operation of the battery charge module is controlled using a series of registers and the corresponding register definitions are described in the accompanying sections.

Register Name	Bit							
	7	6	5	4	3	2	1	0
DA0L	D7	D6	D5	D4	D3	D2	D1	D0
DA0H	—	—	D13	D12	D11	D10	D9	D8
DA1L	D7	D6	D5	D4	D3	D2	D1	D0
DA1H	—	—	D13	D12	D11	D10	D9	D8
DAOPC	DAC1EN	DAC0EN	OP2EN	OP2PG	—	—	—	OPO
OPVOS	OOFM	—	OOF5	OOF4	OOF3	OOF2	OOF1	OOF0

Battery Charge Module Register List

Digital to Analog Converter

The battery charge module contains two 14-bit R2R D/A converters, namely DAC0 and DAC1. Their reference input voltage comes from DAVREF, and can be power down to save power.

The DAC0 and DAC1 are enabled or disabled by the DAOPC register. They are used to set a reference charging current and voltage using the DA0H/DA0L and DA1H/DA1L registers respectively.

• DA0L Register

Bit	7	6	5	4	3	2	1	0
Name	D7	D6	D5	D4	D3	D2	D1	D0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	1	0	0	0	0	0	1	0

Bit 7~0 **D7~D0**: D/A converter 0 output control code low byte
Writing this register will only write the data to a shadow buffer and writing the DA0H register will simultaneously copy the shadow buffer data to the DA0L register.

• DA0H Register

Bit	7	6	5	4	3	2	1	0
Name	—	—	D13	D12	D11	D10	D9	D8
R/W	—	—	R/W	R/W	R/W	R/W	R/W	R/W
POR	—	—	0	0	0	0	0	0

Bit 7~6 Unimplemented, read as “0”
Bit 5~0 **D13~D8**: D/A converter 0 output control code high byte
The D/A converter 0 output voltage is calculated using the following equation:
DAC0OUT=(DAVREF/2^{14})×D[13:0], where DAVREF is D/A converter 0 reference input voltage.

• DA1L Register

Bit	7	6	5	4	3	2	1	0
Name	D7	D6	D5	D4	D3	D2	D1	D0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7~0 **D7~D0**: D/A converter 1 output control code low byte
Writing this register will only write the data to a shadow buffer and writing the DA1H register will simultaneously copy the shadow buffer data to the DA1L register.

- **DA1H Register**

Bit	7	6	5	4	3	2	1	0
Name	—	—	D13	D12	D11	D10	D9	D8
R/W	—	—	R/W	R/W	R/W	R/W	R/W	R/W
POR	—	—	1	0	0	0	0	0

Bit 7~6 Unimplemented, read as "0"

Bit 5~0 **D13~D8**: D/A converter 1 output control code high byte
The D/A converter 1 output voltage is calculated using the following equation:
DAC1OUT=(DAVREF/2^{14})×D[13:0], where DAVREF is D/A converter 1 reference input voltage.

- **DAOPC Register**

Bit	7	6	5	4	3	2	1	0
Name	DAC1EN	DAC0EN	OP2EN	OP2PG	—	—	—	OPO
R/W	R/W	R/W	R/W	R/W	—	—	—	R
POR	1	1	0	0	—	—	—	0

Bit 7 **DAC1EN**: D/A converter 1 enable control
0: Disable, D/A converter 1 output floating
1: Enable

Bit 6 **DAC0EN**: D/A converter 0 enable control
0: Disable, D/A converter 0 output floating
1: Enable

Bit 5 **OP2EN**: OPA2 enable control
0: Disable
1: Enable

Bit 4 **OP2PG**: OPA2 PGA gain selection
0: 20
1: 40

Bit 3~1 Unimplemented, read as "0"

Bit 0 **OPO**: OPA2 digital logic output
The OPO is cleared to 0 when the OPA2 is disabled.

Operational Amplifiers

The battery charge module contains three operational amplifiers, namely OPA0, OPA1 and OPA2. The OPA0 and OPA1 are always enabled and do not need to calibrate the input offset. The OPA2 related functions are controlled using the DAOPC and OPVOS registers.

The DAOPC register is used for control OPA2 enable/disable, OPA2 PGA gain selection and output status monitoring. The OPVOS register is used for OPA2 input offset calibration voltage selection and control.

- **OPVOS Register**

Bit	7	6	5	4	3	2	1	0
Name	OOFM	—	OOF5	OOF4	OOF3	OOF2	OOF1	OOF0
R/W	R/W	—	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	—	1	0	0	0	0	0

Bit 7 **OOFM**: OPA2 normal operation or input offset voltage cancellation mode selection bit
0: Normal operation
1: Offset calibration mode
The input reference voltage comes from OPA2 positive input pin at offset voltage cancellation mode.

Bit 6 Unimplemented, read as "0"

Bit 5~0 **OOF5~OOF0**: OPA2 input offset voltage calibration control bits

Operational Amplifier 2 Operation

The OPA2 provides input offset calibration function. The calibrated data is stored in the OOF5~OOF0 bits. The OOFM bit is used to control cancellation mode selection. The input reference voltage comes from the OPA2P pin in calibration mode. The OPA2P pin is the OPA2 positive input and the 20/40×A2P signal is the OPA2 analog output voltage. The OPA2 digital output flag is OPO, which is used for OPA2 calibration mode. Finally, the OP2EN bit is used to enable or disable the OPA2 function.

Offset Calibration Procedure

As the OPA2 input pin is pin-shared with other functions, it should be configured as the operational amplifier input first by the corresponding pin-shared function selection register.

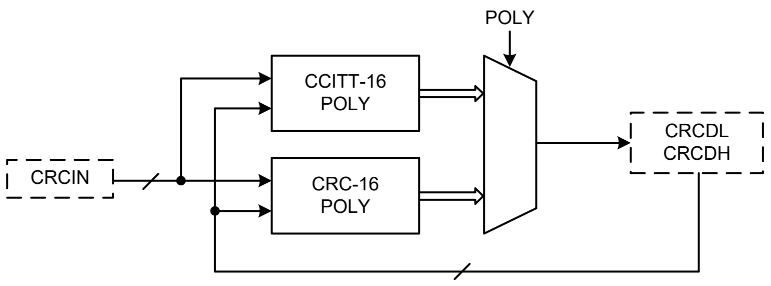
- Step1: Set OOFM=1, the OPA2 is now under offset calibration mode. To make sure the input offset voltage V_{OS} as minimise as possible after calibration, the input reference voltage in calibration mode should be the same as input DC operating voltage in normal mode operation.
- Step2: Set OOF[5:0]=000000 then read OPO flag.
- Step3: Let OOF[5:0]=OOF[5:0]+1 then read OPO flag, if the OPO flag state is changed, record the data as V_{OS1}.
- Step4: Set OOF[5:0]=111111 then read OPO flag.
- Step5: Let OOF[5:0]=OOF[5:0]-1 then read OPO flag, if the OPO flag state is changed; record the data as V_{OS2}.
- Step6: restore $V_{OS}=(V_{OS1}+V_{OS2})/2$ to OOF[5:0] bits, the calibration is finished.

  If $(V_{OS1}+V_{OS2})/2$ is not integral, discard the decimal.

  Residue $V_{OS}=V_{OUT}-V_{IN}$.

Cyclic Redundancy Check – CRC

The Cyclic Redundancy Check, CRC, calculation unit is an error detection technique test algorithm and uses to verify data transmission or storage data correctness. A CRC calculation takes a data stream or a block of data as input and generates a 16-bit output remainder. Ordinarily, a data stream is suffixed by a CRC code and used as a checksum when being sent or stored. Therefore, the received or restored data stream is calculated by the same generator polynomial as described in the following section.



CRC Block Diagram

CRC Registers

The CRC generator contains an 8-bit CRC data input register, CRCIN, and a CRC checksum register pair, CRCDH and CRCDL. The CRCIN register is used to input new data and the CRCDH and CRCDL registers are used to hold the previous CRC calculation result. A CRC control register, CRCCR, is used to select which CRC generating polynomial is used.

Register Name	Bit							
	7	6	5	4	3	2	1	0
CRCIN	D7	D6	D5	D4	D3	D2	D1	D0
CRCDL	D7	D6	D5	D4	D3	D2	D1	D0
CRCDH	D15	D14	D13	D12	D11	D10	D9	D8
CRCCR	—	—	—	—	—	—	—	POLY

CRC Register List

• CRCIN Register

Bit	7	6	5	4	3	2	1	0
Name	D7	D6	D5	D4	D3	D2	D1	D0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7~0 **D7~D0**: CRC input data register

• CRCDL Register

Bit	7	6	5	4	3	2	1	0
Name	D7	D6	D5	D4	D3	D2	D1	D0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7~0 **D7~D0**: 16-bit CRC checksum low byte data register

- **CRCDH Register**

Bit	7	6	5	4	3	2	1	0
Name	D15	D14	D13	D12	D11	D10	D9	D8
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7~0 **D15~D8**: 16-bit CRC checksum high byte data register

- **CRCCR Register**

Bit	7	6	5	4	3	2	1	0
Name	—	—	—	—	—	—	—	POLY
R/W	—	—	—	—	—	—	—	R/W
POR	—	—	—	—	—	—	—	0

Bit 7~1 Unimplemented, read as “0”

Bit 0 **POLY**: 16-bit CRC generating polynomial selection
0: CRC-CCITT: $X^{16}+X^{12}+X^5+1$
1: CRC-16: $X^{16}+X^{15}+X^2+1$

CRC Operation

The CRC generator provides the 16-bit CRC result calculation based on the CRC16 and CCITT CRC16 polynomials. In this CRC generator, there are only these two polynomials available for the numeric values calculation. It can not support the 16-bit CRC calculations based on any other polynomials.

The following two expressions can be used for the CRC generating polynomial which is determined using the POLY bit in the CRC control register, CRCCR. The CRC calculation result is called as the CRC checksum, CRCSUM, and stored in the CRC checksum register pair, CRCDH and CRCDL.

- CRC-CCITT: $X^{16}+X^{12}+X^5+1$
- CRC-16: $X^{16}+X^{15}+X^2+1$

CRC Computation

Each write operation to the CRCIN register creates a combination of the previous CRC value stored in the CRCDH and CRCDL registers and the new data input. The CRC unit calculates the CRC data register value is based on byte by byte. It will take one MCU instruction cycle to calculate the CRC checksum.

CRC Calculation Procedures

1. Clear the checksum register pair, CRCDH and CRCDL.
2. Execute an “Exclusive OR” operation with the 8-bit input data byte and the 16-bit CRCSUM high byte. The result is called the temporary CRCSUM.
3. Shift the temporary CRCSUM value left by one bit and move a “0” into the LSB.
4. Check the shifted temporary CRCSUM value after procedure 3.

   If the MSB is 0, then this shifted temporary CRCSUM will be considered as a new temporary CRCSUM.

   Otherwise, execute an “Exclusive OR” operation with the shifted temporary CRCSUM in procedure 3 and a data “8005H”. Then the operation result will be regarded as the new temporary CRCSUM.

   Note that the data to be perform an “Exclusive OR” operation is “8005H” for the CRC-16 polynomial while for the CRC-CCITT polynomial the data is “1021H”.

5. Repeat the procedure 3 ~ procedure 4 until all bits of the input data byte are completely calculated.
6. Repeat the procedure 2 ~ procedure 5 until all of the input data bytes are completely calculated. Then, the latest calculated result is the final CRC checksum, CRCSUM.

CRC Calculation Examples

- Write 1 byte input data into the CRCIN register and the corresponding CRC checksum are individually calculated as the following table shown.

CRC Data Input / CRC Polynomial	00H	01H	02H	03H	04H	05H	06H	07H
CRC-CCITT ($X^{16}+X^{12}+X^5+1$)	0000H	1021H	2042H	3063H	4084H	50A5H	60C6H	70E7H
CRC-16 ($X^{16}+X^{15}+X^2+1$)	0000H	8005H	800FH	000AH	801BH	001EH	0014H	8011H

Note: The initial value of the CRC checksum register pair, CRCDH and CRCDL, is zero before each CRC input data is written into the CRCIN register.

- Write 4 bytes input data into the CRCIN register sequentially and the CRC checksum are sequentially listed in the following table.

CRC Data Input / CRC Polynomial	CRCIN=78H→56H→34H→12H
CRC-CCITT ($X^{16}+X^{12}+X^5+1$)	(CRCDH, CRCDL)=FF9FH→BBC3H→A367H→D0FAH
CRC-16 ($X^{16}+X^{15}+X^2+1$)	(CRCDH, CRCDL)=0110H→91F1H→F2DEhH→5C43H

Note: The initial value of the CRC checksum register pair, CRCDH and CRCDL, is zero before the sequential CRC data input operation.

Program Memory CRC Checksum Calculation Example

1. Clear the checksum register pair, CRCDH and CRCDL.
2. Select the CRC-CCITT or CRC-16 polynomial as the generating polynomial using the POLY bit in the CRCCR register.
3. Execute the table read instruction to read the program memory data value.
4. Write the table data low byte into the CRCIN register and execute the CRC calculation with the current CRCSUM value. Then a new CRCSUM result will be obtained and stored in the CRC checksum register pair, CRCDH and CRCDL.
5. Write the table data high byte into the CRCIN register and execute the CRC calculation with the current CRCSUM value. Then a new CRCSUM result will be obtained and stored in the CRC checksum register pair, CRCDH and CRCDL.
6. Repeat the procedure 3 ~ procedure 5 to read the next program memory data value and execute the CRC calculation until all program memory data are read followed by the sequential CRC calculation. Then the value in the CRC checksum register pair is the final CRC calculation result.

CAN Bus Controller

The device includes a fully integrated CAN (Controller Area Network) bus controller. This chapter will introduce the CAN bus controller in terms of Power Control Function, Functional Description and Register Description.

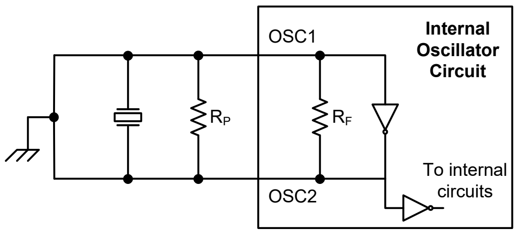
Power Control Function

The CAN bus controller operating clock is from an external crystal oscillator, HXT. The oscillator can be enabled or disabled using a register bit HXTEN. The clock which is a divided version of the CAN bus controller system clock can be output on the CLKOUT pin.

External Crystal Oscillator – HXT

There is a high frequency external crystal oscillator for the CAN bus controller. For most crystal oscillator configurations, the simple connection of a crystal across OSC1 and OSC2 will create the necessary phase shift and feedback for oscillation, without requiring external capacitors. It is recommended to connect an 8MHz, 16MHz or 24MHz crystal to the HXT pins for applications.

For oscillator stability and to minimise the effects of noise and crosstalk, it is important to ensure that the crystal and any associated resistors and capacitors along with interconnecting lines are all located as close to the device as possible.

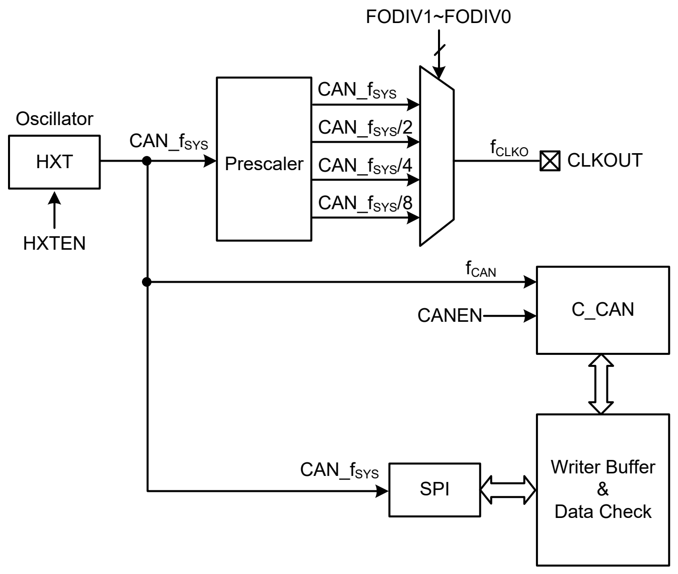


Note: 1. R_P is normally not required.
2. Although not shown OSC1/OSC2 pins have a parasitic capacitance of around 7pF.

CLKOUT Pin

The clock output pin, CLKOUT is provided to the users for use as a clock input for other devices. The CLKOUT pin has an internal prescaler which can devide CAN_f_{SYS} by 1, 2, 4 and 8. The prescaler is selected via the FOCFG register. When clearing the HXTEN bit in the FOCFG register (ADDRESS: 0xC0) to zero, the HXT oscillator is off thus turning off the CLKOUT clock output.

The CLKOUT pin can be enabled or disabled using the CLKOEN bit in the SFIOSTC register.



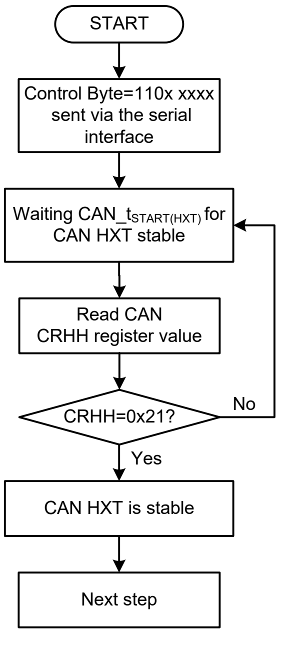
IDLE Mode

The IDLE Mode is entered when the HXTEN bit in the FOCFG register is high while the CANEN bit in the CANCFG register is low. In the IDLE Mode the HXT oscillator is continue to provide a clock. The C_CAN is disabled.

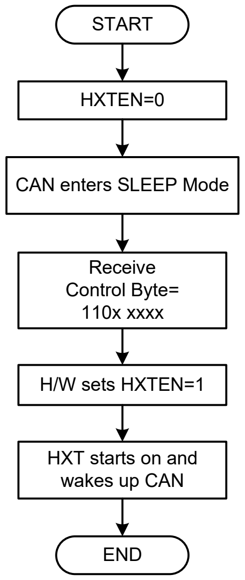
SLEEP Mode and Wake-up

The CAN bus controller has an internal SLEEP mode that is used to minimize the current consumption of the CAN bus controller. In the SLEEP Mode the oscillator is turned off.

To enter the SLEEP mode, the HXTEN bit in the FOCFG register should be cleared to zero. The master can wake up the CAN bus controller by sending a Wake-up command to set the HXTEN bit high and then read the CRHH register (Address: 0x3B) which indicates whether the HXT oscillator is stable after CAN_$t_{START(HXT)}$ time. If the CRHH has a value of 21H, it means the HXT oscillator is stable and the CAN bus controller is success fully waked-up by the master MCU from the SLEEP mode.


START
Control Byte=110x xxxx sent via the serial interface
Waiting CAN_t$_{START(HXT)}$ for CAN HXT stable
Read CAN CRHH register value
CRHH=0x21?
No
Yes
CAN HXT is stable
Next step

MCU


START
HXTEN=0
CAN enters SLEEP Mode
Receive Control Byte= 110x xxxx
H/W sets HXTEN=1
HXT starts on and wakes up CAN
END

CAN Bus Ccontroller

Functional Description

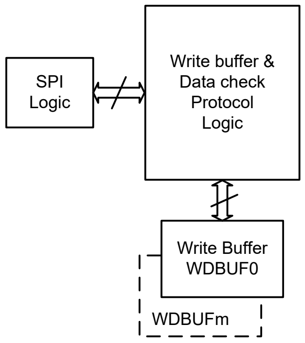
The Controller Area Network, or CAN bus for short, is a standard communication protocol used originally designed for automotive networking applications, however it is also used in other application areas such as industrial automation and some entertainment products. It is a two wire serial bus to which the CAN bus equipped products are connected together using twisted pair cable with a characteristic impedence of 120Ω.

Write Buffer and Data Check

The CAN bus controller provides a 32-byte Write Buffer with Data check function to easier the communication with the MCU.

The CAN bus controller contains an interface which is the four line SPI interface, to allow an easy method of communication with external Master devices. Having relatively simple communication protocol, the serial interface type allows CAN bus controller to interface to external SPI based microcontrollers.

The 32-byte Writter Buffer can be used to store the Control byte, Register Address byte and up to 31 bytes Data which are received or to be transmit in a communication.



Block Diagram

SPI Frame Fields

Following the Communication Protocol, the SPI frame should contain five data fields which are Control Byte, Register Address, Control CheckSum, Data and Data CheckSum.

- A 1-byte control field, including:
  - A 3-bit control instruction code defining the operation command.
  - A 5-bit data length code bits defining the size (in bytes) of the data field.
- A 1-byte Register Address field, defining the start register address to read from or to write into
- A 1-byte Control CheckSum field, Detecting errors during the control byte and the address byte transmission.The control checksum is based on a XOR operation
- A Data field of up to 31 bytes
- A 1-byte Data CheckSum field, Detecting errors during the data transmission.The control checksum is based on a XOR operation of all write/read data.

Control Byte

For each data transfer, a Control Byte is initiated to specify which Instruction is executed and how many bytes of data is transferred. The bit7~bit5 of the Control Byte, named INSTR[2:0], define the instruction while the bit4~bit0 of the Control Byte, named SDLC[4:0], is the data length code for setting the number of the data bytes to be received or transmitted.

1. Serial Data Length Code

The data byte numbers of 1~31 can be determined by programming the SDLC[4:0] bits in the control byte.

SDLC[4:0]	Serial Data Length	Description
00000	Not Valid	Not Valid
00001~11111	1~31	Valid programmed values 1~31

Note: The data length must be defined correctly when the INSTR[2:0] bits are set as 010 or 101. For other instructions, the defining of the data length is not required.

2. Instruction Control Code

The instruction is determined by the INSTR bit field of the control byte.

INSTR[2:0]	Instruction	Description
000 or 001 or 011	Not Valid	Not Valid
010	Write Data	To write data to Buffer
100	Read Status	Read CAN bus controller status
101	Read Data	To read CAN register data
110	Wake Up	Wake CAN Up
111	Reset	Reset CAN Block (can_reset) - Resets internal CAN registers to default state

Note: If set INSTR[2:0]=010 or 101, the SDLC[4:0] bits must be set correctly to define the length of the data to be read or written.

Instruction Description

The instruction byte is sent to the CAN bus controller via the SPI interface for different operations. Refer to "SPI Interface Timing Diagram" for detailed input and output timing diagram.

1. Write Data to Buffer

An instruction code of 010B should be transmitted to the CAN bus controller. The SDLC[4:0] bits in the control byte must be programmed correctly to define the data length to be written. The Register Address field defines the starting register address where the following data will be written into.The register address is automatically incremented by one to store the next byte of data until all the data bytes are written.

A byte Control byte, a byte address byte and the written data bytes will be written into the write buffer. Each transmission of the control and address bytes is followed by a XOR checksum byte for the control and address byte data. And another XOR checksum byte of the data bytes is transferred after the transmission of the data field for detecting errors during the data transmission.

2. Read CAN Register Data

An instruction code of 101B should be transmitted to the CAN bus controller. The SDLC[4:0] bits in the control byte must be programmed correctly to define the data length of reading CAN registers. The Address field byte defines the starting address of the CAN registers that the master wanted to read from. A control checksum byte which computes the exclusive or (XOR) of the control and address data is transmitted for detecting the control and address byte errors. After the control byte, address byte and the checksum byte are sent, the data stored in the registers at the selected starting address can stored in the buffer. The internal register address is automatically incremented by one to read the data and store it to the buffer until the defined SDLC[4+0] bytes of data all were read. And another checksum byte which computes the XOR of the read data is following the data field for detecting data errors. All the data will be shifed out on the MISO line. Then the master can read the data.

Before reading out the required register data, the first byte read by the master is a simple status byte which is used to determine whether the CAN bus controller is busy and the reading address is matched or not. Different values of the first byte and the corresponding status they indicate are summaried in the following table:

1st Byte=	Description
Write Address	It means the register address to read has been written correctly by the host MCU.
0xFD	Control CheckSum error. It means a error in writing the reading address
0xFE	CAN bus controller Busy
Others	Don't care

The "Read CAN Register Data" instruction is used to read the CAN bus controller register content and provide brief current error information about the CAN bus controller internal processing status such as access address error and CAN bus controller busy status.

3. Read Status

The Read Status Instruction allows single instruction access to some bits about the CAN bus controller status.

The part is selected by an instruction code of 100B transmitted to the CAN bus controller. After the read status instruction is sent by the host, the CAN bus controller will return eight bits of data that contain the status.

Status Byte	Description	Initial Value
Bit7	CAN Buffer Busy flag bit 0: Ready 1: Busy	0
Bit6	Transfer Error 0: Normal 1: INSTR[2:0] value in Control Byte invalid	0
Bit5	Control CheckSum Error 0: Ok 1: Error	0
Bit4	Data CheckSum Error 0: Ok 1: Error	0
Bit3	Bit3=$\overline{\text{Bit7}}$	1
Bit2	Bit2=$\overline{\text{Bit6}}$	1
Bit1	Bit1=$\overline{\text{Bit5}}$	1
Bit0	Bit0=$\overline{\text{Bit4}}$	1

In the status byte, Bit3~Bit0 is the Bit3=$\overline{\text{Bit7}}$, Bit2=$\overline{\text{Bit6}}$, Bit1=$\overline{\text{Bit5}}$ and Bit0=$\overline{\text{Bit4}}$. This four bit feild is for the purpose of detecting errors about the status bits. So the Checksum field can be omitted.

The Reading Status instruction should be executed in the following conditions:

- Before the initialisation after a reset, it needs first to determine whether the CAN bus controller is busy or not?
- Reading the CAN bus controller status instruction can be executed after writing data into important registers, to confirm the data was written correctly.
- When using the "Read CAN Register Data" instruction, if the first byte data received by the CAN bus controller has the value of 0xFD or 0xFE and users need complete error information, then the "Read Status Instruction" can be used to determine the actual condition, such as the CAN busy, Control CheckSum error, Transfer error or Data CheckSum error.

4. Wake CAN Up

If the HXTEN bit is cleared to zero, the HXT oscillator will stop and the CAN bus controller enters the Sleep Mode. To wake up the CAN bus controller, an instruction code of 110B can be sent. Refer to the "SLEEP Mode and Wake-up" section for detailed wake-up process.

It needs to note for the wake up instruction, the SDLC[4:0] bits in the Control field and the following four fields which are Register Address, Control CheckSum, Data and Data CheckSum are not required. But if the frame contains these four fields, the CAN bus controller will also save them into the buffer without processing them and an error will not happen.

5. Reset CAN Block Instruction

The Reset CAN Block Instruction can be used to re-initialise the internal CAN registers of the CAN bus controller to default state. Only an instruction code of 111B should be transmitted to the CAN bus controller for the reset operation. The SDLC[4:0] bits in the Control field and the following four fields which are Register Address, Control CheckSum, Data and Data CheckSum are not required. But if the frame contains these four fields, the CAN bus controller will also store them into the buffer without processing them and an error will not happen.

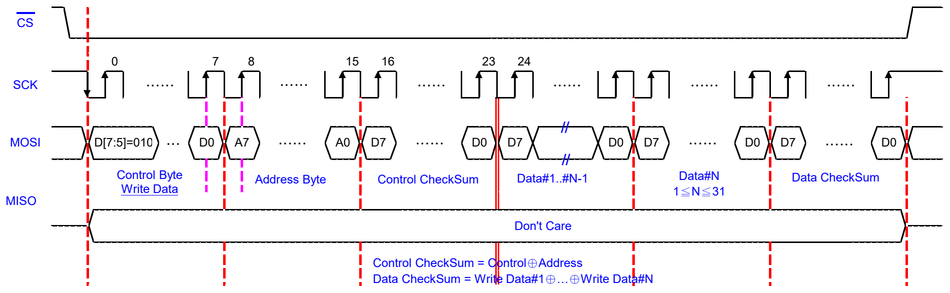
SPI Serial Interface

The CAN bus controller is designed to interface directly with the Serial Peripheral Interface (SPI) port available on many microcontrollers. Commands and data are sent to the CAN bus controller via the MOSI line, with data being clocked in on the rising edge of SCK. Data is driven out by the CAN bus controller, on the MISO line, on the falling edge of SCK. The $\overline{CS}$ line must be held low while any operation is performed. When raising the $\overline{CS}$ line from low to high, the SPI Interface will be reset.

Note: 1. Wait 10 HXT clocks for every 8 bits of command/position/data.
2. The MOSI, MISO, SCK and $\overline{CS}$ lines are internally connected to the MCU PC5, PC6, PD0 and PD1 lines respectively.

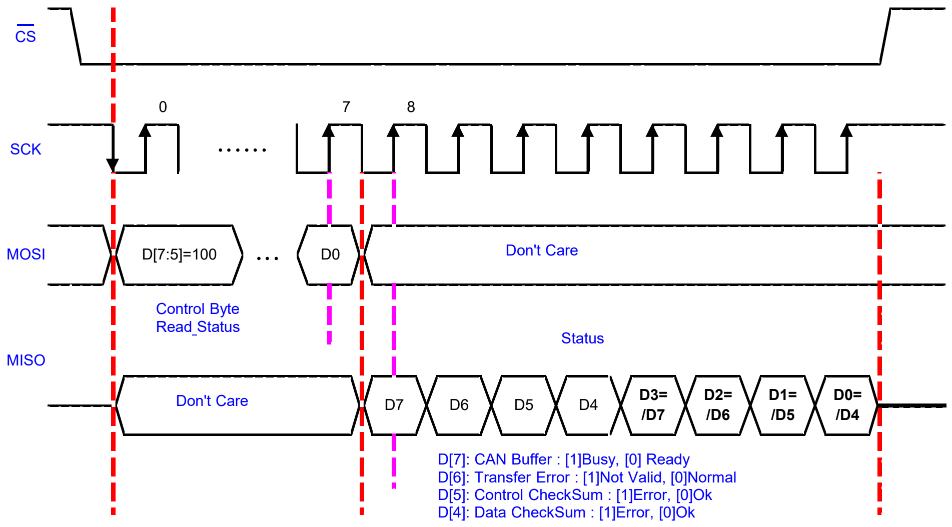
SPI Interface Timing Diagrams

1. Write Data to Buffer



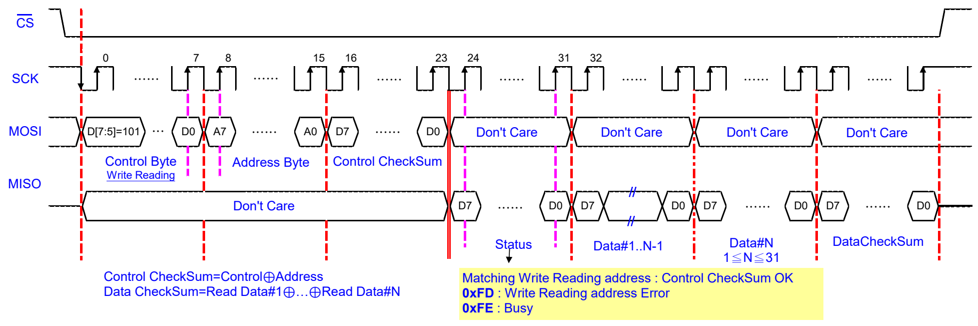
Write Data to Buffer Timing Diagram

2. Read Status


CS
SCK
MOSI
MISO
0
7
8
D[7:5]=100
D0
Don't Care
Control Byte Read_Status
Status
Don't Care
D7
D6
D5
D4
D3= /D7
D2= /D6
D1= /D5
D0= /D4
D[7]: CAN Buffer : [1]Busy, [0] Ready
D[6]: Transfer Error : [1]Not Valid, [0]Normal
D[5]: Control CheckSum : [1]Error, [0]Ok
D[4]: Data CheckSum : [1]Error, [0]Ok

Read Status Timing Diagram

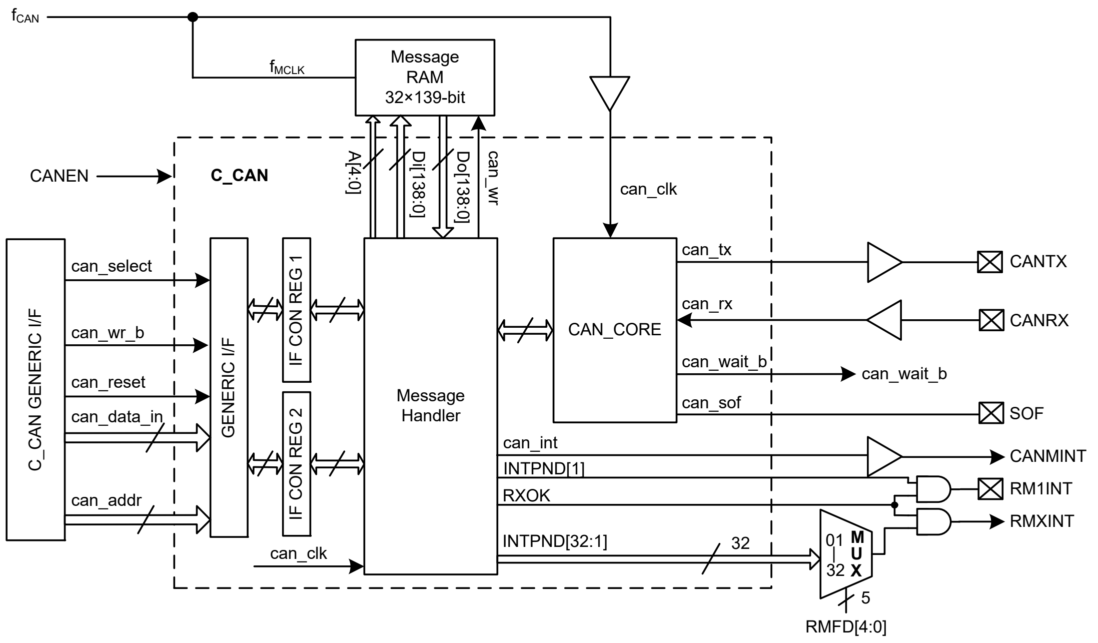
3. Read CAN Register Data


CS
SCK
MOSI
MISO
D[7:5]=101
Control Byte
Write Reading
Address Byte
Control CheckSum
Don't Care
Status
Data#1..N-1
Data#N
1≦N≦31
DataCheckSum
Control CheckSum=Control⊕Address
Data CheckSum=Read Data#1⊕…⊕Read Data#N
Matching Write Reading address : Control CheckSum OK
0xFD : Write Reading address Error
0xFE : Busy

Read CAN Register Data Timing Diagram

CAN Block Diagram

The CAN Module licensed from Bosch which supports CAN communication with up to 8 byte data fields. The device implements the CAN. As the same with the C_CAN, the CAN consists of the components CAN Core, Message RAM, Message Handler, Interface Control Register sets.



Note: The RMXINT and CANMINT lines are internally connected to the MCU PA7/INT2 and PC0 lines respectively.

CAN Block Diagram

1. C_CAN Core

Refer to the following Operating Description and Application section for further CAN module operation details. In this section we give a description about the functional blocks of the C_CAN:

- CAN_Core

  The CAN_Core performs communication according to the CAN protocol version 2.0 A, B and ISO 11898-1.

- Registers

  All registers are used to control and to configure the module.
- Message Handler

  The internal State Machine controls the data transfer between the RX/TX Shift Register of the CAN_Core and the Message RAM as well as the generation of interrupts as programmed in the Control and Configuration Registers. All functions concerning the handling of messages are implemented in the Message Handler. Those functions are the acceptance filtering, the transfer of messages between the CAN Core and the Message RAM, and the handling of transmission requests as well as the generation of the module interrupt.
- Interface Control Register 1 and 2

  The function of the two interface control register sets is identical (except in Basic mode). The interface control register sets are used for the data transfer between the external bus and the Message RAM.
- Message RAM Interface

  Message RAM size: 139-bits×32

2. Message RAM

- Stores 32 Message Objects and Identifier Masks.
- Each Message Object together with Identifier Mask has a length of 139 bits.

Interrupt Outputs

The CAN bus controller has three interrupt outputs, RM1INT pin, CANMINT line and RMXINT line, to be used to indicate different conditions. When a CAN interrupt occurs, the CANMINT line will be driven an active level by the CAN bus controller. When a Message is received into Message Object 1 successfully, an interrupt occurs and the RM1INT pin will output an active level. When a Message is received into Message Object x successfully, an interrupt occurs and the RMXINT line will output an active level.

Interrupt active level can be selected to be high or low using the FOCFG register bits.

Message RAM and FIFO Buffer Configuration

For communication on a CAN network, individual Message Objects are configured. The Message Objects and Identifier Masks for acceptance filtering of received messages are stored in the Message RAM. The CAN bus controller includes a Message Memory capacity of 139-bit ×32 for storing 32 Message Objects and Identifier Masks. A Message Objects and Identifier Masks is 139 bits which is shown in the following table.

Structure of a Message Object in the Message RAM							
MSKn28~00	MXTDn	MDIRn	UMASKn	TXnIEN	RXnIEN	RMTnEN	EOBn
IDn28~00	XTDn	DIRn	MSGnLST	—	—	—	DLCn[3:0]
DATA0	DATA1	DATA2	DATA3	DATA4	DATA5	DATA6	DATA7

MSKn28~00 Identifier Mask
0: The corresponding bit in the identifier of the message object cannot inhibit the match in the acceptance filtering
1: The corresponding identifier bit is used for acceptance filtering

IDn28~00 Message Identifier
IDn28~IDn00: 29-bit Identifier ("Extended Frame").
IDn28~IDn18: 11-bit Identifier ("Standard Frame").

MXTDn Mask Extended Identifier
0: The extended identifier bit (IDE) has no effect on the acceptance filtering
1: The extended identifier bit (IDE) is used for acceptance filtering

Note: When 11-bit (“standard”) Identifiers are used for a Message Object, the identifiers of received Data Frames are written into bits IDn28 to IDn18. For acceptance filtering, only these bits together with MASK bits MSKn28 to MSKn18 are considered.

XTDn Extended Identifier
0: The 11-bit (“standard”) Identifier will be used for this Message Object
1: The 29-bit (“extended”) Identifier will be used for this Message Object

MDIRn Mask Message Direction
0: The message direction bit (DIR) has no effect on the acceptance filtering
1: The message direction bit (DIR) is used for acceptance filtering

Note: The Arbitration Registers IDn28-00, XTDn, and DIRn are used to define the identifier and type of outgoing messages and are used (together with the mask registers MSKn28-00, MXTDn, and MDIRn) for acceptance filtering of incoming messages. A received message is stored into the valid Message Object with matching identifier and Direction=receive (Data Frame) or Direction=transmit (Remote Frame). Extended frames can be stored only in Message Objects with XTDn=one, standard frames in Message Objects with XTDn=zero. If a received message (Data Frame or Remote Frame) matches with more than one valid Message Object, it is stored into that with the lowest message number. For details see chapter Acceptance Filtering of Received Messages.

DIRn Message Direction
0: Direction=receive: On TREQ, a Remote Frame with the identifier of this Message Object is transmitted. On reception of a Data Frame with matching identifier, that message is stored in this Message Object.
1: Direction=transmit: On TREQ, the respective Message Object is transmitted as a Data Frame. On reception of a Remote Frame with matching identifier, the TREQ bit of this Message Object is set (if RMTnEN=one).

UMASKn Use Acceptance Mask
0: MASK ignored.
1: Use MASK (MSKn28~00, MXTDn and MDIRn) for acceptance filtering.

If the UMASKn bit is set to one, the Message Object’s mask bits have to be programmed during initialization of the Message Object before MSGnVA is set to one.

MSGnLST Message Lost (only valid for Message Objects with direction=receive)
0: No message lost since last time this bit was reset by the CPU.
1: The Message Handler stored a new message into this object when NDTA was still set, the CPU has lost a message.

TXnIEN Transmit Interrupt Enable
0: INTPND will be left unchanged after the successful transmission of a frame.
1: INTPND will be set after a successful transmission of a frame.

RXnIEN Receive Interrupt Enable
0: INTPND will be left unchanged after a successful reception of a frame.
1: INTPND will be set after a successful reception of a frame.

RMTnEN Remote Enable
0: At the reception of a Remote Frame, TREQ is left unchanged.
1: At the reception of a Remote Frame, TREQ is set.

EOBn End of Buffer
0: Message Object belongs to a FIFO Buffer and is not the last Message Object of that FIFO Buffer.
1: Single Message Object or last Message Object of a FIFO Buffer.

This bit is used to concatenate two ore more Message Objects (up to 32) to build a FIFO Buffer.

For single Message Objects (not belonging to a FIFO Buffer) this bit must always be set to one.

DLCn3~0	Data Length Code 0~8: CAN: Frame has 0-8 data bytes 9~15: CAN: Frame has 8 data bytes Note: The Data Length Code of a Message Object must be defined the same as in all the corresponding objects with the same identifier at other nodes. When the Message Handler stores a data frame, it will write the DLC to the value given by the received message.
DATA0	1st data byte of a CAN Data Frame
DATA1	2nd data byte of a CAN Data Frame
DATA2	3rd data byte of a CAN Data Frame
DATA3	4th data byte of a CAN Data Frame
DATA4	5th data byte of a CAN Data Frame
DATA5	6th data byte of a CAN Data Frame
DATA6	7th data byte of a CAN Data Frame
DATA7	8th data byte of a CAN Data Frame Note: Byte DATA0 is the first data byte shifted into the shift register of the CAN Core during a reception, byte DATA7 is the last. When the Message Handler stores a Data Frame, it will write all the eight data bytes into a Message Object. If the Data Length Code is less than 8, the remaining bytes of the Message Object will be overwritten by non specified values.

The 32 Message Objects can be configurated to several sets of FIFO buffer. A FIFO buffer can have a single Message Object or several concatenated Message Objects. The FIFO threshold of the Message Object number is determined by the RMFD[4:0] bits in the CAN Configuration Register, CANCFG. When the Message Object of the selected number is received successfully, an interrupt active signal will output on the RMXINT line.

CAN Operating Modes

The Operating modes can be controlled by the registers. Detailed informations about the operating modes refer to the following contents and the related registers.

Software Initialization

The software initialization is started by setting the bit INIT in the CAN Control Register, either by software or by a hardware reset, or by going Bus_Off.

While INIT is set, all message transfered from and to the CAN bus is stopped, the status of the CAN bus output can_tx is recessive (HIGH). The counters of the EML (Error Management Logic) are unchanged. Setting INIT does not change any configuration register.

To initialize the CAN Controller, the CPU has to set up the Bit Timing Register and each Message Object. If a Message Object is not needed, it is sufficient to set it's MSGnVA bit to not valid. Otherwise, the whole Message Object has to be initialized.

Access to the Bit Timing Register and to the BRP Extension Register for the configuration of the bit timing is enabled when both bits INIT and CCE in the CAN Control Register are set.

Resetting INIT (by CPU only) finishes the software initialization. Afterwards the Bit Stream Processor (BSP) synchronizes itself to the data transfer on the CAN bus by waiting for the occurrence of a sequence of 11 consecutive recessive bits (≡ Bus Idle) before it can take part in bus activities and starts the message transfer.

The initialization of the Message Objects is independent of INIT and can be done on the fly, but the Message Objects should all be configured to particular identifiers or set to not valid before the BSP starts the message transfer. To change the configuration of a Message Object during normal operation, the CPU has to start by setting MSGnVA to not valid. When the configuration is completed, MSGnVA is set to valid again.

CAN Message Transfer

Once the CAN is initialized and INIT is reset to zero, the CAN Core synchronizes itself to the CAN bus and starts the message transfer.

Received messages are stored into their appropriate Message Objects if they pass the Message Handler's acceptance filtering. The whole message including all arbitration bits, DLC and eight data bytes is stored into the Message Object. If the Identifier Mask is used, the arbitration bits which are masked to "don't care" may be overwritten in the Message Object.

The CPU may read or write each message any time via the Interface Registers, the Message Handler guarantees data consistency in case of concurrent accesses.

Messages to be transmitted are updated by the CPU. If a permanent Message Object (arbitration and control bits set up during configuration) exists for the message, only the data bytes are updated and then TQnDTA bit is set to start the transmission. If several transmit messages are assigned to the same Message Object (when the number of Message Objects is not sufficient), the whole Message Object has to be configured before the transmission of this message is requested.

The transmission of any number of Message Objects may be requested at the same time, they are transmitted subsequently according to their internal priority. Messages may be updated or set to not valid any time, even when their requested transmission is still pending. The old data will be discarded when a message is updated before its pending transmission has started.

Depending on the configuration of the Message Object, the transmission of a message may be requested autonomously by the reception of a remote frame with a matching identifier.

Note: Remote frames are always transmitted in Classical CAN format.

Disabled Automatic Retransmission

According to the CAN Specification (see ISO11898-1, 6.3.3 Recovery Management), the CAN provides means for automatic retransmission of frames that have lost arbitration or that have been disturbed by errors during transmission. The frame transmission service will not be confirmed to the user before the transmission is successfully completed. By default, this means for automatic retransmission is enabled.

The Disabled Automatic Retransmission mode is enabled by programming bit DAR in the CAN Control Register to '1'. In this operation mode the programmer has to consider the different behaviour of bits TREQ and NDTA in the Control Registers of the Message Buffers:

- When a transmission starts, bit TREQ of the respective Message Buffer is reset, while bit NDTA remains set.
- When the transmission completed successfully bit NDTA is reset.

When a transmission failed (lost arbitration or error) bit NDTA remains set. To restart the transmission the CPU has to set TREQ back to '1'.

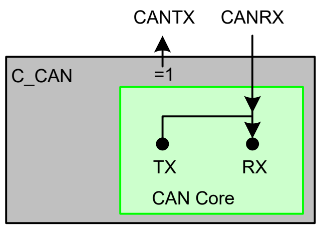
Test Mode

The Test Mode is entered by setting bit TEST in the CAN Control Register to one. In Test Mode the bits TX1, TX0, LBACK, SILENT and BASIC in the Test Register are writable. Bit RX monitors the state of pin CANRX and therefore is only readable. All Test Register functions are disabled when bit TEST is reset to zero. The Test Mode functions as described in the following subsections are intended for device tests outside normal operation. These functions should be used carefully. Switching between Test Mode functions and normal operation while communication is running (INIT='0') should be avoided.

1. Silent Mode

In ISO 11898-1, the Silent Mode is called the Bus Monitoring Mode. The CAN Core can be set in Silent Mode by programming the Test Register bit SILENT to ‘1’.

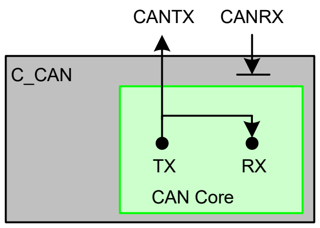
In Silent Mode, the CAN is able to receive valid data frames and valid remote frames, but it sends only recessive bits on the CAN bus and it cannot start a transmission. If the CAN Core is required to send a dominant bit (ACK bit, overload flag, active error flag), the bit is rerouted internally so that the CAN Core monitors this dominant bit, although the CAN bus may remain in recessive state. The Silent Mode can be used to analyse the traffic on a CAN bus without affecting it by the transmission of dominant bits (Acknowledge Bits, Error Frames).



CAN Core in Silent Mode

2. Loop Back Mode

The CAN Core can be set in Loop Back Mode by programming the Test Register bit LBACK to ‘1’. In Loop Back Mode, the CAN Core treats its own transmitted messages as received messages and stores them (if they pass acceptance filtering) into a Receive Buffer.

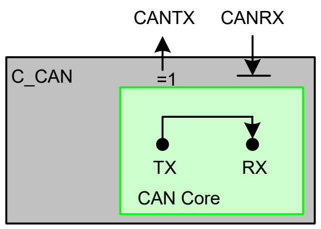


CAN Core in Loop Back Mode

This mode is provided for self-test functions. To be independent from external stimulation, the CAN Core ignores acknowledge errors (recessive bit sampled in the acknowledge slot of a data/remote frame) in Loop Back Mode. In this mode the CAN Core performs an internal feedback from its TX output to its RX input. The actual value of the CANRX input pin is disregarded by the CAN Core. The transmitted messages can be monitored at the CANTX pin.

3. Loop Back combined with Silent Mode

It is also possible to combine Loop Back Mode and Silent Mode by programming both bits LBACK and SILENT to ‘1’ at the same time. This mode can be used for a “Hot Selftest”, meaning the CAN can be tested without affecting a running CAN system connected to the pins CANTX and CANRX. In this mode the CANRX pin is disconnected from the CAN Core and the CANTX pin is held recessive.



CAN Core in Loop Back combined with Silent Mode

4. Basic Mode

The CAN Core can be set in Basic Mode by programming the Test Register bit BASIC to ‘1’. In this mode the CAN module runs without the Message RAM.

The IF1 Registers are used as Transmit Buffer. The transmission of the contents of the IF1 Registers is requested by writing the BUSYn bit of the IF1 Command Request Register to ‘1’. The IF1 Registers are locked while the BUSYn bit is set. The BUSYn bit indicates that the transmission is pending. As soon the CAN bus is idle, the IF1 Registers are loaded into the shift register of the CAN Core and the transmission is started. When the transmission has completed, the BUSYn bit is reset and the locked IF1 Registers are released. A pending transmission can be aborted at any time by resetting the BUSYn bit in the IF1 Command Request Register while the IF1 Registers are locked. If the CPU has reset the BUSYn bit, a possible retransmission in case of lost arbitration or in case of an error is disabled.

The IF2 Registers are used as Receive Buffer. After the reception of a message the contents of the shift register is stored into the IF2 Registers, without any acceptance filtering. Additionally, the actual contents of the shift register can be monitored during the message transfer. Each time a read Message Object is initiated by writing the BUSYn bit of the IF2 Command Request Register to ‘1’, the contents of the shift register is stored into the IF2 Registers.

In Basic Mode the evaluation of all Message Object related control and status bits and of the control bits of the IFn Command Mask Registers is turned off. The message number of the Command request registers is not evaluated. The NnDTA and MSGnLST bits of the IF2 Message Control Register retain their function, DLCn3~DLCn0 will show the received DLC(Data Length Code), the other control bits will be read as ‘0’.

In Basic Mode the ready output can_wait_b is not active.

Software control of Pin CANTX

Four output functions are available for the CAN transmit pin CANTX. Additionally to its default function – the serial data output – it can drive the CAN Sample Point signal to monitor CAN_Core’s bit timing and it can drive constant dominant or recessive values. The last two functions, combined with the readable CAN receive pin CANRX, can be used to check the CAN bus’ physical layer.

The output mode of pin CANTX is selected by programming the Test Register bits TX1 and TX0. The three test functions for pin CANTX interfere with all CAN protocol functions. CANTX must be left in its default function when CAN message transfer or any of the test modes Loop Back Mode, Silent Mode, or Basic Mode are selected.

CAN Application

Management of Message Objects

The configuration of the Message Objects in the Message RAM will (with the exception of the bits MSGVA, NDTA, INTPND, and TREQ) not be affected by resetting the CAN. All the Message Objects must be initialized by the CPU or they must be set not valid (MSGVA=‘0’). The bit timing must be configured before the CPU clears the INIT bit in the CAN Control Register.

The configuration of a Message Object is done by programming Mask, Arbitration, Control and Data field of one of the two interface register sets to the desired values. By writing to the corresponding IFn Command Request Register, the IFn Message Buffer Registers are loaded into the addressed Message Object in the Message RAM.

When the INIT bit in the CAN Control Register is cleared, the CAN Protocol Controller state machine of the CAN_Core and the Message Handler State Machine control the CAN’s internal data flow. Received messages that pass the acceptance filtering are stored into the Message RAM, messages with pending transmission request are loaded into the CAN_Core's Shift Register and are transmitted via the CAN bus.

The CPU reads received messages and updates messages to be transmitted via the IFn Interface Registers. Depending on the configuration, the CPU is interrupted on certain CAN message and CAN error events.

Message Handler State Machine

The Message Handler controls the data transfer between the Rx/Tx Shift Register of the CAN Core, the Message RAM and the IFn Registers.

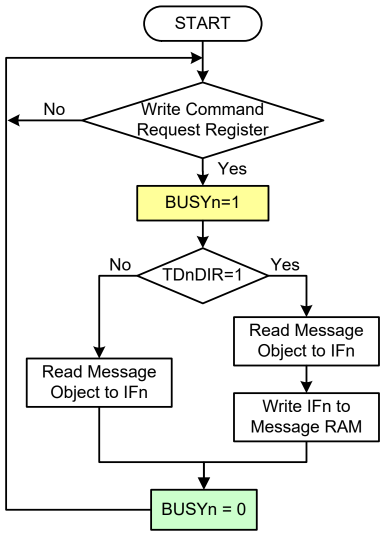
The Message Handler FSM (Finite State Machine) controls the following functions:

- Data Transfer from IFn Registers to the Message RAM
- Data Transfer from Message RAM to the IFn Registers
- Data Transfer from Shift Register to the Message RAM
- Data Transfer from Message RAM to Shift Register
- Data Transfer from Shift Register to the Acceptance Filtering unit
- Scanning of Message RAM for a matching Message Object
- Handling of TREQ flags
- Handling of interrupts

Data Transfer from/to Message RAM

When the CPU initiates a data transfer between the IFn Registers and Message RAM, the Message Handler sets the BUSYn bit in the respective IFn Command Request Register to '1'. After the transfer has completed, the BUSYn bit is set back to '0'.

The respective IFn Command Mask Register specifies whether a complete Message Object or only parts of it will be transferred. Due to the structure of the Message RAM it is not possible to write single bits/bytes of one Message Object, it is always necessary to write a complete Message Object to the Message RAM. Therefore the data transfer from the IFn Message Buffer Registers to the Message RAM (TDnDIR="1") requires a read-modify-write cycle. First those parts of the Message Object that are not to be changed are read from the Message RAM to the selected IFn Message Buffer Registers and then the complete contents of the selected IFn Message Buffer Registers are written to the Message Object.


START
Write Command Request Register
No
Yes
BUSYn=1
TDnDIR=1
No
Yes
Read Message Object to IFn
Read Message Object to IFn
Write IFn to Message RAM
BUSYn = 0

Data Transfer between IFn Registers and Message RAM

After a partial write of a Message Object (TDnDIR="1"), the IFn Message Buffer Registers that are not selected by the respective IFn Command Mask Register will be set to the actual contents of the selected Message Object.

After a partial read of a Message Object (TDnDIR="0"), the IFn Message Buffer Registers that are not selected by the respective IFn Command Mask Register will be left unchanged.

Transmission of Messages

If the shift register of the CAN Core cell is ready for loading and if there is no data transfer between the IFn Registers and Message RAM, the MSGVA bits in the Message Valid Register and the TREQ bits in the Transmission Request Register are evaluated. The valid Message Object with the highest priority pending transmission request is loaded into the shift register by the Message Handler and the transmission is started. The Message Object's NDTA bit is reset.

After a successful transmission and if no new data was written to the Message Object (NDTA='0') since the start of the transmission, the TREQ bit will be reset. If TXnIEN is set, INTPND will be set after a successful transmission. If the CAN has lost the arbitration or if an error occurred during the transmission, the message will be retransmitted as soon as the CAN bus is free again. If meanwhile the transmission of a message with higher priority has been requested, the messages will be transmitted in the order of their priority.

Acceptance Filtering of Received Messages

When the arbitration and control field (Identifier+DLC) of an incoming message is completely shifted into the Rx/Tx Shift Register of the CAN Core, the Message Handler FSM (Finite State Machine) starts the scanning of the Message RAM for a matching valid Message Object.

To scan the Message RAM for a matching Message Object, the Acceptance Filtering unit is loaded with the arbitration bits from the CAN Core shift register. Then the arbitration and mask fields (including MSGVA, UMASKn, NDTA, and EOBn) of Message Object 1 are loaded into the Acceptance Filtering unit and are compared with the arbitration field from the shift register. This is repeated with each following Message Object until a matching Message Object is found or until the end of the Message RAM is reached.

If a match occurs, the scanning is stopped and the Message Handler FSM (Finite State Machine) proceeds depending on the type of frame (Data Frame or Remote Frame) received.

Reception of Data Frame

The Message Handler FSM (Finite State Machine) stores the message from the CAN Core shift register into the respective Message Object in the Message RAM. Not only the data bytes, but also all arbitration bits and the Data Length Code are stored into the corresponding Message Object. This is implemented to keep the data bytes connected with the identifier even if arbitration mask registers are used.

The NDTA bit is set to indicate that new data (not yet seen by the CPU) has been received. The CPU should reset NDTA when it reads the Message Object. If at the time of the reception, the NDTA bit was already set, MSGLST is set to indicate that the previous data (supposedly not seen by the CPU) is lost. If the RXnIE bit is set, the INTPND bit is set, causing the Interrupt Register to point to this Message Object.

The TREQ bit of this Message Object is reset to prevent the transmission of a Remote Frame, while the requested Data Frame has just been received.

Reception of Remote Frame

When a Remote Frame is received, three different configurations of the matching Message Object have to be considered:

1) DIRn=‘1’ (direction=transmit), RMTnEN=‘1’, UMASKn=‘1’ or ‘0’

At the reception of a matching Remote Frame, the TREQ bit of this Message Object is set high.

The rest of the Message Object remains unchanged.

2) DIRn=‘1’ (direction=transmit), RMTnEN=‘0’, UMASKn=‘0’

At the reception of a matching Remote Frame, the TREQ bit of this Message Object remains unchanged; the Remote Frame is ignored.

3) DIRn=‘1’ (direction=transmit), RMTnEN=‘0’, UMASKn=‘1’

At the reception of a matching Remote Frame, the TREQ bit of this Message Object is reset. The arbitration and control field (Identifier + IDE + RTR + DLC) from the shift register is stored into the Message Object in the Message RAM and the NDTA bit of this Message Object is set high.

Note: Remote frames are always transmitted in Classical CAN format.

Receive/Transmit Priority

The receive/transmit priority for the Message Objects is attached to the message number. Message Object 1 has the highest priority, while Message Object 32 has the lowest priority. If more than one transmission request is pending, they are serviced according to the priority of the corresponding Message Object.

Configuration of a Transmit Object

MSGnVA	ARBn	—	DATA	MASK	EOBn	DIRn
1	appl.	—	appl.	appl.	1	1
NnDTA	**MSGnLST**	**RXnIEN**	**TXnIEN**	**INTnPND**	**RMTnEN**	**TnREQ**
0	0	0	appl.	0	appl.	0

Note: “appl.” means by application.

Initialisation of a Transmit Object

The Arbitration Registers (IDn28~00 and XTDn bit) are given by the application. They define the identifier and type of the outgoing message. If an 11-bit Identifier (“Standard Frame”) is used, it is programmed to IDn28~IDn18, IDn17~IDn00 can then be disregarded.

If the TXnIEN bit is set, the INTPND bit will be set after a successful transmission of the Message Object.

If the RMTnEN bit is set, a matching received Remote Frame will cause the TREQ bit to be set; the Remote Frame will autonomously be answered by a Data Frame.

The Data Registers (DLCn3-0, DATA0-7) are given by the application, TnREQ and RMTnEN may not be set before the data is valid.

The Mask Registers (MSKn28-00, UMASKn, MXTDn and MDIRn bits) may be used (UMASKn =‘1’) to allow groups of Remote Frames with similar identifiers to set the TnREQ bit. The DIRn bit should not be masked.

Updating a Transmit Object

The CPU may update the data bytes of a Transmit Object any time via the IFn Interface registers, neither MSGVA nor TREQ have to be reset before the update.

Even if only a part of the data bytes are to be updated, all four bytes of the corresponding IFn DATAnA Register or IFn DATAnB Register have to be valid before the content of that register is transferred to the Message Object. Either the CPU has to write all four bytes into the IFn Data Register or the Message Object is transferred to the IFn Data Register before the CPU writes the new data bytes.

When only the (eight) data bytes are updated, first 0x87 is written to the IFn Command Mask Register and then the number of the Message Object is written to the IFn Command Request Register, concurrently updating the data bytes and setting TQnDTA.

To prevent the reset of TREQ at the end of a transmission that may already be in progress while the data is updated, NDTA has to be set together with TREQ.

When NDTA is set together with TREQ, NDTA will be reset as soon as the new transmission has started.

Configuration of a Receive Object

MSGnVA	ARBn	—	DATA	MASK	EOBn	DIRn
1	appl.	—	appl.	appl.	1	0
NnDTA	MSGnLST	RXnIEN	TXnIEN	INTnPND	RMTnEN	TnREQ
0	0	appl.	0	0	0	0

Note: “appl.” means by application.

Initialisation of a Receive Object

The Arbitration Registers (IDn[28:00] and XTDn bit) are given by the application. They define the identifier and type of accepted received messages. If an 11-bit Identifier (“Standard Frame”) is used, it is programmed to IDn28~IDn18, IDn17~IDn00 can then be disregarded. When a Data Frame with an 11-bit Identifier is received, IDn17~IDn00 will be set to ‘0’.

If the RXnIEN bit is set, the INTPND bit will be set when a received Data Frame is accepted and stored in the Message Object.

The Data Length Code (DLCn[3:0]) is given by the application. When the Message Handler stores a Data Frame in the Message Object, it will store the received Data Length Code and eight data bytes. If the Data Length Code is less than 8, the remaining bytes of the Message Object will be overwritten by non specified values.

The Mask Registers (MSKn[28:00], UMASKn, MXTDn, and MDIRn bits) may be used (UMASKn =‘1’) to allow groups of Data Frames with similar identifiers to be accepted . The DIRn bit should not be masked in typical applications.

Handling of Received Messages

The CPU may read a received message any time via the IFn Interface registers. The data consistency is guaranteed by the Message Handler state machine.

Typically the CPU will write first 0x7F to the IFn Command Mask Register and then the number of the Message Object to the IFn Command Request Register. That combination will transfer the whole received message from the Message RAM into the IFn Message Buffer Register. Additionally, the bits NDTA and INTPND are cleared in the Message RAM (not in the Message Buffer).

If the Message Object uses masks for acceptance filtering, the arbitration bits show which of the matching messages has been received.

The actual value of NDTA shows whether a new message has been received since last time this Message Object was read. The actual value of MSGLST shows whether more than one message has been received since last time this Message Object was read. MSGLST will not be automatically reset.

By means of a Remote Frame, the CPU may request another CAN node to provide new data for a receive object. Setting the TREQ bit of a receive object will cause the transmission of a Remote Frame with the receive object's identifier. This Remote Frame triggers the other CAN node to start the transmission of the matching Data Frame. If the matching Data Frame is received before the Remote Frame could be transmitted, the TREQ bit is automatically reset.

Configuration of a FIFO Buffer

With the exception of the EOBn bit, the configuration of Receive Objects belonging to a FIFO Buffer is the same as the configuration of a (single) Receive Object.

To concatenate two or more Message Objects into a FIFO Buffer, the identifiers and masks (if used) of these Message Objects have to be programmed to matching values. Due to the implicit priority of the Message Objects, the Message Object with the lowest number will be the first Message Object of the FIFO Buffer. The EOBn bit of all Message Objects of a FIFO Buffer except the last have to be programmed to zero. The EOBn bits of the last Message Object of a FIFO Buffer is set to one, configuring it as the End of the Block.

Reception of Messages with FIFO Buffers

Received messages with identifiers matching to a FIFO Buffer are stored into a Message Object of this FIFO Buffer starting with the Message Object with the lowest message number.

When a message is stored into a Message Object of a FIFO Buffer the NDTA bit of this Message Object is set. By setting NDTA while EOBn is zero the Message Object is locked for further write accesses by the Message Handler until the CPU has written the NDTA bit back to zero.

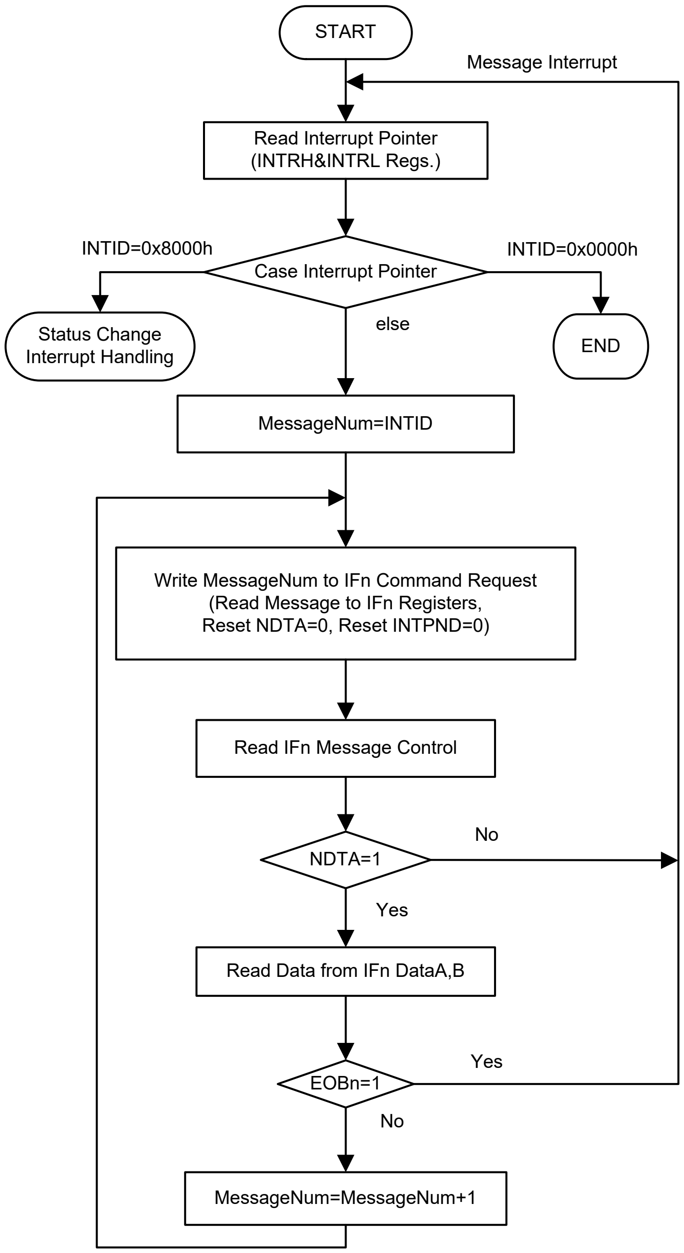
Messages are stored into a FIFO Buffer until the last Message Object of this FIFO Buffer is reached. If none of the preceding Message Objects is released by writing NDTA to zero, all further messages for this FIFO Buffer will be written into the last Message Object of the FIFO Buffer and therefore overwrite previous messages.

Reading from a FIFO Buffer

When the CPU transfers the contents of Message Object to the IFn Message Buffer Registers by writing its number to the IFn Command Request Register, the corresponding IFn Command Mask Register should be programmed the way that bits NDTA and INTPND are reset to zero (TQnDTA='1' and CINTPNDn='1'). The values of these bits in the IFn Message Control Register always reflect the status before resetting the bits.

To assure the correct function of a FIFO Buffer, the CPU should read out the Message Objects starting at the FIFO Object with the lowest message number.

The following figure shows how a set of Message Objects which are concatenated to a FIFO Buffer can be handled by the CPU.


START
Message Interrupt
Read Interrupt Pointer (INTRH&INTRL Regs.)
INTID=0x8000h
Case Interrupt Pointer
INTID=0x0000h
Status Change Interrupt Handling
else
END
MessageNum=INTID
Write MessageNum to IFn Command Request (Read Message to IFn Registers, Reset NDTA=0, Reset INTPND=0)
Read IFn Message Control
No
NDTA=1
Yes
Read Data from IFn DataA,B
Yes
EOBn=1
No
MessageNum=MessageNum+1

CPU Handling of a FIFO Buffer

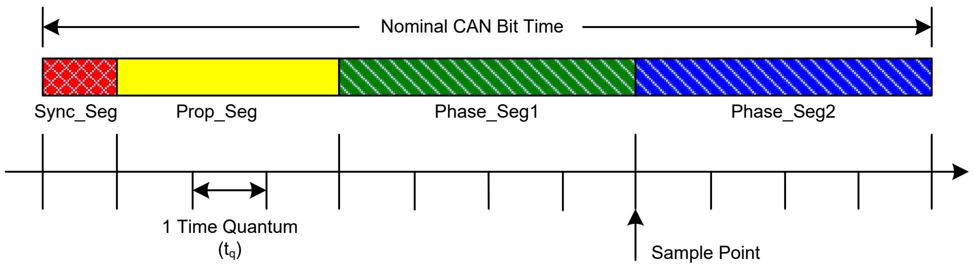
Bit Time and Bit Rate

CAN supports bit rates in the range of 1 kBit/s to 1000 kBit/s. Each member of the CAN network has its own clock generator, usually a quartz oscillator. The timing parameter of the bit time (i.e. the reciprocal of the bit rate) can be configured individually for each CAN node, creating a common bit rate even though the CAN nodes' oscillator periods (f_{OSC}) may be different.

The frequencies of these oscillators are not absolutely stable, small variations are caused by changes in temperature or voltage and by deteriorating components. As long as the variations remain inside a specification oscillator tolerance range (d_F), the CAN nodes are able to compensate for the different bit rates by resynchronising to the bit stream.

According to the CAN specification, the bit time is divided into four segments which are the Synchronisation Segment, the Propagation Time Segment, the Phase Buffer Segment 1 and the Phase Buffer Segment 2. Each segment consists of a specification, programmable number of time quanta. The length of the time quantum (t_q), which is the basic time unit of the bit time, is defined by the CAN controller's system clock CAN_f_{SYS} and the Baud Rate Prescaler (BRP): t_q=BRP/CAN_f_{SYS}. The CAN's system clock CAN_f_{SYS} is the frequency of its can_clk input.

The Synchronisation Segment Sync_Seg is that part of the bit time where edges of the CAN bus level are expected to occur; the distance between an edge that occurs outside of Sync_Seg and the Sync_Seg is called the phase error of that edge. The Propagation Time Segment Prop_Seg is intended to compensate for the physical delay times within the CAN network. The Phase Buffer Segments Phase_Seg1 and Phase_Seg2 surround the Sample Point. The (Re-) Synchronisation Jump Width (SJW) defines how far a resynchronisation may move the Sample Point inside the limits defined by the Phase Buffer Segments to compensate for edge phase errors.



Bit Timing

Parameter	Range	Remark
BRP	[1 .. 32]	Defines the length of the time quantum t_q
Sync_Seg	1 t_q	Fixed length, synchronisation of bus input to CAN system clock
Prop_Seg	[1 .. 8] t_q	Compensates for the physical delay times
Phase_Seg1	[1 .. 8] t_q	May be lengthened temporarily by synchronisation
Phase_Seg2	[1 .. 8] t_q	May be shortened temporarily by synchronisation
SJW	[1 .. 4] t_q	May not be longer than either Phase Buffer Segment
This table describes the minimum programmable ranges required by the CAN protocol		

mtq (minimum time quantum)=system clock period=1/CAN_f_{SYS}

t_q (time quantum)=(BRPE[3:0]×0x40+BRP[5:0]+1)×mtq

SYNC_SEG=1 t_q

SEG1=PROP_SEG+PHASE_SEG1

Bit Time=$t_{SYNC\_SEG}+t_{SEG1}+t_{PHASE\_SEG2}$

For example:

CAN_f_{SYS}=8MHz, Bit Rate=500Kbps, PROP_SEG=0, Sample point=50%, SYNC_SEG=1 t_q, then to calculate the SEG1 & PHASE_SEG2 values.

Sol: mtq=1/CAN_f_{SYS}=1/8MHz=0.125μs

set Baud Rate Prescaler (BRP)=1 → BRP[5:0]=(1−1)=$0_{\#}$

t_q=(BRPE[3:0] • 0x40+BRP[5:0]+1) • mtq=1 • mtq=0.125μs

Bit Time=1/Bit Rate=1/500Kbps=0.002ms=2μs

Nominal Bit Time=Bit Rate/tq=2μs/(0.125μs)=16tq

(1) PHASE_SEG2=Nominal Bit Time−(NominalBit Time • Sample point)=16t_q−(16t_q • 50%)=16t_q−8t_q=8t_q

TSG2D[2:0]=(8−1)=$7_{\#}$

(2) SEG1=(Nominal Bit Time−SYNC_SEG−PHASE_SEG2)=(16t_q–1t_q–8t_q)=7t_q

TSG1D[3:0]=(7−1)=$6_{\#}$

Register Description

The CAN bus controller is controlled using a series of registers which are described in the following section.

Register Map

The following shows the full register bit map of the CAN bus controller.

Note that the symbol "—" represents an unimplemented bit which is read as zero.

Address	Register Name	Bit							
		7	6	5	4	3	2	1	0
0x00	CTRLRL	TEST	CCE	DAR	—	EIE	SIE	CANIE	INIT
0x02	STATRL	BOFF	EWARN	EPASS	RXOK	TXOK	LEC2	LEC1	LEC0
0x04	ERRCNTL	TEC7	TEC6	TEC5	TEC4	TEC3	TEC2	TEC1	TEC0
0x05	ERRCNTH	RP	REC6	REC5	REC4	REC3	REC2	REC1	REC0
0x06	BTRL	SJW1	SJW0	BRP5	BRP4	BRP3	BRP2	BRP1	BRP0
0x07	BTRH	—	TSG2D2	TSG2D1	TSG2D0	TSG1D3	TSG1D2	TSG1D1	TSG1D0
0x08	INTRL	INTID7	INTID6	INTID5	INTID4	INTID3	INTID2	INTID1	INTID0
0x09	INTRH	INTID15	INTID14	INTID13	INTID12	INTID11	INTID10	INTID9	INTID8
0x0A	TESTRL	RX	TX1	TX0	LBACK	SILENT	BASIC	—	—
0x0C	BRPERL	—	—	—	—	BRPE3	BRPE2	BRPE1	BRPE0
0x10	IF1CREQL	—	—	MSG1N5	MSG1N4	MSG1N3	MSG1N2	MSG1N1	MSG1N0
0x11	IF1CREQH	BUSY1	—	—	—	—	—	—	—
0x12	IF1CMSKL	TD1DIR	MASK1	ARB1	CTRL1	CINTPND1	TQ1DTA	DATA1A	DATA1B
0x14	IF1MSK1L	MSK107	MSK106	MSK105	MSK104	MSK103	MSK102	MSK101	MSK100
0x15	IF1MSK1H	MSK115	MSK114	MSK113	MSK112	MSK111	MSK110	MSK109	MSK108
0x16	IF1MSK2L	MSK123	MSK122	MSK121	MSK120	MSK119	MSK118	MSK117	MSK116
0x17	IF1MSK2H	MXTD1	MDIR1	—	MSK128	MSK127	MSK126	MSK125	MSK124
0x18	IF1ARB1L	ID107	ID106	ID105	ID104	ID103	ID102	ID101	ID100
0x19	IF1ARB1H	ID115	ID114	ID113	ID112	ID111	ID110	ID109	ID108
0x1A	IF1ARB2L	ID123	ID122	ID121	ID120	ID119	ID118	ID117	ID116
0x1B	IF1ARB2H	MSG1VA	XTD1	DIR1	ID128	ID127	ID126	ID125	ID124
0x1C	IF1MCTRL	EOB1	—	—	—	DLC13	DLC12	DLC11	DLC10
0x1D	IF1MCTRH	N1DTA	MSG1LST	INT1PND	UMASK1	TX1IEN	RX1IEN	RMT1EN	T1REQ
0x1E	IF1DTA1L	D7	D6	D5	D4	D3	D2	D1	D0
0x1F	IF1DTA1H	D7	D6	D5	D4	D3	D2	D1	D0
0x20	IF1DTA2L	D7	D6	D5	D4	D3	D2	D1	D0
0x21	IF1DTA2H	D7	D6	D5	D4	D3	D2	D1	D0
0x22	IF1DTB1L	D7	D6	D5	D4	D3	D2	D1	D0

Address	Register Name	Bit							
		7	6	5	4	3	2	1	0
0x23	IF1DTB1H	D7	D6	D5	D4	D3	D2	D1	D0
0x24	IF1DTB2L	D7	D6	D5	D4	D3	D2	D1	D0
0x25	IF1DTB2H	D7	D6	D5	D4	D3	D2	D1	D0
0x26~37	Note: Reserved, cannot be changed								
0x38	CRLL	DAY7	DAY6	DAY5	DAY4	DAY3	DAY2	DAY1	DAY0
0x39	CRLH	MON7	MON6	MON5	MON4	MON3	MON2	MON1	MON0
0x3A	CRHL	SUBSTEP3	SUBSTEP2	SUBSTEP1	SUBSTEP0	YEAR3	YEAR2	YEAR1	YEAR0
0x3B	CRHH	REL3	REL2	REL1	REL0	STEP3	STEP2	STEP1	STEP0
0x40	IF2CREQL	—	—	MSG2N5	MSG2N4	MSG2N3	MSG2N2	MSG2N1	MSG2N0
0x41	IF2CREQH	BUSY2	—	—	—	—	—	—	—
0x42	IF2CMSKL	TD2DIR	MASK2	ARB2	CTRL2	CINTPND2	TQ2DTA	DATA2A	DATA2B
0x44	IF2MSK1L	MSK207	MSK206	MSK205	MSK204	MSK203	MSK202	MSK201	MSK200
0x45	IF2MSK1H	MSK215	MSK214	MSK213	MSK212	MSK211	MSK210	MSK209	MSK208
0x46	IF2MSK2L	MSK223	MSK222	MSK221	MSK220	MSK219	MSK218	MSK217	MSK216
0x47	IF2MSK2H	MXTD2	MDIR2	—	MSK228	MSK227	MSK226	MSK225	MSK224
0x48	IF2ARB1L	ID207	ID206	ID205	ID204	ID203	ID202	ID201	ID200
0x49	IF2ARB1H	ID215	ID214	ID213	ID212	ID211	ID210	ID209	ID208
0x4A	IF2ARB2L	ID223	ID222	ID221	ID220	ID219	ID218	ID217	ID216
0x4B	IF2ARB2H	MSG2VA	XTD2	DIR2	ID228	ID227	ID226	ID225	ID224
0x4C	IF2MCTRL	EOB2	—	—	—	DLC23	DLC22	DLC21	DLC20
0x4D	IF2MCTRH	N2DTA	MSG2LST	INT2PND	UMASK2	TX2IEN	RX2IEN	RMT2EN	T2REQ
0x4E	IF2DTA1L	D7	D6	D5	D4	D3	D2	D1	D0
0x4F	IF2DTA1H	D7	D6	D5	D4	D3	D2	D1	D0
0x50	IF2DTA2L	D7	D6	D5	D4	D3	D2	D1	D0
0x51	IF2DTA2H	D7	D6	D5	D4	D3	D2	D1	D0
0x52	IF2DTB1L	D7	D6	D5	D4	D3	D2	D1	D0
0x53	IF2DTB1H	D7	D6	D5	D4	D3	D2	D1	D0
0x54	IF2DTB2L	D7	D6	D5	D4	D3	D2	D1	D0
0x55	IF2DTB2H	D7	D6	D5	D4	D3	D2	D1	D0
0x80	TREQR1L	TREQ8	TREQ7	TREQ6	TREQ5	TREQ4	TREQ3	TREQ2	TREQ1
0x81	TREQR1H	TREQ16	TREQ15	TREQ14	TREQ13	TREQ12	TREQ11	TREQ10	TREQ9
0x82	TREQR2L	TREQ24	TREQ23	TREQ22	TREQ21	TREQ20	TREQ19	TREQ18	TREQ17
0x83	TREQR2H	TREQ32	TREQ31	TREQ30	TREQ29	TREQ28	TREQ27	TREQ26	TREQ25
0x90	NEWDT1L	NDTA8	NDTA7	NDTA6	NDTA5	NDTA4	NDTA3	NDTA2	NDTA1
0x91	NEWDT1H	NDTA16	NDTA15	NDTA14	NDTA13	NDTA12	NDTA11	NDTA10	NDTA9
0x92	NEWDT2L	NDTA24	NDTA23	NDTA22	NDTA21	NDTA20	NDTA19	NDTA18	NDTA17
0x93	NEWDT2H	NDTA32	NDTA31	NDTA30	NDTA29	NDTA28	NDTA27	NDTA26	NDTA25
0xA0	INTPND1L	INTPND8	INTPND7	INTPND6	INTPND5	INTPND4	INTPND3	INTPND2	INTPND1
0xA1	INTPND1H	INTPND16	INTPND15	INTPND14	INTPND13	INTPND12	INTPND11	INTPND10	INTPND9
0xA2	INTPND2L	INTPND24	INTPND23	INTPND22	INTPND21	INTPND20	INTPND19	INTPND18	INTPND17
0xA3	INTPND2H	INTPND32	INTPND31	INTPND30	INTPND29	INTPND28	INTPND27	INTPND26	INTPND25
0xB0	MSGVAL1L	MSGVA8	MSGVA7	MSGVA6	MSGVA5	MSGVA4	MSGVA3	MSGVA2	MSGVA1
0xB1	MSGVAL1H	MSGVA16	MSGVA15	MSGVA14	MSGVA13	MSGVA12	MSGVA11	MSGVA10	MSGVA9
0xB2	MSGVAL2L	MSGVA24	MSGVA23	MSGVA22	MSGVA21	MSGVA20	MSGVA19	MSGVA18	MSGVA17
0xB3	MSGVAL2H	MSGVA32	MSGVA31	MSGVA30	MSGVA29	MSGVA28	MSGVA27	MSGVA26	MSGVA25
0xBF	CANCFG	D7	CANEN	—	RMFD4	RMFD3	RMFD2	RMFD1	RMFD0
0xC0	FOCFG	D7	D6	RMXFIV	RM1FIV	CANIV	HXTEN	FODIV1	FODIV0
0xC1	SFIOSTC	—	SOFT2	SOFT1	SOFT0	—	CLKOEN	CLKHST	MISOHST

Register Reset Condition

A Reset function is a fundamental part of the CAN bus controller ensuring the CAN bus controller can be set to some predetermined condition irrespective of outside parameters.

To ensure reliable operation, it is important to know what condition the CAN bus controller registers is in after a Power on Reset or an external $\overline{\text{CAN\_RES}}$ pin reset or receive a "Reset Can Block" instruction. The following table describes how the reset affects each of the internal registers.

Register	Power On Reset/$\overline{\text{CAN\_RES}}$ Reset/"Reset CAN Block" Instruction Reset
CTRLRL	000- 0001
STATRL	0000 0000
ERRCNTL	0000 0000
ERRCNTH	0000 0000
BTRL	0000 0001
BTRH	-010 0011
INTRL	0000 0000
INTRH	0000 0000
TESTRL	x000 00--
BRPERL	---- 0000
IF1CREQL	--00 0001
IF1CREQH	0--- ----
IF1CMSKL	0000 0000
IF1MSK1L	1111 1111
IF1MSK1H	1111 1111
IF1MSK2L	1111 1111
IF1MSK2H	11-1 1111
IF1ARB1L	0000 0000
IF1ARB1H	0000 0000
IF1ARB2L	0000 0000
IF1ARB2H	0000 0000
IF1MCTRL	0--- 0000
IF1MCTRH	0000 0000
IF1DTA1L	0000 0000
IF1DTA1H	0000 0000
IF1DTA2L	0000 0000
IF1DTA2H	0000 0000
IF1DTB1L	0000 0000
IF1DTB1H	0000 0000
IF1DTB2L	0000 0000
IF1DTB2H	0000 0000
CRLL	0010 0111
CRLH	0000 0010
CRHL	0000 0101
CRHH	0010 0001
IF2CREQL	--00 0001
IF2CREQH	0--- ----
IF2CMSKL	0000 0000
IF2MSK1L	1111 1111
IF2MSK1H	1111 1111
IF2MSK2L	1111 1111

Register	Power On Reset/$\overline{\text{CAN\_RES}}$ Reset/“Reset CAN Block” Instruction Reset
IF2MSK2H	1 1 - 1 1 1 1 1
IF2ARB1L	0 0 0 0 0 0 0 0
IF2ARB1H	0 0 0 0 0 0 0 0
IF2ARB2L	0 0 0 0 0 0 0 0
IF2ARB2H	0 0 0 0 0 0 0 0
IF2MCTRL	0 - - - 0 0 0 0
IF2MCTRH	0 0 0 0 0 0 0 0
IF2DTA1L	0 0 0 0 0 0 0 0
IF2DTA1H	0 0 0 0 0 0 0 0
IF2DTA2L	0 0 0 0 0 0 0 0
IF2DTA2H	0 0 0 0 0 0 0 0
IF2DTB1L	0 0 0 0 0 0 0 0
IF2DTB1H	0 0 0 0 0 0 0 0
IF2DTB2L	0 0 0 0 0 0 0 0
IF2DTB2H	0 0 0 0 0 0 0 0
TREQR1L	0 0 0 0 0 0 0 0
TREQR1H	0 0 0 0 0 0 0 0
TREQR2L	0 0 0 0 0 0 0 0
TREQR2H	0 0 0 0 0 0 0 0
NEWDT1L	0 0 0 0 0 0 0 0
NEWDT1H	0 0 0 0 0 0 0 0
NEWDT2L	0 0 0 0 0 0 0 0
NEWDT2H	0 0 0 0 0 0 0 0
INTPND1L	0 0 0 0 0 0 0 0
INTPND1H	0 0 0 0 0 0 0 0
INTPND2L	0 0 0 0 0 0 0 0
INTPND2H	0 0 0 0 0 0 0 0
MSGVAL1L	0 0 0 0 0 0 0 0
MSGVAL1H	0 0 0 0 0 0 0 0
MSGVAL2L	0 0 0 0 0 0 0 0
MSGVAL2H	0 0 0 0 0 0 0 0
CANCFG	1 0 - 0 0 0 0 0
FOCFG	0 0 0 0 0 1 0 0
SFIOSTC	- 0 0 0 - 1 0 0

Table Legend: “-” Unimplemented
“x” Unknown

Register Description

The following is the detailed register description. The registers are used for different functions.

Programmer's Model

Register	Description	Note
CTRLRL	CAN Control Register	—
STATRL	Status Register	—
ERRCNTH/ERRCNTL	Error Counter	Read only
BTRH/BTRL	Bit Timing Register	Write enabled by CCE
INTRH/INTRL	Interrupt Register	Read only
TESTRL	Test Register	Write enabled by TEST
BRPERL	BRP Extension Register	Write enabled by CCE
IF1CREQH/IF1CREQL	IF1 Command Request	—
IF1CMSKL	IF1 Command Mask	—
IF1MSK1H/IF1MSK1L	IF1 Mask 1	—
IF1MSK2H/IF1MSK2L	IF1 Mask 2	—
IF1ARB1H/IF1ARB1L	IF1 Arbitration 1	—
IF1ARB2H/IF1ARB2L	IF1 Arbitration 2	—
IF1MCTRH/IF1MCTRL	IF1 Message Control	—
IF1DTA1H/IF1DTA1L	IF1 Data A 1	—
IF1DTA2H/IF1DTA2L	IF1 Data A 2	—
IF1DTB1H/IF1DTB1L	IF1 Data B 1	—
IF1DTB2H/IF1DTB2L	IF1 Data B 2	—
IF2CREQH/IF2CREQL	IF2 Command Request	—
IF2CMSKL	IF2 Command Mask	—
IF2MSK1H/IF2MSK1L	IF2 Mask 1	—
IF2MSK2H/IF2MSK2L	IF2 Mask 2	—
IF2ARB1H/IF2ARB1L	IF2 Arbitration 1	—
IF2ARB2H/IF2ARB2L	IF2 Arbitration 2	—
IF2MCTRH/IF2MCTRL	IF2 Message Control	—
IF2DTA1H/IF2DTA1L	IF2 Data A 1	—
IF2DTA2H/IF2DTA2L	IF2 Data A 2	—
IF2DTB1H/IF2DTB1L	IF2 Data B 1	—
IF2DTB2H/IF2DTB2L	IF2 Data B 2	—
CRLH/CRLL	Core Release Low	Read only
CRHH/CRHL	Core Release High	Read only
TREQR1H/TREQR1L	Transmission Request 1	Read only
TREQR2H/TREQR2L	Transmission Request 2	Read only
NEWDT1H/NEWDT1L	New Data 1	Read only
NEWDT2H/NEWDT2L	New Data 2	Read only
INTPND1H/INTPND1L	Interrupt Pending 1	Read only
INTPND2H/INTPND2L	Interrupt Pending 2	Read only
MSGVAL1H/MSGVAL1L	Message Valid 1	Read only
MSGVAL2H/MSGVAL2L	Message Valid 2	Read only
CANCFG	CAN Configuration	—
FOCFG	CAN Bus Controller Output Configuration	—
SFIOSTC	Output pin Configuration	—

CAN Bus Controller Output Configuration Register

• FOCFG Register (ADDRESS: 0xC0)

Bit	7	6	5	4	3	2	1	0
Name	D7	D6	RMXFIV	RM1FIV	CANIV	HXTEN	FODIV1	FODIV0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	1	0	0

Bit 7~6 **D6~D5**: Reserved bits, should remain unchanged after power-on reset

Bit 5 **RMXFIV**: RMXINT interrupt output level selection
0: Active Low
1: Active High

Bit 4 **RM1FIV**: RM1INT interrupt output level selection
0: Active Low
1: Active High

Bit 3 **CANIV**: CANMINT interrupt output level selection
0: Active Low
1: Active High

Bit 2 **HXTEN**: HXT oscillator enable control
0: Disable
1: Enable

Bit 1~0 **FODIV1~FODIV0**: CLKOUT pin prescaler control
00: f_{CLKO}=CAN_f_{SYS}
01: f_{CLKO}=CAN_$f_{SYS}/2$
10: f_{CLKO}=CAN_$f_{SYS}/4$
11: f_{CLKO}=CAN_$f_{SYS}/8$

• SFIOSTC Register (ADDRESS: 0xC1)

Bit	7	6	5	4	3	2	1	0
Name	—	SOFT2	SOFT1	SOFT0	—	CLKOEN	CLKHST	MISOHST
R/W	—	R/W	R/W	R/W	—	R/W	R/W	R/W
POR	—	0	0	0	—	1	0	0

Bit 7 Unimplemented, read as “0”

Bit 6~4 **SOFT2~SOFT0**: SOF signal width selection
SOF signal width=$2^{[(SOFT[2:0]+3)]}\times(1/f_{HXT})$
Here, SOFT[2:0]=000~111

Bit 3 Unimplemented, read as “0”

Bit 2 **CLKOEN**: CLKOUT Pin enable control
0: Disable
1: Enable

Bit 1 **CLKHST**: CLKOUT Pin output state for HXT disabled
0: Output low
1: Output high

Bit 0 **MISOHST**: MISO line output state for HXT disabled
0: Output low
1: Output high

Note: At HXT off state, the CANCFG, FOCFG and SFIOSTC registers can be correctly written but not read if using the SPI interface.

CAN Configuration Register

• **CANCFG Register (ADDRESS: 0xBF)**

Bit	7	6	5	4	3	2	1	0
Name	D7	CANEN	—	RMFD4	RMFD3	RMFD2	RMFD1	RMFD0
R/W	R/W	R/W	—	R/W	R/W	R/W	R/W	R/W
POR	1	0	—	0	0	0	0	0

Bit 7 **D7**: Reserved, must be fixed at "1"

Bit 6 **CANEN**: CAN Core Enable Control
0: Disable
1: Enable
When this bit is cleared to zero, CAN core remains in reset state and cannot be written in.

Bit 5 Unimplemented, read as "0"

Bit 4~0 **RMFD4~RMFD0**: Set receive FIFO threshold
0x00 : Message Object Number 32.
0x01~0x1F: Message Object Number 1 ~ Message Object Number 31.
These bits are used to select one of the Message Object Number 1 ~ Message Object Number 32. When the selected Message Object received a Message successfully, a interrrupt signal RMXINT will output on the RMXINT line.

CAN Protocol Related Registers

These registers are related to the CAN protocol controller in the CAN Core. They control the operating modes and the configuration of the CAN bit timing and provide status information.

1. CAN Control Registers

The contents of the control register are used to change the behavior of the CAN operation. Bits may be set or reset by the connected MCU via a SPI interface.

• **CTRLRL Register (ADDRESS: 0x00)**

Bit	7	6	5	4	3	2	1	0
Name	TEST	CCE	DAR	—	EIE	SIE	CANIE	INIT
R/W	R/W	R/W	R/W	—	R/W	R/W	R/W	R/W
POR	0	0	0	—	0	0	0	1

Bit 7 **TEST**: Test Mode Enable Control
0: Normal Operation
1: Test Mode

Bit 6 **CCE**: Configuration Change Enable
0: The CPU has no write access to protected register bits
1: While INIT='1', the CPU has write access to protected register bits

Bit 5 **DAR**: Disable Automatic Retransmission
0: Enable Automatic Retransmission of disturbed messages
1: Disable Automatic Retransmission of disturbed messages

Bit 4 Unimplemented, read as "0"

Bit 3 **EIE**: Error Interrupt Enable
0: Disabled – No Error Status Interrupt will be generated
1: Enabled – A change of bits BOFF or EWARN in the Status Register will cause the Interrupt Register to be set to Status Interrupt (INTID15~INTID0=0x8000)

Bit 2 **SIE**: Status Change Interrupt Enable
0: Disabled – No Status Change Interrupt will be generated
1: Enabled – The Interrupt Register will be set to Status Interrupt (INTID15~INTID0=0x8000) when the CAN sets LEC[2:0] to a value ≠ 7

Bit 1 **CANIE**: Module Interrupt Enable
0: Disabled - Module Interrupt can_int is always inactive
1: Enabled - When the Interrupt Register is ≠ zero, the interrupt line can_int is set to active. can_int remains active until all interrupts are processed (Interrupt Register returns to zero)

Bit 0 **INIT**: Initialization
0: Normal Operation
1: Initialization is started

The busoff recovery sequence (see CAN Specification Rev. 2.0) cannot be shortened by setting or resetting INIT. If the device goes busoff, it will set INIT of its own accord, stopping all bus activities.

Once INIT has been cleared by the CPU, the device will then wait for 129 occurrences of Bus Idle (129×11 consecutive recessive bits) before resuming normal operations. At the end of the busoff recovery sequence, the Error Management Counters will be reset.

2. Test Register

Write access to the Test Register TESTRL is enabled by setting bit TEST in the CAN Control Register. The different test functions may be combined, but TX[1:0] ≠ "00" disturbs message transfer. The TESTRL register should be cleared to zero before exiting the Test Mode.

• TESTRL Register (ADDRESS: 0x0A)

Bit	7	6	5	4	3	2	1	0
Name	RX	TX1	TX0	LBACK	SILENT	BASIC	—	—
R/W	R	R/W	R/W	R/W	R/W	R/W	—	—
POR	x	0	0	0	0	0	—	—

Bit 7 **RX**: Monitors the actual value of the CANRX Pin
0: The CAN bus is dominant (CANRX='0').
1: The CAN bus is recessive (CANRX='1').
Note: The POR value of 'x' signifies the actual POR value of the CANRX pin.

Bit 6~5 **TX1~TX0**: Control of CANTX pin
00: Reset value, CANTX is controlled by the CAN Core.
01: Sample Point can be monitored at CANTX pin
10: CANTX pin drives a dominant ('0') value.
11: CANTX pin drives a recessive ('1') value.

Bit 4 **LBACK**: Loop Back Mode Control
0: Loop Back Mode is disabled.
1: Loop Back Mode is enabled.

Bit 3 **SILENT**: Silent Mode Control
0: Normal operation.
1: The module is in Silent Mode.

Bit 2 **BASIC**: Basic Mode Control
0: Basic Mode disabled.
1: Basic Mode, IF1 Registers used as TX Buffer, IF2 Registers used as RX Buffer.

Bit 1~0 Unimplemented, read as "0"

3. Status Register

The contents of the status register reflect the status of the CAN. Some bits can only be read while some are read/write bits.

• STATRL Register (ADDRESS: 0x02)

Bit	7	6	5	4	3	2	1	0
Name	BOFF	EWARN	EPASS	RXOK	TXOK	LEC2	LEC1	LEC0
R/W	R	R	R	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7 **BOFF**: Busoff Status
0: The CAN module is not busoff
1: The CAN module is in busoff state

Bit 6 **EWARN**: Error Warning Status
0: Both error counters are below the error warning limit of 96
1: At least one of the error counters in the EML has reached the error warning limit of 96

Bit 5 **EPASS**: Error Passive
0: The CAN Core is error active
1: The CAN Core is error passive as defined in the CAN Specification

Bit 4 **RXOK**: Received a Message Successfully
0: Since this bit was last reset by the CPU, no message has been successfully received.
1: Since this bit was last reset by the CPU, a message has been successfully received (independent of the result of acceptance filtering).
This RXOK bit is never reset by the CAN Core.

Bit 3 **TXOK**: Transmitted a Message Successfully
0: Since this bit was reset by the CPU, no message has been successfully transmitted.
1: Since this bit was last reset by the CPU, a message has been successfully (error free and acknowledged by at least one other node) transmitted
This TXOK bit is never reset by the CAN Core

Bit 2~0 **LEC2~LEC0**: Last Error Code (Type of the last protocol event to occur on the CAN bus)
000: No Error
Message successfully transmitted or received.
001: Stuff Error
More than 5 equal bits in a sequence have occurred in a part of a received message where this is not allowed.
010: Form Error
Fixed format part of a received frame has the wrong format.
011: AckError
The message this CAN Core transmitted was not acknowedged by another node.
100: Bit1Error
During the transmission of a message (with the exception of the arbitration field), the device wanted to send a recessive level (bit of logical value '1'), but the monitored bus value was dominant.
101: Bit0 Error
During the transmission of a message (or acknowledge bit, or active error flag, or overload flag), the device wanted to send a dominant level (data or identifier bit logical value '0', but the monitored Bus value was recessive. During busoff recovery this status is set each time a sequence of 11 recessive bits has been monitored. This enables the CPU to monitor the proceeding of the busoff recovery sequence (indicating the bus is not stuck at dominant or continuously disturbed). During the waiting time after the resetting of INIT, each time a sequence of 11 recessive bits has been monitored, a Bit0 Error code is written to the Status Register, enabling the CPU to readily check up whether the CAN bus

is stuck at dominant or continuously disturbed and to monitor the proceeding of the busoff recovery sequence.

110: CRC Error
The CRC check sum was incorrect in the message received, the CRC received for an incoming wanted to send a recessive level (bit of logical value '1'), but the monitored bus value was dominant.

111: No Change
When the LEC bit field shows the value '7', no CAN bus event was detected since the CPU wrote this value to the LEC bit field.

The LEC field holds a code which indicates the type of the last error to occur on the CAN bus. This field will be cleared to '0' when a message has been transferred (reception or transmission) without error. The code '7' may be written by the CPU to check for updates.

Note: A Status Interrupt is generated by bits BOFF and EWARN (Error Interrupt) or by RXOK, TXOK and LEC (Status Change Interrupt) assumed that the corresponding enable bits in the CAN Control Register are set. A change of bit EPASS or a write to RXOK, TXOK or LEC will never generate a Status Interrupt.

Reading the Status Register will clear the Status Interrupt value (INTID15~INTID0=0x8000) in the Interrupt Register, if it is pending.

4. CAN Error Counter Registers

• ERRCNTL Register (ADDRESS: 0x04)

Bit	7	6	5	4	3	2	1	0
Name	TEC7	TEC6	TEC5	TEC4	TEC3	TEC2	TEC1	TEC0
R/W	R	R	R	R	R	R	R	R
POR	0	0	0	0	0	0	0	0

Bit 7~0 **TEC7~TEC0**: Transmit Error Counter
Actual state of the Transmit Error Counter.

• ERRCNTH Register (ADDRESS: 0x05)

Bit	7	6	5	4	3	2	1	0
Name	RP	REC6	REC5	REC4	REC3	REC2	REC1	REC0
R/W	R	R	R	R	R	R	R	R
POR	0	0	0	0	0	0	0	0

Bit 7 **RP**: Receive Error Passive
0: The Receive Error Counter is below the error passive level
1: The Receive Error Counter has reached the error passive level as defined in the CAN Specification

Bit 6~0 **REC6~REC0**: Receive Error Counter
Actual state of the Receive Error Counter.

5. Bit Timing Registers

The bit time is divided into four segments which are the Synchronisation Segment, the Propagation Time Segment, the Phase Buffer Segment 1 and the Phase Buffer Segment 2. Each segment consists of a specification, programmable number of time quanta. The length of the time quantum (t_q), which is the basic time unit of the bit time, is defined by the CAN controller's system clock f_{CAN} and the Baud Rate Prescaler (BRP) and the BRP Extension Register.

The time quantum is defined as:

$$t_q=(BRPE[3:0]\times 0x40+BRP[5:0]+1)/f_{CAN}$$

Where f_{CAN}=CAN module clock frequency

The contents of the BTRL register define the values of the Baud Rate Prescaler and the (Re) Synchronisation Jump Width (SJW). The BTRH register bits define the length of the time segment before and after the sample point. The registers are writable only when bits CCE and INIT in the CAN Control Register are set.

- **BTRL Register (ADDRESS: 0x06)**

Bit	7	6	5	4	3	2	1	0
Name	SJW1	SJW0	BRP5	BRP4	BRP3	BRP2	BRP1	BRP0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	1

Bit 7~6 **SJW1~SJW0**: (Re) Synchronisation Jump Width
0x00~0x03: Valid programmed values are 0~3.
The actual interpretation by the hardware of this value is such that one more than the value programmed here is used.

Bit 5~0 **BRP5~BRP0**: Baud Rate Prescaler (CAN module value)
0x01~0x3F: The value by which the system clock frequency is divided for generating the bit time quanta. The bit time is built up from a multiple of this time quanta. Valid values for BRP[5:0] are 0~63.
The actual interpretation by the hardware of this value is such that one more than the programmed value is used.

- **BTRH Register (ADDRESS: 0x07)**

Bit	7	6	5	4	3	2	1	0
Name	—	TSG2D2	TSG2D1	TSG2D0	TSG1D3	TSG1D2	TSG1D1	TSG1D0
R/W	—	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	—	0	1	0	0	0	1	1

Bit 7 Unimplemented, read as “0”

Bit 6~4 **TSG2D2~TSG2D0**: The time segment after the sample point
0x00~0x07: Valid values for TSG2D[2:0] are 0~7. The actual interpretation by the hardware of this value is such that one more than the value programmed here is used.

Bit 3~0 **TSG1D3~TSG1D0**: The time segment before the sample point
0x01~0x0F: Valid values for TSG1D[3:0] are 1~15. The actual interpretation by the hardware of this value is such that one more than the value programmed here is used.

Note: If f_{CAN}=8MHz, the reset value of BTRL=0x01 and BTRH=0x23 configures the CAN for a bit rate of 500 kBit/s.

- **BRPERL Register (ADDRESS: 0x0C)**

This BRPERL register configures the BRP extension for Classic CAN operation. The register is writable by setting CCE bit.

Bit	7	6	5	4	3	2	1	0
Name	—	—	—	—	BRPE3	BRPE2	BRPE1	BRPE0
R/W	—	—	—	—	R/W	R/W	R/W	R/W
POR	—	—	—	—	0	0	0	0

Bit 7~4 Unimplemented, read as "0"

Bit 3~0 **BRPE3~BRPE0**: Baud Rate Prescaler Extension
0x00~0x0F: By programming BRPE[3:0] the Baud Rate Prescaler can be extended to values up to 1023.
The actual interpretation by the hardware is that one more than the value programmed by BRPE[3:0] (MSBs) and BRP[5:0] (LSBs) is used.

Message Interface Registers

There are two sets of Interface Registers which are used to control the CPU access to the Message RAM. The Interface Registers avoid conflicts between CPU access to the Message RAM and CAN message reception and transmission by buffering the data to be transferred. A complete Message Object or parts of the Message Object may be transferred between the Message RAM and the IFn Message Buffer registers in one single transfer.

The function of the two interface register sets is identical (except for test mode Basic). They can be used the way that one set of registers is used for data transfer to the Message RAM while the other set of registers is used for the data transfer from the Message RAM, allowing both processes to be interrupted by each other.

Each set of Interface Registers consists of Message Buffer Registers controlled by their own Command Registers. The Command Mask Register specifies the direction of the data transfer and which parts of a Message Object will be transferred. The Command Request Register is used to select a Message Object in the Message RAM as target or source for the transfer and to start the action specified in the Command Mask Register.

IF1 Register Set	Description	IF2 Register Set	Description
IF1CREQH/IF1CREQL	IF1 Command Request	IF2CREQH/IF2CREQL	IF2 Command Request
IF1CMSKL	IF1 Command Mask	IF2CMSKL	IF2 Command Mask
IF1MSK1H/IF1MSK1L	IF1 Mask 1	IF2MSK1H/IF2MSK1L	IF2 Mask 1
IF1MSK2H/IF1MSK2L	IF1 Mask 2	IF2MSK2H/IF2MSK2L	IF2 Mask 2
IF1ARB1H/IF1ARB1L	IF1 Arbitration 1	IF2ARB1H/IF2ARB1L	IF2 Arbitration 1
IF1ARB2H/IF1ARB2L	IF1 Arbitration 2	IF2ARB2H/IF2ARB2L	IF2 Arbitration 2
IF1MCTRH/IF1MCTRL	IF1 Message Control	IF2MCTRH/IF2MCTRL	IF2 Message Control
IF1DTA1H/IF1DTA1L	IF1 Data A1	IF2DTA1H/IF2DTA1L	IF2 Data A1
IF1DTA2H/IF1DTA2L	IF1 Data A2	IF2DTA2H/IF2DTA2L	IF2 Data A2
IF1DTB1H/IF1DTB1L	IF1 Data B1	IF2DTB1H/IF2DTB1L	IF2 Data B1
IF1DTB2H/IF1DTB2L	IF1 Data B2	IF2DTB2H/IF2DTB2L	IF2 Data B2

IF1 and IF2 Message Interface Register Sets

1. IFn Command Request Registers

A message transfer is started as soon as the CPU has written the message number to the Command Request Register. With this write operation the BUSYn bit is automatically set to '1' and output can_wait_b is activated to notify the CPU that a transfer is in progress. After a wait time of 3 to 6 can_clk periods, the transfer between the Interface Register and the Message RAM has completed. The BUSYn bit is set back to zero and can_wait_b is deactivated (see Module Integration Guide).

• IFnCREQL Register

Bit	7	6	5	4	3	2	1	0
Name	—	—	MSGnN5	MSGnN4	MSGnN3	MSGnN2	MSGnN1	MSGnN0
R/W	—	—	R/W	R/W	R/W	R/W	R/W	R/W
POR	—	—	0	0	0	0	0	1

Bit 7~6 Unimplemented, read as "0"

Bit 5~0 **MSGnN5~MSGnN0**: Message Number
0x00: Not a valid Message Number, interpreted as 0x20
0x01~0x20: Valid Message Number, the Message Object in the Message RAM is selected for data transfer.
0x21~0x3F: Not a valid Message Number, interpreted as 0x01-0x1F

Note: When a Message Number that is not valid is written into the Command Request Register, the Message Number will be transformed into a valid value and that Message Object will be transferred.

• **IFnCREQH Register**

Bit	7	6	5	4	3	2	1	0
Name	BUSYn	—	—	—	—	—	—	—
R/W	R/W	—	—	—	—	—	—	—
POR	0	—	—	—	—	—	—	—

Bit 7 **BUSYn**: Busy Flag
0: Reset to zero when read/write action has finished.
1: Set to one when writing to the IFn Command Request Register.

Bit 6~0 Unimplemented, read as "0"

2. IFn Command Mask Registers

The control bits of the IFn Command Mask Register specify the transfer direction and select which of the IFn Message Buffer Registers as source or target of the data transfer.

• **IFnCMSKL Register**

Bit	7	6	5	4	3	2	1	0
Name	TDnDIR	MASKn	ARBn	CTRLn	CINTPNDn	TQnDTA	DATAnA	DATAnB
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7 **TDnDIR**: Write/Read Selection
0: Read - Transfer data from the Message Object addressed by the Command Request Register into the selected Message Buffer Registers.
1: Write - Transfer data from the selected Message Buffer Registers to the Message Object addressed by the Command Request Register.
The other bits of IFn Command MASKn Register have different functions depending on the transfer direction:

Direction=Write (TDnDIR=1)

Bit 6 **MASKn**: Access MASK Bits
0: MASK bits unchanged.
1: Transfer Identifier MASKn + MDIRn + MXTDn to Message Object.

Bit 5 **ARBn**: Access Arbitration Bits
0: Arbitration bits unchanged.
1: Transfer Identifier + DIRn + XTDn + MSGnVA to Message Object.

Bit 4 **CTRLn**: Access Control Bits
0: Control Bits unchanged.
1: Transfer Control Bits to Message Object.

Bit 3 **CINTPNDn**: Clear Interrupt Pending Bit
0: INTPND bit remains unchanged.
1: Clear INTPND bit in the Message Object.
Note: When writing to a Message Object, this bit is ignored.

Bit 2 **TQnDTA**: Access Transmission Request Bit
0: TREQ bit unchanged
1: Set TREQ bit
If a transmission is requested by programming bit TQnDTA in the IFn Command Mask Register, bit TnREQ in the IFn Message Control Register will be ignored.

Bit 1 **DATAnA**: Access Data Bytes 0-3
0: Data Bytes 0-3 unchanged.
1: Transfer Data Bytes 0-3 to Message Object.

Bit 0 **DATAnB**: Access Data Bytes 4-7
0: Data Bytes 4-7 unchanged
1: Transfer Data Bytes 4-7 to Message Object.

Direction=Read (TDnDIR=0)

Bit 6 **MASKn**: Access MASK Bits
0: MASK bits unchanged.
1: Transfer Identifier MASKn + MDIRn + MXTDn to IFn Message Buffer Register.

Bit 5 **ARBn**: Access Arbitration Bits
0: Arbitration bits unchanged.
1: transfer Identifier + DIRn + XTDn + MSGnVA to IFn Message Buffer Register.

Bit 4 **CTRLn**: Access Control Bits
0: Control Bits unchanged.
1: Transfer Control Bits to IFn Message Buffer Register.

Bit 3 **CINTPNDn**: Clear Interrupt Pending Bit
0: INTPND bit remains unchanged.
1: Clear INTPND bit in the Message Object.

Bit 2 **TQnDTA**: Access New Data Bit
0: NDTA bit remains unchanged.
1: Clear NDTA bit in the Message Object.
Note: A read access to a Message Object can be combined with the reset of the control bits INTnPND and NnDTA.The values of these bits transferred to the IFn Message Control Register always reflect the status before resetting these bits.

Bit 1 **DATAnA**: Access Data Bytes 0-3
0: Data Bytes 0-3 unchanged.
1: Transfer Data Bytes 0-3 to IFn Message Buffer Register.

Bit 0 **DATAnB**: Access Data Bytes 4-7
0: Data Bytes 4-7 unchanged.
1: Transfer Data Bytes 4-7 to IFn Message Buffer Register.

3. IFn Message Buffer Registers

The bits of the Message Buffer registers mirror the Message Objects in the Message RAM.

• IFnMSK1L Register

Bit	7	6	5	4	3	2	1	0
Name	MSKn07	MSKn06	MSKn05	MSKn04	MSKn03	MSKn02	MSKn01	MSKn00
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	1	1	1	1	1	1	1	1

Bit 7~0 **MSKn07~MSKn00**: Identifier MASK
0: The corresponding bit in the identifier of the message object cannot inhibit the match in the acceptance filtering.
1: The corresponding identifier bit is used for acceptance filtering.

• IFnMSK1H Register

Bit	7	6	5	4	3	2	1	0
Name	MSKn15	MSKn14	MSKn13	MSKn12	MSKn11	MSKn10	MSKn09	MSKn08
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	1	1	1	1	1	1	1	1

Bit 7~0 **MSKn15~MSKn08**: Identifier MASK
0: The corresponding bit in the identifier of the message object cannot inhibit the match in the acceptance filtering.
1: The corresponding identifier bit is used for acceptance filtering.

• **IFnMSK2L Register**

Bit	7	6	5	4	3	2	1	0
Name	MSKn23	MSKn22	MSKn21	MSKn20	MSKn19	MSKn18	MSKn17	MSKn16
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	1	1	1	1	1	1	1	1

Bit 7~0 **MSKn23~MSKn16**: Identifier MASK
0: The corresponding bit in the identifier of the message object cannot inhibit the match in the acceptance filtering.
1: The corresponding identifier bit is used for acceptance filtering.

• **IFnMSK2H Register**

Bit	7	6	5	4	3	2	1	0
Name	MXTDn	MDIRn	—	MSKn28	MSKn27	MSKn26	MSKn25	MSKn24
R/W	R/W	R/W	—	R/W	R/W	R/W	R/W	R/W
POR	1	1	—	1	1	1	1	1

Bit 7 **MXTDn**: MASK Extended Identifier
0: The extended identifier bit (IDE) has no effect on the acceptance filtering.
1: The extended identifier bit (IDE) is used for acceptance filtering.

Bit 6 **MDIRn**: MASK Message Direction
0: The message direction bit (DIR) has no effect on the acceptance filtering.
1: The message direction bit (DIR) is used for acceptance filtering.

Bit 5 Unimplemented, read as “1”

Bit 4~0 **MSKn28~MSKn24**: Identifier MASK
0: The corresponding bit in the identifier of the message object cannot inhibit the match in the acceptance filtering.
1: The corresponding identifier bit is used for acceptance filtering.

• **IFnARB1L Register**

Bit	7	6	5	4	3	2	1	0
Name	IDn07	IDn06	IDn05	IDn04	IDn03	IDn02	IDn01	IDn00
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7~0 **IDn07~IDn00**: Message Identifier 7~0

• **IFnARB1H Register**

Bit	7	6	5	4	3	2	1	0
Name	IDn15	IDn14	IDn13	IDn12	IDn11	IDn10	IDn09	IDn08
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7~0 **IDn15~IDn8**: Message Identifier 15~8

• **IFnARB2L Register**

Bit	7	6	5	4	3	2	1	0
Name	IDn23	IDn22	IDn21	IDn20	IDn19	IDn18	IDn17	IDn16
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7~0 **IDn23~IDn16**: Message Identifier 23~16

• **IFnARB2H Register**

Bit	7	6	5	4	3	2	1	0
Name	MSGnVA	XTDn	DIRn	IDn28	IDn27	IDn26	IDn25	IDn24
R/W	RW	RW	RW	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7 **MSGnVA**: Message Valid Bits (of all Message Objects)
0: This Message Object is ignored by the Message Handler
1: This Message Object is configured and should be considered by the Message Handler

Bit 6 **XTDn**: Extended Identifier
0: The 11-bit ("standard") Identifier will be used for this Message Object.
1: The 29-bit ("extended") Identifier will be used for this Message Object.

Bit 5 **DIRn**: Message Direction
0: Direction=receive. On TREQ, a Remote Frame with the identifier of this Message Object is transmitted. On reception of a Data Frame with matching identifier, that message is stored in this Message Object.
1: Direction=transmit. On TREQ, the respective Message Object is transmitted as a Data Frame. On reception of a Remote Frame with matching identifier, the TREQ bit of this Message Object is set (if RMTnEN=1).

Bit 4~0 **IDn28~IDn24**: Message Identifier 28~24

4. IFn Message Control Registers

• **IFnMCTRL Register**

Bit	7	6	5	4	3	2	1	0
Name	EOBn	—	—	—	DLCn3	DLCn2	DLCn1	DLCn0
R/W	R/W	—	—	—	R/W	R/W	R/W	R/W
POR	0	—	—	—	0	0	0	0

Bit 7 **EOBn**: End of Buffer
0: Message Object belongs to a FIFO Buffer and is not the last Message Object of that FIFO Buffer.
1: Single Message Object or last Message Object of a FIFO Buffer.
This bit is used to concatenate two or more Message Objects (up to 32) to build a FIFO Buffer. For single Message Objects (not belonging to a FIFO Buffer), this bit must always be set to one.

Bit 6~4 Unimplemented, read as "0"

Bit 3~0 **DLCn3~DLCn0**: Data Length Code
0~8: CAN: Frame has 0 ~ 8 data bytes
9~15: CAN: Frame has 8 data bytes
The Data Length Code of a Message Object must be defined the same as in all the corresponding objects with the same identifier at other nodes.
When the Message Handler stores a data frame, it will write the DLCn to the value given by the received message.

• **IFnMCTRH Register**

Bit	7	6	5	4	3	2	1	0
Name	NnDTA	MSGnLST	INTnPND	UMASKn	TXnIEN	RXnIEN	RMTnEN	TnREQ
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7 **NnDTA**: New Data Bits
0: No new data has been written into the data portion of this Message Object by the Message Handler since last time this flag was cleared by the CPU.

1: The Message Handler or the CPU has written new data into the data portion of this Message Object

Bit 6 **MSGnLST**: Message Lost (only valid for Message Objects with direction=receive)
0: No message lost since last time this bit was reset by the CPU.
1: The Message Handler stored a new message into this object when NDTA was still set, the CPU has lost a message.

Bit 5 **INTnPND**: Interrupt Pending Bits
0: This message object is not the source of an interrupt.
1: This message object is the source of an interrupt. The Interrupt Identifier in the Interrupt Register will point to this message object if there is no other interrupt source with higher priority.

Bit 4 **UMASKn**: Use Acceptance Mask
0: MASK ignored.
1: Use MASK (MSKn[28:00], MXTDn, and MDIRn) for acceptance filtering.

Bit 3 **TXnIEN**: Transmit Interrupt Enable
0: INTPND will be left unchanged after the successful transmission of a frame.
1: INTPND will be set after a successful transmission of a frame.

Bit 2 **RXnIEN**: Receive Interrupt Enable
0: INTPND will be left unchanged after a successful reception of a frame.
1: INTPND will be set after a successful reception of a frame.

Bit 1 **RMTnEN**: Remote Enable
0: At the reception of a Remote Frame, TREQ is left unchanged.
1: At the reception of a Remote Frame, TREQ is set.

Bit 0 **TnREQ**: Transmission Request Bits
0: This Message Object is not waiting for transmission.
1: The transmission of this Message Object is requested and is not yet done.

5. IFn Data A and Data B Registers

In a CAN Data Frame, Data0 is the first, Data7 is the last byte to be transmitted or received. In CAN's serial bit stream, the MSB of each byte will be transmitted first.

DATA0: 1st data byte of a CAN Data Frame

DATA1: 2nd data byte of a CAN Data Frame

DATA2: 3rd data byte of a CAN Data Frame

DATA3: 4th data byte of a CAN Data Frame

DATA4: 5th data byte of a CAN Data Frame

DATA5: 6th data byte of a CAN Data Frame

DATA6: 7th data byte of a CAN Data Frame

DATA7: 8th data byte of a CAN Data Frame

The data bytes of CAN messages are stored in the IFn Message Buffer Registers in the following order:

• IFnDTA1L Register

Bit	7	6	5	4	3	2	1	0
Name	D7	D6	D5	D4	D3	D2	D1	D0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7~0 **D7~D0**: DATA0, 1st data byte of a CAN Data Frame

- **IFnDTA1H Register**

Bit	7	6	5	4	3	2	1	0
Name	D7	D6	D5	D4	D3	D2	D1	D0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7~0 **D7~D0**: DATA1, 2nd data byte of a CAN Data Frame

- **IFnDTA2L Register**

Bit	7	6	5	4	3	2	1	0
Name	D7	D6	D5	D4	D3	D2	D1	D0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7~0 **D7~D0**: DATA2, 3rd data byte of a CAN Data Frame

- **IFnDTA2H Register**

Bit	7	6	5	4	3	2	1	0
Name	D7	D6	D5	D4	D3	D2	D1	D0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7~0 **D7~D0**: DATA3, 4th data byte of a CAN Data Frame

- **IFnDTB1L Register**

Bit	7	6	5	4	3	2	1	0
Name	D7	D6	D5	D4	D3	D2	D1	D0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7~0 **D7~D0**: DATA4, 5th data byte of a CAN Data Frame

- **IFnDTB1H Register**

Bit	7	6	5	4	3	2	1	0
Name	D7	D6	D5	D4	D3	D2	D1	D0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7~0 **D7~D0**: DATA5, 6th data byte of a CAN Data Frame

- **IFnDTB2L Register**

Bit	7	6	5	4	3	2	1	0
Name	D7	D6	D5	D4	D3	D2	D1	D0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7~0 **D7~D0**: DATA6, 7th data byte of a CAN Data Frame

- **IFnDTB2H Register**

Bit	7	6	5	4	3	2	1	0
Name	D7	D6	D5	D4	D3	D2	D1	D0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7~0 **D7~D0**: DATA7, 8th data byte of a CAN Data Frame

Note: Byte DATA0 is the first data byte shifted into the shift register of the CAN Core during a reception, byte DATA7 is the last. When the Message Handler stores a Data Frame, it will write all the eight data bytes into a Message Object. If the Data Length Code is less than 8, the remaining bytes of the Message Object will be overwritten by non specified values.

Message Handler Registers

All Message Handler registers are read-only. Their contents, which include the TREQ, NDTA, INTPND, and MSGVA bits of each Message Object and the Interrupt Identifier) is status information provided by the Message Handler FSM (Finite State Machine).

1. Interrupt Registers

The interrupt registers allow the identification of an interrupt source. When an interrupt occurs, a CAN interrupt will be indicated to the CPU and an active interrupt level will output on the CANMINT line to inform users the CAN interrupt.

The interrupt register appears to the CPU as a read only memory.

• INTRL Register (ADDRESS: 0x08)

Bit	7	6	5	4	3	2	1	0
Name	INTID7	INTID6	INTID5	INTID4	INTID3	INTID2	INTID1	INTID0
R/W	R	R	R	R	R	R	R	R
POR	0	0	0	0	0	0	0	0

Bit 7~0 **INTID7~INTID0**: Interrupt Identifier

• INTRH Register (ADDRESS: 0x09)

Bit	7	6	5	4	3	2	1	0
Name	INTID15	INTID14	INTID13	INTID12	INTID11	INTID10	INTID9	INTID8
R/W	R	R	R	R	R	R	R	R
POR	0	0	0	0	0	0	0	0

Bit 7~0 **INTID15~INTID8**: Interrupt Identifier
The interrupt identifier INTID[15:0] indicates the source of the interrupt.

INTID[15:0] Value	Indicated Interrupt
0000H	No interrupt is pending
0001H~0020H	Number of Message Object which caused the interrupt
0021H~7FFFH	Unused
8000H	Status Interrupt
8001H~FFFFH	Unused

If several interrupts are pending, the CAN Interrupt Register will point to the pending interrupt with the highest priority disregarding their chronological order. An interrupt remains pending until the CPU has cleared it. If INTID15~INTID0 is different from 0x0000 and CANIE is set, the CANMINT line is driven an active level by the CAN bus controller and will remain the active level until INTID15~INTID0 is back to value 0x0000 (the cause of the interrupt is reset) or until CANIE is reset. The active CAN interrupt level is determined by the CANIV bit in the FOCFG register.

The Status Interrupt has the highest priority. Among the message interrupts, the Message Object's interrupt priority decreases with increasing message number.

A message interrupt is cleared by clearing the Message Object's INTPND bit. The Status Interrupt is cleared by reading the Status Register resp.

The Status Interrupt value 0x8000 (INTID15~INTID0) indicates that an interrupt is pending because the CAN Core has updated (not necessarily changed) the Status Register (Error Interrupt or Status Interrupt). This interrupt has the highest priority. The CPU can update (reset) the Status Register bits

RXOK, TXOK and LEC by writing to the Status Register. A write access by the CPU to the Status Registers can never generate or reset an interrupt.

All other values indicate that the source of the interrupt is one of the Message Objects, INTID points to the pending message interrupt with the highest interrupt priority.

The CPU controls whether a change of the Status Registers may cause the Interrupt Register to be set to INTID Status interrupt and whether the interrupt line becomes active when the Interrupt Register is different from zero (bit CANIE in the CAN Control Register). The Interrupt Register will be updated even when CANIE is not set.

The CPU has two possibilities to follow the source of a message interrupt. First it can follow the INTID in the Interrupt Register and second it can poll the Interrupt Pending Register.

An interrupt service routine reading the message that is the source of the interrupt may read the message and reset the Message Object's INTPND at the same time (bit CINTPNDn in the IFn Command Mask Register). When INTPND is cleared, the Interrupt Register will point to the next Message Object with a pending interrupt.

2. Transmission Request Registers

These registers hold the TREQ[32:0] bits of the 32 Message Objects. By reading out the TREQ[32:0] bits, the CPU can check for which Message Object a Transmission Request is pending. The TREQ bit of a specific Message Object can be set or reset by the CPU via the IFn Message Interface Registers or by the Message Handler after reception of a Remote Frame or after a successful transmission.

• TREQR1L Register (ADDRESS: 0x80)

Bit	7	6	5	4	3	2	1	0
Name	TREQ8	TREQ7	TREQ6	TREQ5	TREQ4	TREQ3	TREQ2	TREQ1
R/W	R	R	R	R	R	R	R	R
POR	0	0	0	0	0	0	0	0

Bit 7~0 **TREQ8~TREQ1**: Transmission Request Bits of Message Object 8~Message Object 1
0: This Message Object is not waiting for transmission.
1: The transmission of this Message Object is requested and is not yet done.

• TREQR1H Register (ADDRESS: 0x81)

Bit	7	6	5	4	3	2	1	0
Name	TREQ16	TREQ15	TREQ14	TREQ13	TREQ12	TREQ11	TREQ10	TREQ9
R/W	R	R	R	R	R	R	R	R
POR	0	0	0	0	0	0	0	0

Bit 7~0 **TREQ16~TREQ9**: Transmission Request Bits of Message Object 16 ~ Message Object 9
0: This Message Object is not waiting for transmission.
1: The transmission of this Message Object is requested and is not yet done.

• TREQR2L Register (ADDRESS: 0x82)

Bit	7	6	5	4	3	2	1	0
Name	TREQ24	TREQ23	TREQ22	TREQ21	TREQ20	TREQ19	TREQ18	TREQ17
R/W	R	R	R	R	R	R	R	R
POR	0	0	0	0	0	0	0	0

Bit 7~0 **TREQ24~TREQ17**: Transmission Request Bits of Message Object 24 ~ Message Object 17
0: This Message Object is not waiting for transmission.
1: The transmission of this Message Object is requested and is not yet done.

- **TREQR2H Register (ADDRESS: 0x83)**

Bit	7	6	5	4	3	2	1	0
Name	TREQ32	TREQ31	TREQ30	TREQ29	TREQ28	TREQ27	TREQ26	TREQ25
R/W	R	R	R	R	R	R	R	R
POR	0	0	0	0	0	0	0	0

Bit 7~0 **TREQ32~TREQ25**: Transmission Request Bits of Message Object 32 ~ Message Object 25
0: This Message Object is not waiting for transmission.
1: The transmission of this Message Object is requested and is not yet done.

3. New Data Registers

These registers hold the NDTA bits of the 32 Message Objects. By reading out the NDTA bits, the CPU can check for which Message Object the data portion was updated. The NDTA bit of a specific Message Object can be set/reset by the CPU via the IFn Message Interface Registers or by the Message Handler after reception of a Data Frame or after a successful transmission.

- **NEWDT1L Register**

Bit	7	6	5	4	3	2	1	0
Name	NDTA8	NDTA7	NDTA6	NDTA5	NDTA4	NDTA3	NDTA2	NDTA1
R/W	R	R	R	R	R	R	R	R
POR	0	0	0	0	0	0	0	0

Bit 7~0 **NDTA8~NDTA1**: New Data Bits of Message Object 8 ~ Message Object 1.
0: No new data has been written into the data portion of this Message Object by the Message Handler since last time this flag was cleared by the CPU.
1: The Message Handler or the CPU has written new data into the data portion of this Message Object.

- **NEWDT1H Register**

Bit	7	6	5	4	3	2	1	0
Name	NDTA16	NDTA15	NDTA14	NDTA13	NDTA12	NDTA11	NDTA10	NDTA9
R/W	R	R	R	R	R	R	R	R
POR	0	0	0	0	0	0	0	0

Bit 7~0 **NDTA16~NDTA9**: New Data Bits of Message Object 16 ~ Message Object 9.
0: No new data has been written into the data portion of this Message Object by the Message Handler since last time this flag was cleared by the CPU.
1: The Message Handler or the CPU has written new data into the data portion of this Message Object.

- **NEWDT2L Register**

Bit	7	6	5	4	3	2	1	0
Name	NDTA24	NDTA23	NDTA22	NDTA21	NDTA20	NDTA19	NDTA18	NDTA17
R/W	R	R	R	R	R	R	R	R
POR	0	0	0	0	0	0	0	0

Bit 7~0 **NDTA24~NDTA17**: New Data Bits of Message Object 24 ~ Message Object 17
0: No new data has been written into the data portion of this Message Object by the Message Handler since last time this flag was cleared by the CPU.
1: The Message Handler or the CPU has written new data into the data portion of this Message Object.

• **NEWDT2H Register**

Bit	7	6	5	4	3	2	1	0
Name	NDTA32	NDTA31	NDTA30	NDTA29	NDTA28	NDTA27	NDTA26	NDTA25
R/W	R	R	R	R	R	R	R	R
POR	0	0	0	0	0	0	0	0

Bit 7~0 **NDTA32~NDTA25**: New Data Bits of Message Object 32 ~ Message Object 25.
0: No new data has been written into the data portion of this Message Object by the Message Handler since last time this flag was cleared by the CPU.
1: The Message Handler or the CPU has written new data into the data portion of this Message Object.

4. Interrupt Pending Registers

These registers hold the INTPND bits of the 32 Message Objects. By reading out the INTPND bits, the CPU can check for which Message Object an interrupt is pending. The INTPND bit of a specific Message Object can be set/reset by the CPU via the IFn Message Interface Registers or by the Message Handler after reception or after a successful transmission of a frame. This will also affect the value of INTID in the Interrupt Register.

• **INTPND1L Register (ADDRESS: 0xA0)**

Bit	7	6	5	4	3	2	1	0
Name	INTPND8	INTPND7	INTPND6	INTPND5	INTPND4	INTPND3	INTPND2	INTPND1
R/W	R	R	R	R	R	R	R	R
POR	0	0	0	0	0	0	0	0

Bit 7~0 **INTPND8~INTPND1**: Interrupt Pending Bits of Message Object 8 ~ Message Object 1
0: This message object is not the source of an interrupt.
1: This message object is the source of an interrupt.
If the message object whose interrupt pending bit is set high is the source of an interrupt, the Interrupt Identifier in the Interrupt Register will point to this message object if there is no other interrupt source with higher priority.

• **INTPND1H Register (ADDRESS: 0xA1)**

Bit	7	6	5	4	3	2	1	0
Name	INTPND16	INTPND15	INTPND14	INTPND13	INTPND12	INTPND11	INTPND10	INTPND9
R/W	R	R	R	R	R	R	R	R
POR	0	0	0	0	0	0	0	0

Bit 7~0 **INTPND16~INTPND9**: Interrupt Pending Bits of Message Object 16 ~ Message Object 9
0: This message object is not the source of an interrupt.
1: This message object is the source of an interrupt.
If the message object whose interrupt pending bit is set high is the source of an interrupt, the Interrupt Identifier in the Interrupt Register will point to this message object if there is no other interrupt source with higher priority.

• **INTPND2L Register (ADDRESS: 0xA2)**

Bit	7	6	5	4	3	2	1	0
Name	INTPND24	INTPND23	INTPND22	INTPND21	INTPND20	INTPND19	INTPND18	INTPND17
R/W	R	R	R	R	R	R	R	R
POR	0	0	0	0	0	0	0	0

Bit 7~0 **INTPND24~INTPND17**: Interrupt Pending Bits of Message Object 24 ~ Message Object 17
0: This message object is not the source of an interrupt.
1: This message object is the source of an interrupt.
If the message object whose interrupt pending bit is set high is the source of an interrupt, the Interrupt Identifier in the Interrupt Register will point to this message object if there is no other interrupt source with higher priority.

• **INTPND2H Register (ADDRESS: 0xA3)**

Bit	7	6	5	4	3	2	1	0
Name	INTPND32	INTPND31	INTPND30	INTPND29	INTPND28	INTPND27	INTPND26	INTPND25
R/W	R	R	R	R	R	R	R	R
POR	0	0	0	0	0	0	0	0

Bit 7~0 **INTPND32~INTPND25**: Interrupt Pending Bits of Message Object 32 ~ Message Object 25
0: This message object is not the source of an interrupt.
1: This message object is the source of an interrupt.
If the message object whose interrupt pending bit is set high is the source of an interrupt, the Interrupt Identifier in the Interrupt Register will point to this message object if there is no other interrupt source with higher priority.

5. Message Valid Registers

These registers hold the MSGVA bits of the 32 Message Objects. By reading out the MSGVA bits, the CPU can check which Message Object is valid. The MSGVA bit of a specific Message Object can be set/reset by the CPU via the IFn Message Interface Registers.

• **MSGVAL1L Register (ADDRESS: 0xB0)**

Bit	7	6	5	4	3	2	1	0
Name	MSGVA8	MSGVA7	MSGVA6	MSGVA5	MSGVA4	MSGVA3	MSGVA2	MSGVA1
R/W	R	R	R	R	R	R	R	R
POR	0	0	0	0	0	0	0	0

Bit 7~0 **MSGVA8~MSGVA1**: Message Valid Bits of Message Object 8 ~ Message Object 1
0: This Message Object is ignored by the Message Handler.
1: This Message Object is configured and should be considered by the Message Handler.

• **MSGVAL1H Register (ADDRESS: 0xB1)**

Bit	7	6	5	4	3	2	1	0
Name	MSGVA16	MSGVA15	MSGVA14	MSGVA13	MSGVA12	MSGVA11	MSGVA10	MSGVA9
R/W	R	R	R	R	R	R	R	R
POR	0	0	0	0	0	0	0	0

Bit 7~0 **MSGVA16~MSGVA9**: Message Valid Bits of Message Object 16 ~ Message Object 9
0: This Message Object is ignored by the Message Handler.
1: This Message Object is configured and should be considered by the Message Handler.

• MSGVAL2L Register (ADDRESS: 0xB2)

Bit	7	6	5	4	3	2	1	0
Name	MSGVA24	MSGVA23	MSGVA22	MSGVA21	MSGVA20	MSGVA19	MSGVA18	MSGVA17
R/W	R	R	R	R	R	R	R	R
POR	0	0	0	0	0	0	0	0

Bit 7~0 **MSGVA24~MSGVA17**: Message Valid Bits of Message Object 24 ~ Message Object 17
0: This Message Object is ignored by the Message Handler.
1: This Message Object is configured and should be considered by the Message Handler.

• MSGVAL2H Register (ADDRESS: 0xB3)

Bit	7	6	5	4	3	2	1	0
Name	MSGVA32	MSGVA31	MSGVA30	MSGVA29	MSGVA28	MSGVA27	MSGVA26	MSGVA25
R/W	R	R	R	R	R	R	R	R
POR	0	0	0	0	0	0	0	0

Bit 7~0 **MSGVA32~MSGVA25**: Message Valid Bits of Message Object 32 ~ Message Object 25
0: This Message Object is ignored by the Message Handler.
1: This Message Object is configured and should be considered by the Message Handler.
Note: The CPU must reset the MSGVA bit of all unused Messages Objects during the initialization before it resets bit INIT in the CAN Control Register. This bit must also be reset before the identifier IDn[28:00], the control bits XTDn, DIRn, or the Data Length Code DLCn[3:0] are modified, or if the Messages Object is no longer required.

Core Release Registers

The design step of a CAN implementation can be identified by reading the Core Release Registers Low/High.

Release	Step	SubStep	Year	Month	Day	Name
2	1	0	5	02	27	Revision 2.1.0, Date 2015/02/27

1. Example for Coding of Revisions

• CRLL Register

Bit	7	6	5	4	3	2	1	0
Name	DAY7	DAY6	DAY5	DAY4	DAY3	DAY2	DAY1	DAY0
R/W	R	R	R	R	R	R	R	R
POR	0	0	1	0	0	1	1	1

Bit 7~0 **DAY7~DAY0**: Time Stamp Day
Two digits, BCD-coded. Configured by constant on CAN synthesis.

• CRLH Register

Bit	7	6	5	4	3	2	1	0
Name	MON7	MON6	MON5	MON4	MON3	MON2	MON1	MON0
R/W	R	R	R	R	R	R	R	R
POR	0	0	0	0	0	0	1	0

Bit 7~0 **MON7~MON0**: Time Stamp Month
Two digits, BCD-coded. Configured by constant on CAN synthesis.

• **CRHL Register**

Bit	7	6	5	4	3	2	1	0
Name	SUBSTEP3	SUBSTEP2	SUBSTEP1	SUBSTEP0	YEAR3	YEAR2	YEAR1	YEAR0
R/W	R	R	R	R	R	R	R	R
POR	0	0	0	0	0	1	0	1

Bit 7~4 **SUBSTEP3~SUBSTEP0**: Sub-step of Core Release
One digit, BCD-coded.

Bit 3~0 **YEAR3~YEAR0**: Time Stamp Year (2010 + digit)
One digit, BCD-coded. Configured by constant on CAN synthesis.

• **CRHH Register**

Bit	7	6	5	4	3	2	1	0
Name	REL3	REL2	REL1	REL0	STEP3	STEP2	STEP1	STEP0
R/W	R	R	R	R	R	R	R	R
POR	0	0	1	0	0	0	0	1

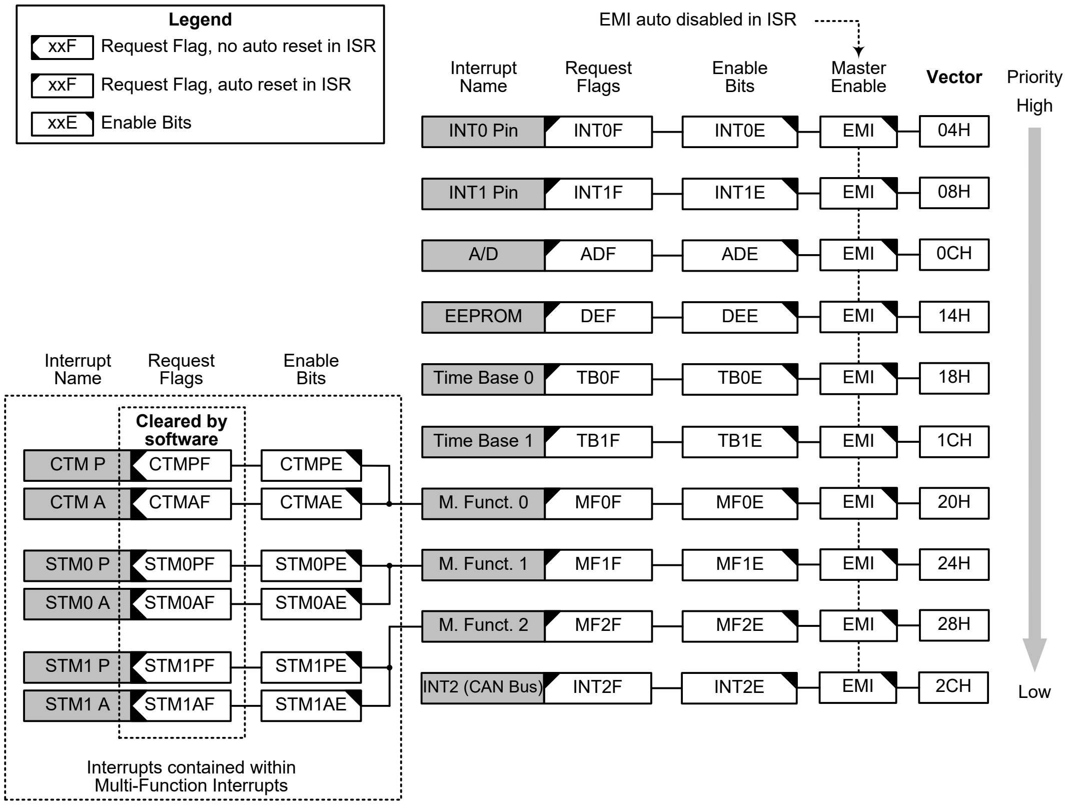
Bit7~4 **REL3~REL0**: Core Release
One digit, BCD-coded.

Bit 3~0 **STEP3~STEP0**: Step of Core Release
One digit, BCD-coded.

Interrupts

Interrupts are an important part of any microcontroller system. When an external event or an internal function such as a Timer Module or an A/D converter requires microcontroller attention, their corresponding interrupt will enforce a temporary suspension of the main program allowing the microcontroller to direct attention to their respective needs. The device contains several external interrupt and internal interrupts functions. The external interrupts are generated by the action of the external INT0~INT1 pins, while the internal interrupts are generated by various internal functions such as the TMs, Time Base, EEPROM and the A/D converter, etc.

The various interrupt enable bits, together with their associated request flags, are shown in the accompanying diagrams with their order of priority. Some interrupt sources have their own individual vector while others share the same multi-function interrupt vector.


Legend
xxF Request Flag, no auto reset in ISR
xxF Request Flag, auto reset in ISR
xxE Enable Bits
EMI auto disabled in ISR
Interrupt Name
Request Flags
Enable Bits
Master Enable
Vector
Priority
High
Low
INT0 Pin INT0F INT0E EMI 04H
INT1 Pin INT1F INT1E EMI 08H
A/D ADF ADE EMI 0CH
EEPROM DEF DEE EMI 14H
Time Base 0 TB0F TB0E EMI 18H
Time Base 1 TB1F TB1E EMI 1CH
M. Funct. 0 MF0F MF0E EMI 20H
M. Funct. 1 MF1F MF1E EMI 24H
M. Funct. 2 MF2F MF2E EMI 28H
INT2 (CAN Bus) INT2F INT2E EMI 2CH
Cleared by software
CTM P CTMPF CTMPE
CTM A CTMAF CTMAE
STM0 P STM0PF STM0PE
STM0 A STM0AF STM0AE
STM1 P STM1PF STM1PE
STM1 A STM1AF STM1AE
Interrupts contained within Multi-Function Interrupts

Interrupt Structure

Interrupt Registers

Overall interrupt control, which basically means the setting of request flags when certain microcontroller conditions occur and the setting of interrupt enable bits by the application program, is controlled by a series of registers, located in the Special Purpose Data Memory, as shown in the accompanying table. The number of registers depends upon the device chosen but fall into three categories. The first is the INTC0~INTC2 registers which setup the primary interrupts, the second is the MFI0~MFI2 registers which setup the Multi-function interrupts. Finally there is an INTEG register to setup the external interrupt trigger edge type.

Each register contains a number of enable bits to enable or disable individual interrupts as well as interrupt flags to I ndicate the presence of an interrupt request. The naming convention of these follows a specific pattern. First is listed an abbreviated interrupt type, then the (optional) number of that interrupt followed by either an “E” for enable/disable bit or “F” for request flag.

Function	Enable Bit	Request Flag	Notes
Global	EMI	—	—
INTn Pins	INTnE	INTnF	n=0~1
INT2 Line (for CAN Bus)	INT2E	INT2F	—
A/D Converter	ADE	ADF	—
Multi-function	MFnE	MFnF	n=0~2
Time Base	TBnE	TBnF	n=0~1
EEPROM	DEE	DEF	—

Function	Enable Bit	Request Flag	Notes
CTM	CTMPE	CTMPF	—
	CTMAE	CTMAF	
STM	STMnPE	STMnPF	n=0~1
	STMnAE	STMnAF	

Interrupt Register Bit Naming Conventions

Register Name	Bit							
	7	6	5	4	3	2	1	0
INTEG	D7	D6	INT2S1	INT2S0	INT1S1	INT1S0	INT0S1	INT0S0
INTC0	—	ADF	INT1F	INT0F	ADE	INT1E	INT0E	EMI
INTC1	TB1F	TB0F	DEF	D4	TB1E	TB0E	DEE	D0
INTC2	INT2F	MF2F	MF1F	MF0F	INT2E	MF2E	MF1E	MF0E
MFI0	—	—	CTMAF	CTMPF	—	—	CTMAE	CTMPE
MFI1	—	—	STM0AF	STM0PF	—	—	STM0AE	STM0PE
MFI2	—	—	STM1AF	STM1PF	—	—	STM1AE	STM1PE

Interrupt Register List

• INTEG Register

Bit	7	6	5	4	3	2	1	0
Name	D7	D6	INT2S1	INT2S0	INT1S1	INT1S0	INT0S1	INT0S0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7~6 **D7~D6**: Reserved bits, should remain unchanged after power-on reset

Bit 5~4 **INT2S1~INT2S0**: Interrupt edge control for INT2 line

Note that the INT2 line is internally connected to the CAN Bus interrupt output signal, RMXINT, therefore the CAN Bus interrupt uses the INT2 interrupt vector.

When the RMXINT interrupt output level is selected to be active high, these bits should be set to "01". When the RMXINT interrupt output level is selected to be active low, these bits should be set to "10".

Bit 3~2 **INT1S1~INT1S0**: Interrupt edge control for INT1 pin
00: Disable
01: Rising edge
10: Falling edge
11: Rising and falling edges

Bit 1~0 **INT0S1~INT0S0**: Interrupt edge control for INT0 pin
00: Disable
01: Rising edge
10: Falling edge
11: Rising and falling edges

• INTC0 Register

Bit	7	6	5	4	3	2	1	0
Name	—	ADF	INT1F	INT0F	ADE	INT1E	INT0E	EMI
R/W	—	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	—	0	0	0	0	0	0	0

Bit 7 Unimplemented, read as "0"

Bit 6 **ADF**: A/D Converter interrupt request flag
0: No request
1: Interrupt request

Bit 5 **INT1F**: INT1 interrupt request flag
0: No request
1: Interrupt request

Bit 4 **INT0F**: INT0 interrupt request flag
0: No request
1: Interrupt request

Bit 3 **ADE**: A/D Converter interrupt control
0: Disable
1: Enable

Bit 2 **INT1E**: INT1 interrupt control
0: Disable
1: Enable

Bit 1 **INT0E**: INT0 interrupt control
0: Disable
1: Enable

Bit 0 **EMI**: Global interrupt control
0: Disable
1: Enable

• INTC1 Register

Bit	7	6	5	4	3	2	1	0
Name	TB1F	TB0F	DEF	D4	TB1E	TB0E	DEE	D0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7 **TB1F**: Time Base 1 interrupt request flag
0: No request
1: Interrupt request

Bit 6 **TB0F**: Time Base 0 interrupt request flag
0: No request
1: Interrupt request

Bit 5 **DEF**: Data EEPROM interrupt request flag
0: No request
1: Interrupt request

Bit 4 **D4**: Reserved bit, should remain unchanged after power-on reset

Bit 3 **TB1E**: Time Base 1 interrupt control
0: Disable
1: Enable

Bit 2 **TB0E**: Time Base01 interrupt control
0: Disable
1: Enable

Bit 1 **DEE**: Data EEPROM interrupt control
0: Disable
1: Enable

Bit 0 **D0**: Reserved bit, should remain unchanged after power-on reset

• INTC2 Register

Bit	7	6	5	4	3	2	1	0
Name	INT2F	MF2F	MF1F	MF0F	INT2E	MF2E	MF1E	MF0E
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7 **INT2F**: INT2 interrupt request flag
0: No request
1: Interrupt request

Note that the INT2 line is internally connected to the CAN Bus interrupt output signal, RMXINT, therefore the CAN Bus interrupt uses the INT2 interrupt vector.

Bit 6 **MF2F**: Multi-function 2 interrupt request flag
0: No request
1: Interrupt request

Bit 5 **MF1F**: Multi-function 1 interrupt request flag
0: No request
1: Interrupt request

Bit 4 **MF0F**: Multi-function 0 interrupt request flag
0: No request
1: Interrupt request

Bit 3 **INT2E**: INT2 interrupt control
0: Disable
1: Enable
Note that the INT2 line is internally connected to the CAN Bus interrupt output signal, RMXINT, therefore the CAN Bus interrupt uses the INT2 interrupt vector.

Bit 2 **MF2E**: Multi-function 2 interrupt control
0: Disable
1: Enable

Bit 1 **MF1E**: Multi-function 1 interrupt control
0: Disable
1: Enable

Bit 0 **MF0E**: Multi-function 0 interrupt control
0: Disable
1: Enable

• MFI0 Register

Bit	7	6	5	4	3	2	1	0
Name	—	—	CTMAF	CTMPF	—	—	CTMAE	CTMPE
R/W	—	—	R/W	R/W	—	—	R/W	R/W
POR	—	—	0	0	—	—	0	0

Bit 7~6 Unimplemented, read as "0"

Bit 5 **CTMAF**: CTM Comparator A match interrupt request flag
0: No request
1: Interrupt request
Note that this bit must be cleared to zero by the application program when the interrupt is serviced.

Bit 4 **CTMPF**: CTM Comparator P match interrupt request flag
0: No request
1: Interrupt request
Note that this bit must be cleared to zero by the application program when the interrupt is serviced.

Bit 3~2 Unimplemented, read as "0"

Bit 1 **CTMAE**: CTM Comparator A match interrupt control
0: Disable
1: Enable

Bit 0 **CTMPE**: CTM Comparator P match interrupt control
0: Disable
1: Enable

• **MFI1 Register**

Bit	7	6	5	4	3	2	1	0
Name	—	—	STM0AF	STM0PF	—	—	STM0AE	STM0PE
R/W	—	—	R/W	R/W	—	—	R/W	R/W
POR	—	—	0	0	—	—	0	0

Bit 7~6 Unimplemented, read as "0"

Bit 5 **STM0AF**: STM0 Comparator A match interrupt request flag
0: No request
1: Interrupt request
Note that this bit must be cleared to zero by the application program when the interrupt is serviced.

Bit 4 **STM0PF**: STM0 Comparator P match interrupt request flag
0: No request
1: Interrupt request
Note that this bit must be cleared to zero by the application program when the interrupt is serviced.

Bit 3~2 Unimplemented, read as "0"

Bit 1 **STM0AE**: STM0 Comparator A match interrupt control
0: Disable
1: Enable

Bit 0 **STM0PE**: STM0 Comparator P match interrupt control
0: Disable
1: Enable

• **MFI2 Register**

Bit	7	6	5	4	3	2	1	0
Name	—	—	STM1AF	STM1PF	—	—	STM1AE	STM1PE
R/W	—	—	R/W	R/W	—	—	R/W	R/W
POR	—	—	0	0	—	—	0	0

Bit 7~6 Unimplemented, read as "0"

Bit 5 **STM1AF**: STM1 Comparator A match interrupt request flag
0: No request
1: Interrupt request
Note that this bit must be cleared to zero by the application program when the interrupt is serviced.

Bit 4 **STM1PF**: STM1 Comparator P match interrupt request flag
0: No request
1: Interrupt request
Note that this bit must be cleared to zero by the application program when the interrupt is serviced.

Bit 3~2 Unimplemented, read as "0"

Bit 1 **STM1AE**: STM1 Comparator A match interrupt control
0: Disable
1: Enable

Bit 0 **STM1PE**: STM1 Comparator P match interrupt control
0: Disable
1: Enable

Interrupt Operation

When the conditions for an interrupt event occur, such as a TM Comparator P or Comparator A or A/D conversion completion, etc, the relevant interrupt request flag will be set. Whether the request flag actually generates a program jump to the relevant interrupt vector is determined by the condition of the interrupt enable bit. If the enable bit is set high then the program will jump to its relevant vector; if the enable bit is zero then although the interrupt request flag is set an actual interrupt will not be generated and the program will not jump to the relevant interrupt vector. The global interrupt enable bit, if cleared to zero, will disable all interrupts.

When an interrupt is generated, the Program Counter, which stores the address of the next instruction to be executed, will be transferred onto the stack. The Program Counter will then be loaded with a new address which will be the value of the corresponding interrupt vector. The microcontroller will then fetch its next instruction from this interrupt vector. The instruction at this vector will usually be a JMP which will jump to another section of program which is known as the interrupt service routine. Here is located the code to control the appropriate interrupt. The interrupt service routine must be terminated with a RETI, which retrieves the original Program Counter address from the stack and allows the microcontroller to continue with normal execution at the point where the interrupt occurred.

Once an interrupt subroutine is serviced, all other interrupts will be blocked, as the global interrupt enable bit, EMI bit will be cleared automatically. This will prevent any further interrupt nesting from occurring. However, if other interrupt requests occur during this interval, although the interrupt will not be immediately serviced, the request flag will still be recorded.

If an interrupt requires immediate servicing while the program is already in another interrupt service routine, the EMI bit should be set after entering the routine, to allow interrupt nesting. If the stack is full, the interrupt request will not be acknowledged, even if the related interrupt is enabled, until the Stack Pointer is decremented. If immediate service is desired, the stack must be prevented from becoming full. In case of simultaneous requests, the Interrupt Structure diagram shows the priority that is applied. All of the interrupt request flags when set will wake up the device if it is in SLEEP or IDLE Mode, however to prevent a wake-up from occurring the corresponding flag should be set before the device is in SLEEP or IDLE Mode.

External Interrupts

The external interrupts are controlled by signal transitions on the pins INT0~INT1. An external interrupt request will take place when the external interrupt request flags, INT0F~INT1F, are set, which will occur when a transition, whose type is chosen by the edge select bits, appears on the external interrupt pins. To allow the program to branch to their respective interrupt vector addresses, the global interrupt enable bit, EMI, and respective external interrupt enable bit, INT0E~INT1E, must first be set. Additionally the correct interrupt edge type must be selected using the INTEG register to enable the external interrupt function and to choose the trigger edge type. As the external interrupt pins are pin-shared with I/O pins, they can only be configured as external interrupt pins if their external interrupt enable bit in the corresponding interrupt register has been set and the external interrupt pin is selected by the corresponding pin-shared function selection bits. The pin must also be setup as an input by setting the corresponding bit in the port control register. When the interrupt is enabled, the stack is not full and the correct transition type appears on the external interrupt pin, a subroutine call to the external interrupt vector, will take place. When the interrupt is serviced, the external interrupt request flags, INT0F~INT1F, will be automatically reset and the EMI bit will be automatically cleared to disable other interrupts. Note that any pull-high resistor selections on the external interrupt pins will remain valid even if the pin is used as an external interrupt input.

The INTEG register is used to select the type of active edge that will trigger the external interrupt. A choice of either rising or falling or both edge types can be chosen to trigger an external interrupt. Note that the INTEG register can also be used to disable the external interrupt function.

CAN Bus Interrupt

The CAN Bus interrupt uses the INT2 interrupt vector as the INT2 line is internally connected to the CAN Bus interrupt output signal, RMXINT.

When a Message is received into Message Object x successfully, an interrupt signal RMXINT will be generated to get the attention of the microcontroller. At this point the RMXINT will output a selected active level and a corresponding edge transition will appear on the INT2 line, the INT2 interrupt request flag, INT2F, is set and an INT2 interrupt request will take place. To allow the program to branch to its interrupt vector address, the global interrupt enable bit, EMI, and respective INT2 interrupt enable bit, INT2E, must first be set. When the interrupt is enabled, the stack is not full and the correct transition type appears on the INT2 signal, a subroutine call to the INT2 interrupt vector will take place. When the interrupt is serviced, the INT2 interrupt request flag, INT2F, will be automatically reset and the EMI bit will be automatically cleared to disable other interrupts.

In addition, when the RMXINT interrupt output level is selected to be active high, the correct interrupt edge type of INT2 is rising edge and vice versa. Therefore the INTEG register should be configured properly to trigger the INT2 interrupt.

A/D Converter Interrupt

An A/D Converter Interrupt request will take place when the A/D Converter Interrupt request flag, ADF, is set, which occurs when the A/D conversion process finishes. To allow the program to branch to its interrupt vector address, the global interrupt enable bit, EMI, and A/D Interrupt enable bit, ADE, must first be set. When the interrupt is enabled, the stack is not full and the A/D conversion process has ended, a subroutine call to the A/D Converter Interrupt vector, will take place. When the interrupt is serviced, the A/D Converter Interrupt flag, ADF, will be automatically cleared. The EMI bit will also be automatically cleared to disable other interrupts.

Multi-function Interrupts

Within the device there are three Multi-function interrupts. Unlike the other independent interrupts, these interrupts have no independent source, but rather are formed from other existing interrupt sources, namely the TM interrupts.

A Multi-function interrupt request will take place when any of the Multi-function interrupt request flags, MFnF, is set. The Multi-function interrupt flags will be set when any of their included functions generate an interrupt request flag. To allow the program to branch to their respective interrupt vector addresses, when the Multi-function interrupt is enabled and the stack is not full, and one of the interrupts contained within each of Multi-function interrupt occurs, a subroutine call to the relevant Multi-function interrupt vector will take place. When the interrupt is serviced, the related Multi-Function request flag will be automatically reset and the EMI bit will be automatically cleared to disable other interrupts.

However, it must be noted that, although the Multi-function Interrupt request flags will be automatically reset when the interrupt is serviced, the request flags from the original source of the Multi-function interrupts will not be automatically reset and must be manually reset by the application program.

Timer Module Interrupts

The Compact and Standard type TMs each has two interrupts, one comes from the comparator A match situation and the other comes from the comparator P match situation. All of the TM interrupts are contained within the Multi-function Interrupts. For all of the TM Types there are two interrupt request flags and two interrupt enable bits. A TM interrupt request will take place when any of the TM request flags are set, a situation which occurs when a TM comparator P or A match situation happens.

To allow the program to branch to their respective interrupt vector addresses, the global interrupt enable bit, EMI, respective TM Interrupt enable bit, and relevant Multi-function Interrupt enable bit, MFnE, must first be set. When the interrupt is enabled, the stack is not full and a TM comparator match situation occurs, a subroutine call to the relevant Multi-function Interrupt vector location, will take place. When the TM interrupt is serviced, the EMI bit will be automatically cleared to disable other interrupts. However, only the related MFnF flag will be automatically cleared. As the TM interrupt request flags will not be automatically cleared, they have to be cleared by the application program.

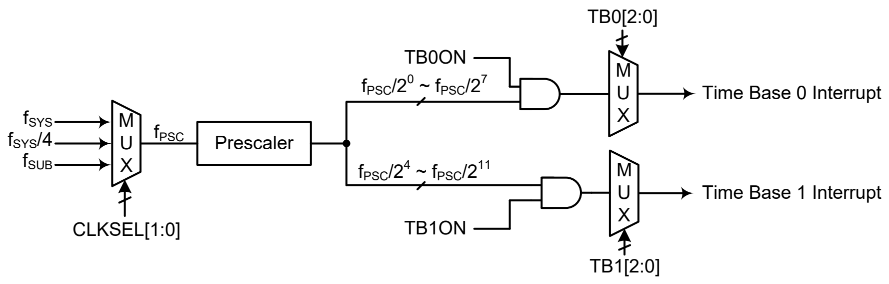
EEPROM Interrupt

An EEPROM Interrupt request will take place when the EEPROM Interrupt request flag, DEF, is set, which occurs when an EEPROM erase or write cycle ends. To allow the program to branch to its interrupt vector address, the global interrupt enable bit, EMI, and EEPROM Interrupt enable bit, DEE, must first be set. When the interrupt is enabled, the stack is not full and an EEPROM erase or write cycle ends, a subroutine call to the EEPROM Interrupt vector will take place. When the EEPROM Interrupt is serviced, the EEPROM Interrupt flag, DEF, will be automatically cleared. The EMI bit will also be automatically cleared to disable other interrupts.

Time Base Interrupts

The function of the Time Base Interrupts is to provide regular time signals in the form of an internal interrupt. They are controlled by the overflow signals from their respective timer functions. When these happens their respective interrupt request flags, TB0F or TB1F will be set. To allow the program to branch to their respective interrupt vector addresses, the global interrupt enable bit, EMI and Time Base enable bits, TB0E or TB1E, must first be set. When the interrupt is enabled, the stack is not full and the Time Base overflows, a subroutine call to their respective vector locations will take place. When the interrupt is serviced, the respective interrupt request flag, TB0F or TB1F, will be automatically cleared, the EMI bit will also be automatically cleared to disable other interrupts.

The purpose of the Time Base Interrupts is to provide an interrupt signal at fixed time periods. Its clock source, f_{PSC}, originates from the internal clock source f_{SYS}, $f_{SYS}/4$ or f_{SUB} and then passes through a divider, the division ratio of which is selected by programming the appropriate bits in the TB0C and TB1C registers to obtain longer interrupt periods whose value ranges. The clock cource that generate f_{PSC}, which in turn controls the Time Base interrupt period, is selected using the CLKSEL[1:0] bits in the PSCR register.



Time Base Interrupts

• PSCR Register

Bit	7	6	5	4	3	2	1	0
Name	—	—	—	—	—	—	CLKSEL1	CLKSEL0
R/W	—	—	—	—	—	—	R/W	R/W
POR	—	—	—	—	—	—	0	0

Bit 7~2 Unimplemented, read as "0"

Bit 1~0 **CLKSEL1~CLKSEL0**: Prescaler clock source f_{PSC} selection
00: f_{SYS}
01: $f_{SYS}/4$
10/11: f_{SUB}

• TB0C Register

Bit	7	6	5	4	3	2	1	0
Name	TB0ON	—	—	—	—	TB02	TB01	TB00
R/W	R/W	—	—	—	—	R/W	R/W	R/W
POR	0	—	—	—	—	0	0	0

Bit 7 **TB0ON**: Time Base 0 Enable Control
0: Disable
1: Enable

Bit 6~3 Unimplemented, read as "0"

Bit 2~0 **TB02~TB00**: Time Base 0 time-out period selection
000: $2^0/f_{PSC}$
001: $2^1/f_{PSC}$
010: $2^2/f_{PSC}$
011: $2^3/f_{PSC}$
100: $2^4/f_{PSC}$
101: $2^5/f_{PSC}$
110: $2^6/f_{PSC}$
111: $2^7/f_{PSC}$

• TB1C Register

Bit	7	6	5	4	3	2	1	0
Name	TB1ON	—	—	—	—	TB12	TB11	TB10
R/W	R/W	—	—	—	—	R/W	R/W	R/W
POR	0	—	—	—	—	0	0	0

Bit 7 **TB1ON**: Time Base 1 Enable Control
0: Disable
1: Enable

Bit 6~3 Unimplemented, read as "0"

Bit 2~0	**TB12~TB10**: Time Base 1 time-out period selection 000: $2^4/f_{PSC}$ 001: $2^5/f_{PSC}$ 010: $2^6/f_{PSC}$ 011: $2^7/f_{PSC}$ 100: $2^8/f_{PSC}$ 101: $2^9/f_{PSC}$ 110: $2^{10}/f_{PSC}$ 111: $2^{11}/f_{PSC}$

Programming Considerations

By disabling the relevant interrupt enable bits, a requested interrupt can be prevented from being serviced, however, once an interrupt request flag is set, it will remain in this condition in the interrupt register until the corresponding interrupt is serviced or until the request flag is cleared by the application program.

Where a certain interrupt is contained within a Multi-function interrupt, then when the interrupt service routine is executed, as only the Multi-function interrupt request flags, MFnF, will be automatically cleared, the individual request flag for the function needs to be cleared by the application program.

It is recommended that programs do not use the “CALL” instruction within the interrupt service subroutine. Interrupts often occur in an unpredictable manner or need to be serviced immediately. If only one stack is left and the interrupt is not well controlled, the original control sequence will be damaged once a CALL subroutine is executed in the interrupt subroutine.

Every interrupt has the capability of waking up the microcontroller when it is in the SLEEP or IDLE Mode, the wake up being generated when the interrupt request flag changes from low to high. If it is required to prevent a certain interrupt from waking up the microcontroller then its respective request flag should be first set high before enter SLEEP or IDLE Mode.

As only the Program Counter is pushed onto the stack, then when the interrupt is serviced, if the contents of the accumulator, status register or other registers are altered by the interrupt service program, their contents should be saved to the memory at the beginning of the interrupt service routine.

To return from an interrupt subroutine, either a RET or RETI instruction may be executed. The RETI instruction in addition to executing a return to the main program also automatically sets the EMI bit high to allow further interrupts. The RET instruction however only executes a return to the main program leaving the EMI bit in its present zero state and therefore disabling the execution of further interrupts.

Application Descriptions

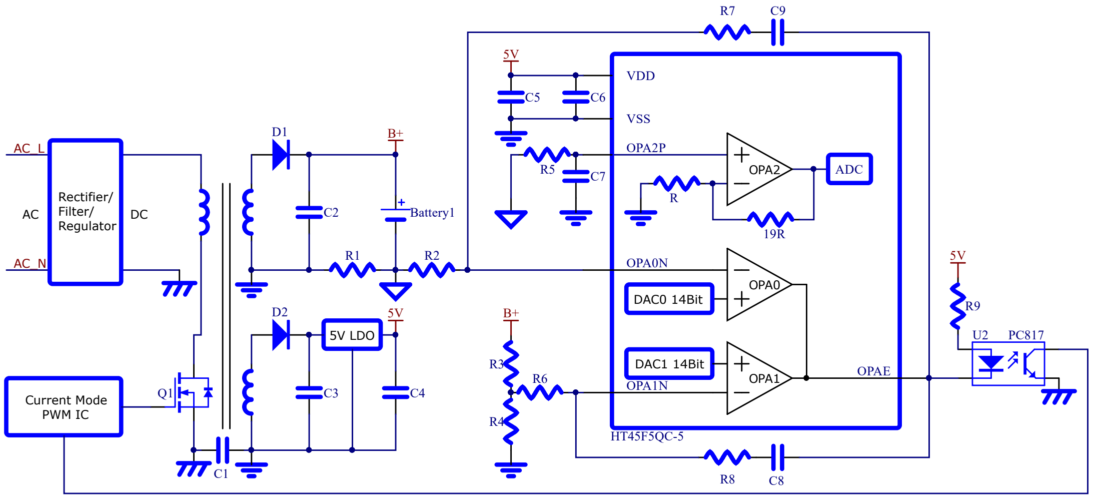
Introduction

According to the battery current condition, the charger can use a Buck circuit to implement charger management. The battery charging contains constant voltage Mode and Constant current Mode. The HT45F5QC-5 device is specifically designed for battery charger applications. The above-mentioned function control can be implemented by the integrated battery charger management, these are described below.

Functional Description

Operating Principle

The device contains a battery charge module which consists of three operational amplifier (OPA0~OPA2) functions, two 14-bit D/A converter (DAC0 and DAC1) functions. The open drain OPA0~OPA1 and DAC0~DAC1 are used for constant current and constant voltage signal control. The OPA output can directly drive the photo-coupler, which makes the PWM IC on the primary side can implement output power adjustment, shown in the figure below. The internal 20/40 times amplifier OPA2 is used to amplify the charge current signal, thus increasing the current resolution and reducing the detecting resistance power consumption. The constant voltage mode, constant current mode and constant current and constant voltage resolution increasing method are described as follows.



Battery Charge Module

Constant Current Mode Description

Constant current charging means that the charge current will remain at a constant value no matter how the battery internal resistance changes. The principle is that the charge current flows through the detecting resistor R1 and in turn generates a voltage, which will be input to the OPA0 negative terminal through the OPA0N pin. The difference between the OPA0N voltage and the D/A converter voltage is amplified and then output on the OPAE pin. This output will be sent to the PWM IC via a photo-coupler. If the OPA0N voltage is lower than the DAC0 voltage, the PWM IC will increase the PWM duty cycle and vice versa.

Note: The DA0H and DA0L registers are used to set the maximum current threshold.

Constant Voltage Mode Description

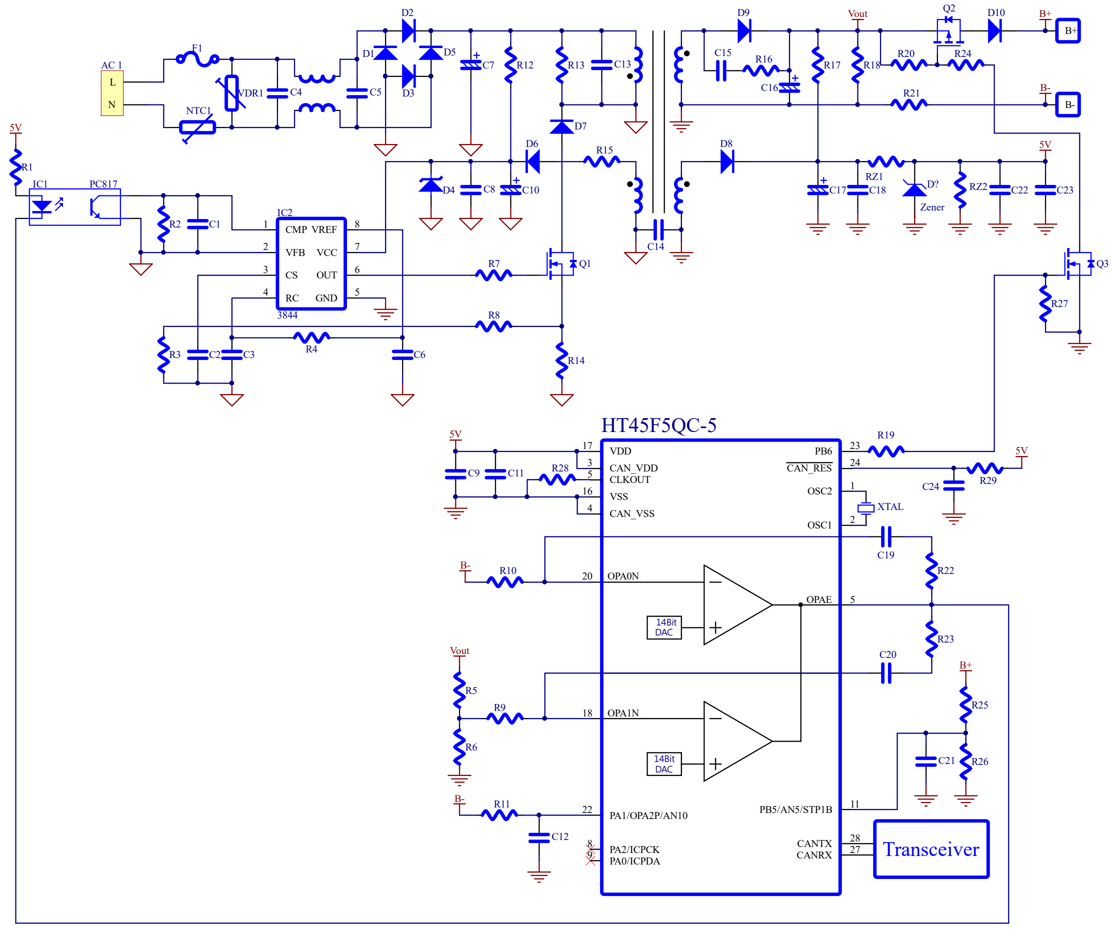
Constant voltage charging means that the charge voltage will remain at a constant value no matter how the battery internal resistance changes. The principle is that the charge voltage B+ is divided by R3 and R4 resistors and then supplied to the OPA1 negative terminal through the OPA1N pin. The difference between the OPA1N voltage and the D/A converter voltage is amplified and then output on the OPAE pin. This output will be sent to the PWM IC via a photo-coupler. If the OPA1N voltage is lower than the DAC1 voltage, the PWM IC will increase the PWM duty cycle and vice versa.

Note: The DA1H and DA1L registers are used to set the maximum voltage threshold.

Improving the Constant Current and Constant Voltage Resolution

If the internal 14-bit D/A Converter resolution is not high enough, the OPA0 and OPA1 positive terminals can be supplied by an external divider resistor to increase the voltage and current resolution.

Hardware Circuit



Charger Application Circuit

Instruction Set

Introduction

Central to the successful operation of any microcontroller is its instruction set, which is a set of program instruction codes that directs the microcontroller to perform certain operations. In the case of Holtek microcontroller, a comprehensive and flexible set of over 60 instructions is provided to enable programmers to implement their application with the minimum of programming overheads.

For easier understanding of the various instruction codes, they have been subdivided into several functional groupings.

Instruction Timing

Most instructions are implemented within one instruction cycle. The exceptions to this are branch, call, or table read instructions where two instruction cycles are required. One instruction cycle is equal to 4 system clock cycles, therefore in the case of an 8MHz system oscillator, most instructions would be implemented within 0.5µs and branch or call instructions would be implemented within 1µs. Although instructions which require one more cycle to implement are generally limited to the JMP, CALL, RET, RETI and table read instructions, it is important to realize that any other instructions which involve manipulation of the Program Counter Low register or PCL will also take one more cycle to implement. As instructions which change the contents of the PCL will imply a direct jump to that new address, one more cycle will be required. Examples of such instructions would be "CLR PCL" or "MOV PCL, A". For the case of skip instructions, it must be noted that if the result of the comparison involves a skip operation then this will also take one more cycle, if no skip is involved then only one cycle is required.

Moving and Transferring Data

The transfer of data within the microcontroller program is one of the most frequently used operations. Making use of several kinds of MOV instructions, data can be transferred from registers to the Accumulator and vice-versa as well as being able to move specific immediate data directly into the Accumulator. One of the most important data transfer applications is to receive data from the input ports and transfer data to the output ports.

Arithmetic Operations

The ability to perform certain arithmetic operations and data manipulation is a necessary feature of most microcontroller applications. Within the Holtek microcontroller instruction set are a range of add and subtract instruction mnemonics to enable the necessary arithmetic to be carried out. Care must be taken to ensure correct handling of carry and borrow data when results exceed 255 for addition and less than 0 for subtraction. The increment and decrement instructions such as INC, INCA, DEC and DECA provide a simple means of increasing or decreasing by a value of one of the values in the destination specified.

Logical and Rotate Operation

The standard logical operations such as AND, OR, XOR and CPL all have their own instruction within the Holtek microcontroller instruction set. As with the case of most instructions involving data manipulation, data must pass through the Accumulator which may involve additional programming steps. In all logical data operations, the zero flag may be set if the result of the operation is zero. Another form of logical data manipulation comes from the rotate instructions such as RR, RL, RRC and RLC which provide a simple means of rotating one bit right or left. Different rotate instructions exist depending on program requirements. Rotate instructions are useful for serial port programming applications where data can be rotated from an internal register into the Carry bit from where it can be examined and the necessary serial bit set high or low. Another application which rotate data operations are used is to implement multiplication and division calculations.

Branches and Control Transfer

Program branching takes the form of either jumps to specified locations using the JMP instruction or to a subroutine using the CALL instruction. They differ in the sense that in the case of a subroutine call, the program must return to the instruction immediately when the subroutine has been carried out. This is done by placing a return instruction "RET" in the subroutine which will cause the program to jump back to the address right after the CALL instruction. In the case of a JMP instruction, the program simply jumps to the desired location. There is no requirement to jump back to the original jumping off point as in the case of the CALL instruction. One special and extremely useful set of branch instructions are the conditional branches. Here a decision is first made regarding the condition of a certain data memory or individual bits. Depending upon the conditions, the program will continue with the next instruction or skip over it and jump to the following instruction. These instructions are the key to decision making and branching within the program perhaps determined by the condition of certain input switches or by the condition of internal data bits.

Bit Operations

The ability to provide single bit operations on Data Memory is an extremely flexible feature of all Holtek microcontrollers. This feature is especially useful for output port bit programming where individual bits or port pins can be directly set high or low using either the "SET [m].i" or "CLR [m].i" instructions respectively. The feature removes the need for programmers to first read the 8-bit output port, manipulate the input data to ensure that other bits are not changed and then output the port with the correct new data. This read-modify-write process is taken care of automatically when these bit operation instructions are used.

Table Read Operations

Data storage is normally implemented by using registers. However, when working with large amounts of fixed data, the volume involved often makes it inconvenient to store the fixed data in the Data Memory. To overcome this problem, Holtek microcontrollers allow an area of Program Memory to be setup as a table where data can be directly stored. A set of easy to use instructions provides the means by which this fixed data can be referenced and retrieved from the Program Memory.

Other Operations

In addition to the above functional instructions, a range of other instructions also exist such as the "HALT" instruction for Power-down operations and instructions to control the operation of the Watchdog Timer for reliable program operations under extreme electric or electromagnetic environments. For their relevant operations, refer to the functional related sections.

Instruction Set Summary

The instructions related to the data memory access in the following table can be used when the desired data memory is located in Data Memory sector 0.

Table Conventions

x: Bits immediate data
m: Data Memory address
A: Accumulator
i: 0~7 number of bits
addr: Program memory address

Mnemonic	Description	Cycles	Flag Affected
Arithmetic			
ADD A,[m]	Add Data Memory to ACC	1	Z, C, AC, OV, SC
ADDM A,[m]	Add ACC to Data Memory	1Note	Z, C, AC, OV, SC
ADD A,x	Add immediate data to ACC	1	Z, C, AC, OV, SC
ADC A,[m]	Add Data Memory to ACC with Carry	1	Z, C, AC, OV, SC
ADCM A,[m]	Add ACC to Data memory with Carry	1Note	Z, C, AC, OV, SC
SUB A,x	Subtract immediate data from the ACC	1	Z, C, AC, OV, SC, CZ
SUB A,[m]	Subtract Data Memory from ACC	1	Z, C, AC, OV, SC, CZ
SUBM A,[m]	Subtract Data Memory from ACC with result in Data Memory	1Note	Z, C, AC, OV, SC, CZ
SBC A,x	Subtract immediate data from ACC with Carry	1	Z, C, AC, OV, SC, CZ
SBC A,[m]	Subtract Data Memory from ACC with Carry	1	Z, C, AC, OV, SC, CZ
SBCM A,[m]	Subtract Data Memory from ACC with Carry, result in Data Memory	1Note	Z, C, AC, OV, SC, CZ
DAA [m]	Decimal adjust ACC for Addition with result in Data Memory	1Note	C
Logic Operation			
AND A,[m]	Logical AND Data Memory to ACC	1	Z
OR A,[m]	Logical OR Data Memory to ACC	1	Z
XOR A,[m]	Logical XOR Data Memory to ACC	1	Z
ANDM A,[m]	Logical AND ACC to Data Memory	1Note	Z
ORM A,[m]	Logical OR ACC to Data Memory	1Note	Z
XORM A,[m]	Logical XOR ACC to Data Memory	1Note	Z
AND A,x	Logical AND immediate Data to ACC	1	Z
OR A,x	Logical OR immediate Data to ACC	1	Z
XOR A,x	Logical XOR immediate Data to ACC	1	Z
CPL [m]	Complement Data Memory	1Note	Z
CPLA [m]	Complement Data Memory with result in ACC	1	Z
Increment & Decrement			
INCA [m]	Increment Data Memory with result in ACC	1	Z
INC [m]	Increment Data Memory	1Note	Z
DECA [m]	Decrement Data Memory with result in ACC	1	Z
DEC [m]	Decrement Data Memory	1Note	Z
Rotate			
RRA [m]	Rotate Data Memory right with result in ACC	1	None
RR [m]	Rotate Data Memory right	1Note	None
RRCA [m]	Rotate Data Memory right through Carry with result in ACC	1	C
RRC [m]	Rotate Data Memory right through Carry	1Note	C
RLA [m]	Rotate Data Memory left with result in ACC	1	None
RL [m]	Rotate Data Memory left	1Note	None
RLCA [m]	Rotate Data Memory left through Carry with result in ACC	1	C
RLC [m]	Rotate Data Memory left through Carry	1Note	C

Mnemonic	Description	Cycles	Flag Affected
Data Move			
MOV A,[m]	Move Data Memory to ACC	1	None
MOV [m],A	Move ACC to Data Memory	1Note	None
MOV A,x	Move immediate data to ACC	1	None
Bit Operation			
CLR [m].i	Clear bit of Data Memory	1Note	None
SET [m].i	Set bit of Data Memory	1Note	None
Branch Operation			
JMP addr	Jump unconditionally	2	None
SZ [m]	Skip if Data Memory is zero	1Note	None
SZA [m]	Skip if Data Memory is zero with data movement to ACC	1Note	None
SZ [m].i	Skip if bit i of Data Memory is zero	1Note	None
SNZ [m]	Skip if Data Memory is not zero	1Note	None
SNZ [m].i	Skip if bit i of Data Memory is not zero	1Note	None
SIZ [m]	Skip if increment Data Memory is zero	1Note	None
SDZ [m]	Skip if decrement Data Memory is zero	1Note	None
SIZA [m]	Skip if increment Data Memory is zero with result in ACC	1Note	None
SDZA [m]	Skip if decrement Data Memory is zero with result in ACC	1Note	None
CALL addr	Subroutine call	2	None
RET	Return from subroutine	2	None
RET A,x	Return from subroutine and load immediate data to ACC	2	None
RETI	Return from interrupt	2	None
Table Read Operation			
TABRD [m]	Read table (specific page) to TBLH and Data Memory	2Note	None
TABRDL [m]	Read table (last page) to TBLH and Data Memory	2Note	None
ITABRD [m]	Increment table pointer TBLP first and Read table (specific page) to TBLH and Data Memory	2Note	None
ITABRDL [m]	Increment table pointer TBLP first and Read table (last page) to TBLH and Data Memory	2Note	None
Miscellaneous			
NOP	No operation	1	None
CLR [m]	Clear Data Memory	1Note	None
SET [m]	Set Data Memory	1Note	None
CLR WDT	Clear Watchdog Timer	1	TO, PDF
SWAP [m]	Swap nibbles of Data Memory	1Note	None
SWAPA [m]	Swap nibbles of Data Memory with result in ACC	1	None
HALT	Enter power down mode	1	TO, PDF

Note: 1. For skip instructions, if the result of the comparison involves a skip then two cycles are required, if no skip takes place only one cycle is required.

2. Any instruction which changes the contents of the PCL will also require 2 cycles for execution.

Extended Instruction Set

The extended instructions are used to support the full range address access for the data memory. When the accessed data memory is located in any data memory sector except sector 0, the extended instruction can be used to directly access the data memory instead of using the indirect addressing access. This can not only reduce the use of Flash memory space but also improve the CPU execution efficiency.

Mnemonic	Description	Cycles	Flag Affected
Arithmetic			
LADD A,[m]	Add Data Memory to ACC	2	Z, C, AC, OV, SC
LADDM A,[m]	Add ACC to Data Memory	2^{Note}	Z, C, AC, OV, SC
LADC A,[m]	Add Data Memory to ACC with Carry	2	Z, C, AC, OV, SC
LADCM A,[m]	Add ACC to Data memory with Carry	2^{Note}	Z, C, AC, OV, SC
LSUB A,[m]	Subtract Data Memory from ACC	2	Z, C, AC, OV, SC, CZ
LSUBM A,[m]	Subtract Data Memory from ACC with result in Data Memory	2^{Note}	Z, C, AC, OV, SC, CZ
LSBC A,[m]	Subtract Data Memory from ACC with Carry	2	Z, C, AC, OV, SC, CZ
LSBCM A,[m]	Subtract Data Memory from ACC with Carry, result in Data Memory	2^{Note}	Z, C, AC, OV, SC, CZ
LDAA [m]	Decimal adjust ACC for Addition with result in Data Memory	2^{Note}	C
Logic Operation			
LAND A,[m]	Logical AND Data Memory to ACC	2	Z
LOR A,[m]	Logical OR Data Memory to ACC	2	Z
LXOR A,[m]	Logical XOR Data Memory to ACC	2	Z
LANDM A,[m]	Logical AND ACC to Data Memory	2^{Note}	Z
LORM A,[m]	Logical OR ACC to Data Memory	2^{Note}	Z
LXORM A,[m]	Logical XOR ACC to Data Memory	2^{Note}	Z
LCPL [m]	Complement Data Memory	2^{Note}	Z
LCPLA [m]	Complement Data Memory with result in ACC	2	Z
Increment & Decrement			
LINCA [m]	Increment Data Memory with result in ACC	2	Z
LINC [m]	Increment Data Memory	2^{Note}	Z
LDECA [m]	Decrement Data Memory with result in ACC	2	Z
LDEC [m]	Decrement Data Memory	2^{Note}	Z
Rotate			
LRRA [m]	Rotate Data Memory right with result in ACC	2	None
LRR [m]	Rotate Data Memory right	2^{Note}	None
LRRCA [m]	Rotate Data Memory right through Carry with result in ACC	2	C
LRRC [m]	Rotate Data Memory right through Carry	2^{Note}	C
LRLA [m]	Rotate Data Memory left with result in ACC	2	None
LRL [m]	Rotate Data Memory left	2^{Note}	None
LRLCA [m]	Rotate Data Memory left through Carry with result in ACC	2	C
LRLC [m]	Rotate Data Memory left through Carry	2^{Note}	C
Data Move			
LMOV A,[m]	Move Data Memory to ACC	2	None
LMOV [m],A	Move ACC to Data Memory	2^{Note}	None
Bit Operation			
LCLR [m].i	Clear bit of Data Memory	2^{Note}	None
LSET [m].i	Set bit of Data Memory	2^{Note}	None

Mnemonic	Description	Cycles	Flag Affected
Branch			
LSZ [m]	Skip if Data Memory is zero	2Note	None
LSZA [m]	Skip if Data Memory is zero with data movement to ACC	2Note	None
LSNZ [m]	Skip if Data Memory is not zero	2Note	None
LSZ [m].i	Skip if bit i of Data Memory is zero	2Note	None
LSNZ [m].i	Skip if bit i of Data Memory is not zero	2Note	None
LSIZ [m]	Skip if increment Data Memory is zero	2Note	None
LSDZ [m]	Skip if decrement Data Memory is zero	2Note	None
LSIZA [m]	Skip if increment Data Memory is zero with result in ACC	2Note	None
LSDZA [m]	Skip if decrement Data Memory is zero with result in ACC	2Note	None
Table Read			
LTABRD [m]	Read table (specific page) to TBLH and Data Memory	3Note	None
LTABRDL [m]	Read table (last page) to TBLH and Data Memory	3Note	None
LITABRD [m]	Increment table pointer TBLP first and Read table (specific page) to TBLH and Data Memory	3Note	None
LITABRDL [m]	Increment table pointer TBLP first and Read table (last page) to TBLH and Data Memory	3Note	None
Miscellaneous			
LCLR [m]	Clear Data Memory	2Note	None
LSET [m]	Set Data Memory	2Note	None
LSWAP [m]	Swap nibbles of Data Memory	2Note	None
LSWAPA [m]	Swap nibbles of Data Memory with result in ACC	2	None

Note: 1. For these extended skip instructions, if the result of the comparison involves a skip then three cycles are required, if no skip takes place two cycles is required.

2. Any extended instruction which changes the contents of the PCL register will also require three cycles for execution.

Instruction Definition

ADC A,[m]	Add Data Memory to ACC with Carry
Description	The contents of the specified Data Memory, Accumulator and the carry flag are added. The result is stored in the Accumulator.
Operation	ACC ← ACC + [m] + C
Affected flag(s)	OV, Z, AC, C, SC

ADCM A,[m]	Add ACC to Data Memory with Carry
Description	The contents of the specified Data Memory, Accumulator and the carry flag are added. The result is stored in the specified Data Memory.
Operation	[m] ← ACC + [m] + C
Affected flag(s)	OV, Z, AC, C, SC

ADD A,[m]	Add Data Memory to ACC
Description	The contents of the specified Data Memory and the Accumulator are added. The result is stored in the Accumulator.
Operation	ACC ← ACC + [m]
Affected flag(s)	OV, Z, AC, C, SC

ADD A,x	Add immediate data to ACC
Description	The contents of the Accumulator and the specified immediate data are added. The result is stored in the Accumulator.
Operation	ACC ← ACC + x
Affected flag(s)	OV, Z, AC, C, SC

ADDM A,[m]	Add ACC to Data Memory
Description	The contents of the specified Data Memory and the Accumulator are added. The result is stored in the specified Data Memory.
Operation	[m] ← ACC + [m]
Affected flag(s)	OV, Z, AC, C, SC

AND A,[m]	Logical AND Data Memory to ACC
Description	Data in the Accumulator and the specified Data Memory perform a bitwise logical AND operation. The result is stored in the Accumulator.
Operation	ACC ← ACC ″AND″ [m]
Affected flag(s)	Z

AND A,x	Logical AND immediate data to ACC
Description	Data in the Accumulator and the specified immediate data perform a bit wise logical AND operation. The result is stored in the Accumulator.
Operation	ACC ← ACC ″AND″ x
Affected flag(s)	Z

ANDM A,[m]	Logical AND ACC to Data Memory
Description	Data in the specified Data Memory and the Accumulator perform a bitwise logical AND operation. The result is stored in the Data Memory.
Operation	[m] ← ACC ″AND″ [m]
Affected flag(s)	Z

CALL addr	Subroutine call
Description	Unconditionally calls a subroutine at the specified address. The Program Counter then increments by 1 to obtain the address of the next instruction which is then pushed onto the stack. The specified address is then loaded and the program continues execution from this new address. As this instruction requires an additional operation, it is a two cycle instruction.
Operation	Stack ← Program Counter + 1 Program Counter ← addr
Affected flag(s)	None

CLR [m]	Clear Data Memory
Description	Each bit of the specified Data Memory is cleared to 0.
Operation	[m] ← 00H
Affected flag(s)	None

CLR [m].i	Clear bit of Data Memory
Description	Bit i of the specified Data Memory is cleared to 0.
Operation	[m].i ← 0
Affected flag(s)	None

CLR WDT	Clear Watchdog Timer
Description	The TO, PDF flags and the WDT are all cleared.

Operation	WDT cleared
	TO ← 0
	PDF ← 0

Affected flag(s)	TO, PDF

CPL [m]	Complement Data Memory
Description	Each bit of the specified Data Memory is logically complemented (1′s complement). Bits which previously contained a 1 are changed to 0 and vice versa.
Operation	[m] ← $\overline{[m]}$
Affected flag(s)	Z

CPLA [m]	Complement Data Memory with result in ACC
Description	Each bit of the specified Data Memory is logically complemented (1′s complement). Bits which previously contained a 1 are changed to 0 and vice versa. The complemented result is stored in the Accumulator and the contents of the Data Memory remain unchanged.
Operation	ACC ← $\overline{[m]}$
Affected flag(s)	Z

DAA [m]	Decimal-Adjust ACC for addition with result in Data Memory
Description	Convert the contents of the Accumulator value to a BCD (Binary Coded Decimal) value resulting from the previous addition of two BCD variables. If the low nibble is greater than 9 or if AC flag is set, then a value of 6 will be added to the low nibble. Otherwise the low nibble remains unchanged. If the high nibble is greater than 9 or if the C flag is set, then a value of 6 will be added to the high nibble. Essentially, the decimal conversion is performed by adding 00H, 06H, 60H or 66H depending on the Accumulator and flag conditions. Only the C flag may be affected by this instruction which indicates that if the original BCD sum is greater than 100, it allows multiple precision decimal addition.
Operation	[m] ← ACC + 00H or [m] ← ACC + 06H or [m] ← ACC + 60H or [m] ← ACC + 66H
Affected flag(s)	C

DEC [m]	Decrement Data Memory
Description	Data in the specified Data Memory is decremented by 1.
Operation	[m] ← [m] − 1
Affected flag(s)	Z

DECA [m]	Decrement Data Memory with result in ACC
Description	Data in the specified Data Memory is decremented by 1. The result is stored in the Accumulator. The contents of the Data Memory remain unchanged.
Operation	ACC ← [m] − 1
Affected flag(s)	Z

HALT	Enter power down mode
Description	This instruction stops the program execution and turns off the system clock. The contents of the Data Memory and registers are retained. The WDT and prescaler are cleared. The power down flag PDF is set and the WDT time-out flag TO is cleared.

Operation	TO ← 0
	PDF ← 1

Affected flag(s)	TO, PDF

INC [m]	Increment Data Memory
Description	Data in the specified Data Memory is incremented by 1.
Operation	[m] ← [m] + 1
Affected flag(s)	Z

INCA [m]	Increment Data Memory with result in ACC
Description	Data in the specified Data Memory is incremented by 1. The result is stored in the Accumulator. The contents of the Data Memory remain unchanged.
Operation	ACC ← [m] + 1
Affected flag(s)	Z

JMP addr	Jump unconditionally
Description	The contents of the Program Counter are replaced with the specified address. Program execution then continues from this new address. As this requires the insertion of a dummy instruction while the new address is loaded, it is a two cycle instruction.
Operation	Program Counter ← addr
Affected flag(s)	None

MOV A,[m]	Move Data Memory to ACC
Description	The contents of the specified Data Memory are copied to the Accumulator.
Operation	ACC ← [m]
Affected flag(s)	None

MOV A,x	Move immediate data to ACC
Description	The immediate data specified is loaded into the Accumulator.
Operation	ACC ← x
Affected flag(s)	None

MOV [m],A	Move ACC to Data Memory
Description	The contents of the Accumulator are copied to the specified Data Memory.
Operation	[m] ← ACC
Affected flag(s)	None

NOP	No operation
Description	No operation is performed. Execution continues with the next instruction.
Operation	No operation
Affected flag(s)	None

OR A,[m]	Logical OR Data Memory to ACC
Description	Data in the Accumulator and the specified Data Memory perform a bitwise logical OR operation. The result is stored in the Accumulator.
Operation	ACC ← ACC ″OR″ [m]
Affected flag(s)	Z

OR A,x	Logical OR immediate data to ACC
Description	Data in the Accumulator and the specified immediate data perform a bitwise logical OR operation. The result is stored in the Accumulator.
Operation	ACC ← ACC ″OR″ x
Affected flag(s)	Z

ORM A,[m]	Logical OR ACC to Data Memory
Description	Data in the specified Data Memory and the Accumulator perform a bitwise logical OR operation. The result is stored in the Data Memory.
Operation	[m] ← ACC ″OR″ [m]
Affected flag(s)	Z

RET	Return from subroutine
Description	The Program Counter is restored from the stack. Program execution continues at the restored address.
Operation	Program Counter ← Stack
Affected flag(s)	None

RET A,x	Return from subroutine and load immediate data to ACC
Description	The Program Counter is restored from the stack and the Accumulator loaded with the specified immediate data. Program execution continues at the restored address.
Operation	Program Counter ← Stack ACC ← x
Affected flag(s)	None

RETI	Return from interrupt
Description	The Program Counter is restored from the stack and the interrupts are re-enabled by setting the EMI bit. EMI is the master interrupt global enable bit. If an interrupt was pending when the RETI instruction is executed, the pending Interrupt routine will be processed before returning to the main program.
Operation	Program Counter ← Stack EMI ← 1
Affected flag(s)	None

RL [m]	Rotate Data Memory left
Description	The contents of the specified Data Memory are rotated left by 1 bit with bit 7 rotated into bit 0.
Operation	[m].(i+1) ← [m].i; (i=0~6) [m].0 ← [m].7
Affected flag(s)	None

RLA [m]	Rotate Data Memory left with result in ACC
Description	The contents of the specified Data Memory are rotated left by 1 bit with bit 7 rotated into bit 0. The rotated result is stored in the Accumulator and the contents of the Data Memory remain unchanged.

Operation	ACC.(i+1) ← [m].i; (i=0~6)
	ACC.0 ← [m].7

Affected flag(s)	None

RLC [m]	Rotate Data Memory left through Carry
Description	The contents of the specified Data Memory and the carry flag are rotated left by 1 bit. Bit 7 replaces the Carry bit and the original carry flag is rotated into bit 0.
Operation	[m].(i+1) ← [m].i; (i=0~6) [m].0 ← C C ← [m].7
Affected flag(s)	C

RLCA [m]	Rotate Data Memory left through Carry with result in ACC
Description	Data in the specified Data Memory and the carry flag are rotated left by 1 bit. Bit 7 replaces the Carry bit and the original carry flag is rotated into the bit 0. The rotated result is stored in the Accumulator and the contents of the Data Memory remain unchanged.

Operation	ACC.(i+1) ← [m].i; (i=0~6)
	ACC.0 ← C
	C ← [m].7

Affected flag(s)	C

RR [m]	Rotate Data Memory right
Description	The contents of the specified Data Memory are rotated right by 1 bit with bit 0 rotated into bit 7.
Operation	[m].i ← [m].(i+1); (i=0~6) [m].7 ← [m].0
Affected flag(s)	None

RRA [m]	Rotate Data Memory right with result in ACC
Description	Data in the specified Data Memory is rotated right by 1 bit with bit 0 rotated into bit 7. The rotated result is stored in the Accumulator and the contents of the Data Memory remain unchanged.

Operation	ACC.i ← [m].(i+1); (i=0~6)
	ACC.7 ← [m].0

Affected flag(s)	None

RRC [m]	Rotate Data Memory right through Carry
Description	The contents of the specified Data Memory and the carry flag are rotated right by 1 bit. Bit 0 replaces the Carry bit and the original carry flag is rotated into bit 7.
Operation	[m].i ← [m].(i+1); (i=0~6) [m].7 ← C C ← [m].0
Affected flag(s)	C

RRCA [m]	Rotate Data Memory right through Carry with result in ACC
Description	Data in the specified Data Memory and the carry flag are rotated right by 1 bit. Bit 0 replaces the Carry bit and the original carry flag is rotated into bit 7. The rotated result is stored in the Accumulator and the contents of the Data Memory remain unchanged.

Operation	ACC.i ← [m].(i+1); (i=0~6)
	ACC.7 ← C
	C ← [m].0

Affected flag(s)	C

SBC A,[m]	Subtract Data Memory from ACC with Carry
Description	The contents of the specified Data Memory and the complement of the carry flag are subtracted from the Accumulator. The result is stored in the Accumulator. Note that if the result of subtraction is negative, the C flag will be cleared to 0, otherwise if the result is positive or zero, the C flag will be set to 1.
Operation	ACC ← ACC − [m] − $\overline{C}$
Affected flag(s)	OV, Z, AC, C, SC, CZ

SBC A, x	Subtract immediate data from ACC with Carry
Description	The immediate data and the complement of the carry flag are subtracted from the Accumulator. The result is stored in the Accumulator. Note that if the result of subtraction is negative, the C flag will be cleared to 0, otherwise if the result is positive or zero, the C flag will be set to 1.
Operation	ACC ← ACC – [m] – $\overline{C}$
Affected flag(s)	OV, Z, AC, C, SC, CZ

SBCM A,[m]	Subtract Data Memory from ACC with Carry and result in Data Memory
Description	The contents of the specified Data Memory and the complement of the carry flag are subtracted from the Accumulator. The result is stored in the Data Memory. Note that if the result of subtraction is negative, the C flag will be cleared to 0, otherwise if the result is positive or zero, the C flag will be set to 1.
Operation	[m] ← ACC − [m] − $\overline{C}$
Affected flag(s)	OV, Z, AC, C, SC, CZ

SDZ [m]	Skip if decrement Data Memory is 0
Description	The contents of the specified Data Memory are first decremented by 1. If the result is 0 the following instruction is skipped. As this requires the insertion of a dummy instruction while the next instruction is fetched, it is a two cycle instruction. If the result is not 0 the program proceeds with the following instruction.
Operation	[m] ← [m] − 1 Skip if [m]=0
Affected flag(s)	None

SDZA [m]	Skip if decrement Data Memory is zero with result in ACC
Description	The contents of the specified Data Memory are first decremented by 1. If the result is 0, the following instruction is skipped. The result is stored in the Accumulator but the specified Data Memory contents remain unchanged. As this requires the insertion of a dummy instruction while the next instruction is fetched, it is a two cycle instruction. If the result is not 0, the program proceeds with the following instruction.
Operation	ACC ← [m] − 1 Skip if ACC=0
Affected flag(s)	None

SET [m]	Set Data Memory
Description	Each bit of the specified Data Memory is set to 1.
Operation	[m] ← FFH
Affected flag(s)	None

SET [m].i	Set bit of Data Memory
Description	Bit i of the specified Data Memory is set to 1.
Operation	[m].i ← 1
Affected flag(s)	None

SIZ [m]	Skip if increment Data Memory is 0
Description	The contents of the specified Data Memory are first incremented by 1. If the result is 0, the following instruction is skipped. As this requires the insertion of a dummy instruction while the next instruction is fetched, it is a two cycle instruction. If the result is not 0 the program proceeds with the following instruction.
Operation	[m] ← [m] + 1 Skip if [m]=0
Affected flag(s)	None

SIZA [m]	Skip if increment Data Memory is zero with result in ACC
Description	The contents of the specified Data Memory are first incremented by 1. If the result is 0, the following instruction is skipped. The result is stored in the Accumulator but the specified Data Memory contents remain unchanged. As this requires the insertion of a dummy instruction while the next instruction is fetched, it is a two cycle instruction. If the result is not 0 the program proceeds with the following instruction.
Operation	ACC ← [m] + 1 Skip if ACC=0
Affected flag(s)	None

SNZ [m].i	Skip if bit i of Data Memory is not 0
Description	If bit i of the specified Data Memory is not 0, the following instruction is skipped. As this requires the insertion of a dummy instruction while the next instruction is fetched, it is a two cycle instruction. If the result is 0 the program proceeds with the following instruction.
Operation	Skip if [m].i ≠ 0
Affected flag(s)	None

SNZ [m]	Skip if Data Memory is not 0
Description	The contents of the specified Data Memory are read out and then written back to the specified Data Memory again. If the specified Data Memory is not 0, the following instruction is skipped. As this requires the insertion of a dummy instruction while the next instruction is fetched, it is a two cycle instruction. If the result is 0 the program proceeds with the following instruction.
Operation	Skip if [m]≠ 0
Affected flag(s)	None

SUB A,[m]	Subtract Data Memory from ACC
Description	The specified Data Memory is subtracted from the contents of the Accumulator. The result is stored in the Accumulator. Note that if the result of subtraction is negative, the C flag will be cleared to 0, otherwise if the result is positive or zero, the C flag will be set to 1.
Operation	ACC ← ACC − [m]
Affected flag(s)	OV, Z, AC, C, SC, CZ

SUBM A,[m]	Subtract Data Memory from ACC with result in Data Memory
Description	The specified Data Memory is subtracted from the contents of the Accumulator. The result is stored in the Data Memory. Note that if the result of subtraction is negative, the C flag will be cleared to 0, otherwise if the result is positive or zero, the C flag will be set to 1.
Operation	[m] ← ACC − [m]
Affected flag(s)	OV, Z, AC, C, SC, CZ

SUB A,x	Subtract immediate data from ACC
Description	The immediate data specified by the code is subtracted from the contents of the Accumulator. The result is stored in the Accumulator. Note that if the result of subtraction is negative, the C flag will be cleared to 0, otherwise if the result is positive or zero, the C flag will be set to 1.
Operation	ACC ← ACC − x
Affected flag(s)	OV, Z, AC, C, SC, CZ

SWAP [m]	Swap nibbles of Data Memory
Description	The low-order and high-order nibbles of the specified Data Memory are interchanged.
Operation	[m].3~[m].0 ↔ [m].7~[m].4
Affected flag(s)	None

SWAPA [m]	Swap nibbles of Data Memory with result in ACC
Description	The low-order and high-order nibbles of the specified Data Memory are interchanged. The result is stored in the Accumulator. The contents of the Data Memory remain unchanged.
Operation	ACC.3~ACC.0 ← [m].7~[m].4 ACC.7~ACC.4 ← [m].3~[m].0
Affected flag(s)	None

SZ [m]	Skip if Data Memory is 0
Description	The contents of the specified Data Memory are read out and then written back to the specified Data Memory again. If the contents of the specified Data Memory is 0, the following instruction is skipped. As this requires the insertion of a dummy instruction while the next instruction is fetched, it is a two cycle instruction. If the result is not 0 the program proceeds with the following instruction.
Operation	Skip if [m]=0
Affected flag(s)	None

SZA [m]	Skip if Data Memory is 0 with data movement to ACC
Description	The contents of the specified Data Memory are copied to the Accumulator. If the value is zero, the following instruction is skipped. As this requires the insertion of a dummy instruction while the next instruction is fetched, it is a two cycle instruction. If the result is not 0 the program proceeds with the following instruction.
Operation	ACC ← [m] Skip if [m]=0
Affected flag(s)	None

SZ [m].i	Skip if bit i of Data Memory is 0
Description	If bit i of the specified Data Memory is 0, the following instruction is skipped. As this requires the insertion of a dummy instruction while the next instruction is fetched, it is a two cycle instruction. If the result is not 0, the program proceeds with the following instruction.
Operation	Skip if [m].i=0
Affected flag(s)	None

TABRD [m]	Read table (specific page) to TBLH and Data Memory
Description	The low byte of the program code (specific page) addressed by the table pointer (TBLP and TBHP) is moved to the specified Data Memory and the high byte moved to TBLH.
Operation	[m] ← program code (low byte) TBLH ← program code (high byte)
Affected flag(s)	None

TABRDL [m]	Read table (last page) to TBLH and Data Memory
Description	The low byte of the program code (last page) addressed by the table pointer (TBLP) is moved to the specified Data Memory and the high byte moved to TBLH.
Operation	[m] ← program code (low byte) TBLH ← program code (high byte)
Affected flag(s)	None

ITABRD [m]	Increment table pointer low byte first and read table (specific page) to TBLH and Data Memory
Description	Increment table pointer low byte, TBLP, first and then the program code (specific page) addressed by the table pointer (TBHP and TBLP) is moved to the specified Data Memory and the high byte moved to TBLH.
Operation	[m] ← program code (low byte) TBLH ← program code (high byte)
Affected flag(s)	None

ITABRDL [m]	Increment table pointer low byte first and read table (last page) to TBLH and Data Memory
Description	Increment table pointer low byte, TBLP, first and then the low byte of the program code (last page) addressed by the table pointer (TBLP) is moved to the specified Data Memory and the high byte moved to TBLH.
Operation	[m] ← program code (low byte) TBLH ← program code (high byte)
Affected flag(s)	None

XOR A,[m]	Logical XOR Data Memory to ACC
Description	Data in the Accumulator and the specified Data Memory perform a bitwise logical XOR operation. The result is stored in the Accumulator.
Operation	ACC ← ACC ″XOR″ [m]
Affected flag(s)	Z

XORM A,[m]	Logical XOR ACC to Data Memory
Description	Data in the specified Data Memory and the Accumulator perform a bitwise logical XOR operation. The result is stored in the Data Memory.
Operation	[m] ← ACC ″XOR″ [m]
Affected flag(s)	Z

XOR A,x	Logical XOR immediate data to ACC
Description	Data in the Accumulator and the specified immediate data perform a bitwise logical XOR operation. The result is stored in the Accumulator.
Operation	ACC ← ACC ″XOR″ x
Affected flag(s)	Z

Extended Instruction Definition

The extended instructions are used to directly access the data stored in any data memory sections.

LADC A,[m]	Add Data Memory to ACC with Carry
Description	The contents of the specified Data Memory, Accumulator and the carry flag are added. The result is stored in the Accumulator.
Operation	ACC ← ACC + [m] + C
Affected flag(s)	OV, Z, AC, C, SC

LADCM A,[m]	Add ACC to Data Memory with Carry
Description	The contents of the specified Data Memory, Accumulator and the carry flag are added. The result is stored in the specified Data Memory.
Operation	[m] ← ACC + [m] + C
Affected flag(s)	OV, Z, AC, C, SC

LADD A,[m]	Add Data Memory to ACC
Description	The contents of the specified Data Memory and the Accumulator are added. The result is stored in the Accumulator.
Operation	ACC ← ACC + [m]
Affected flag(s)	OV, Z, AC, C, SC

LADDM A,[m]	Add ACC to Data Memory
Description	The contents of the specified Data Memory and the Accumulator are added. The result is stored in the specified Data Memory.
Operation	[m] ← ACC + [m]
Affected flag(s)	OV, Z, AC, C, SC

LAND A,[m]	Logical AND Data Memory to ACC
Description	Data in the Accumulator and the specified Data Memory perform a bitwise logical AND operation. The result is stored in the Accumulator.
Operation	ACC ← ACC ″AND″ [m]
Affected flag(s)	Z

LANDM A,[m]	Logical AND ACC to Data Memory
Description	Data in the specified Data Memory and the Accumulator perform a bitwise logical AND operation. The result is stored in the Data Memory.
Operation	[m] ← ACC ″AND″ [m]
Affected flag(s)	Z

LCLR [m]	Clear Data Memory
Description	Each bit of the specified Data Memory is cleared to 0.
Operation	[m] ← 00H
Affected flag(s)	None

LCLR [m].i	Clear bit of Data Memory
Description	Bit i of the specified Data Memory is cleared to 0.
Operation	[m].i ← 0
Affected flag(s)	None

LCPL [m]	Complement Data Memory
Description	Each bit of the specified Data Memory is logically complemented (1′s complement). Bits which previously contained a 1 are changed to 0 and vice versa.
Operation	$[m] \leftarrow \overline{[m]}$
Affected flag(s)	Z

LCPLA [m]	Complement Data Memory with result in ACC
Description	Each bit of the specified Data Memory is logically complemented (1′s complement). Bits which previously contained a 1 are changed to 0 and vice versa. The complemented result is stored in the Accumulator and the contents of the Data Memory remain unchanged.
Operation	$\text{ACC} \leftarrow \overline{[m]}$
Affected flag(s)	Z

LDAA [m]	Decimal-Adjust ACC for addition with result in Data Memory
Description	Convert the contents of the Accumulator value to a BCD (Binary Coded Decimal) value resulting from the previous addition of two BCD variables. If the low nibble is greater than 9 or if AC flag is set, then a value of 6 will be added to the low nibble. Otherwise the low nibble remains unchanged. If the high nibble is greater than 9 or if the C flag is set, then a value of 6 will be added to the high nibble. Essentially, the decimal conversion is performed by adding 00H, 06H, 60H or 66H depending on the Accumulator and flag conditions. Only the C flag may be affected by this instruction which indicates that if the original BCD sum is greater than 100, it allows multiple precision decimal addition.
Operation	$[m] \leftarrow \text{ACC} + 00\text{H}$ or $[m] \leftarrow \text{ACC} + 06\text{H}$ or $[m] \leftarrow \text{ACC} + 60\text{H}$ or $[m] \leftarrow \text{ACC} + 66\text{H}$
Affected flag(s)	C

LDEC [m]	Decrement Data Memory
Description	Data in the specified Data Memory is decremented by 1.
Operation	$[m] \leftarrow [m] - 1$
Affected flag(s)	Z

LDECA [m]	Decrement Data Memory with result in ACC
Description	Data in the specified Data Memory is decremented by 1. The result is stored in the Accumulator. The contents of the Data Memory remain unchanged.
Operation	$\text{ACC} \leftarrow [m] - 1$
Affected flag(s)	Z

LINC [m]	Increment Data Memory
Description	Data in the specified Data Memory is incremented by 1.
Operation	$[m] \leftarrow [m] + 1$
Affected flag(s)	Z

LINCA [m]	Increment Data Memory with result in ACC
Description	Data in the specified Data Memory is incremented by 1. The result is stored in the Accumulator. The contents of the Data Memory remain unchanged.
Operation	$\text{ACC} \leftarrow [m] + 1$
Affected flag(s)	Z

LMOV A,[m]	Move Data Memory to ACC
Description	The contents of the specified Data Memory are copied to the Accumulator.
Operation	ACC ← [m]
Affected flag(s)	None

LMOV [m],A	Move ACC to Data Memory
Description	The contents of the Accumulator are copied to the specified Data Memory.
Operation	[m] ← ACC
Affected flag(s)	None

LOR A,[m]	Logical OR Data Memory to ACC
Description	Data in the Accumulator and the specified Data Memory perform a bitwise logical OR operation. The result is stored in the Accumulator.
Operation	ACC ← ACC ″OR″ [m]
Affected flag(s)	Z

LORM A,[m]	Logical OR ACC to Data Memory
Description	Data in the specified Data Memory and the Accumulator perform a bitwise logical OR operation. The result is stored in the Data Memory.
Operation	[m] ← ACC ″OR″ [m]
Affected flag(s)	Z

LRL [m]	Rotate Data Memory left
Description	The contents of the specified Data Memory are rotated left by 1 bit with bit 7 rotated into bit 0.
Operation	[m].(i+1) ← [m].i; (i=0~6) [m].0 ← [m].7
Affected flag(s)	None

LRLA [m]	Rotate Data Memory left with result in ACC
Description	The contents of the specified Data Memory are rotated left by 1 bit with bit 7 rotated into bit 0. The rotated result is stored in the Accumulator and the contents of the Data Memory remain unchanged.

Operation	ACC.(i+1) ← [m].i; (i=0~6)
	ACC.0 ← [m].7

Affected flag(s)	None

LRLC [m]	Rotate Data Memory left through Carry
Description	The contents of the specified Data Memory and the carry flag are rotated left by 1 bit. Bit 7 replaces the Carry bit and the original carry flag is rotated into bit 0.
Operation	[m].(i+1) ← [m].i; (i=0~6) [m].0 ← C C ← [m].7
Affected flag(s)	C

LRLCA [m]	Rotate Data Memory left through Carry with result in ACC
Description	Data in the specified Data Memory and the carry flag are rotated left by 1 bit. Bit 7 replaces the Carry bit and the original carry flag is rotated into the bit 0. The rotated result is stored in the Accumulator and the contents of the Data Memory remain unchanged.

Operation	ACC.(i+1) ← [m].i; (i=0~6)
	ACC.0 ← C
	C ← [m].7

Affected flag(s)	C

LRR [m]	Rotate Data Memory right
Description	The contents of the specified Data Memory are rotated right by 1 bit with bit 0 rotated into bit 7.
Operation	[m].i ← [m].(i+1); (i=0~6) [m].7 ← [m].0
Affected flag(s)	None

LRRA [m]	Rotate Data Memory right with result in ACC
Description	Data in the specified Data Memory is rotated right by 1 bit with bit 0 rotated into bit 7. The rotated result is stored in the Accumulator and the contents of the Data Memory remain unchanged.

Operation	ACC.i ← [m].(i+1); (i=0~6)
	ACC.7 ← [m].0

Affected flag(s)	None

LRRC [m]	Rotate Data Memory right through Carry
Description	The contents of the specified Data Memory and the carry flag are rotated right by 1 bit. Bit 0 replaces the Carry bit and the original carry flag is rotated into bit 7.
Operation	[m].i ← [m].(i+1); (i=0~6) [m].7 ← C C ← [m].0
Affected flag(s)	C

LRRCA [m]	Rotate Data Memory right through Carry with result in ACC
Description	Data in the specified Data Memory and the carry flag are rotated right by 1 bit. Bit 0 replaces the Carry bit and the original carry flag is rotated into bit 7. The rotated result is stored in the Accumulator and the contents of the Data Memory remain unchanged.

Operation	ACC.i ← [m].(i+1); (i=0~6)
	ACC.7 ← C
	C ← [m].0

Affected flag(s)	C

LSBC A,[m]	Subtract Data Memory from ACC with Carry
Description	The contents of the specified Data Memory and the complement of the carry flag are subtracted from the Accumulator. The result is stored in the Accumulator. Note that if the result of subtraction is negative, the C flag will be cleared to 0, otherwise if the result is positive or zero, the C flag will be set to 1.
Operation	ACC ← ACC − [m] − $\overline{C}$
Affected flag(s)	OV, Z, AC, C, SC, CZ

LSBCM A,[m]	Subtract Data Memory from ACC with Carry and result in Data Memory
Description	The contents of the specified Data Memory and the complement of the carry flag are subtracted from the Accumulator. The result is stored in the Data Memory. Note that if the result of subtraction is negative, the C flag will be cleared to 0, otherwise if the result is positive or zero, the C flag will be set to 1.
Operation	[m] ← ACC − [m] − $\overline{C}$
Affected flag(s)	OV, Z, AC, C, SC, CZ

LSDZ [m]	Skip if decrement Data Memory is 0
Description	The contents of the specified Data Memory are first decremented by 1. If the result is 0 the following instruction is skipped. As this requires the insertion of a dummy instruction while the next instruction is fetched, it is a three cycle instruction. If the result is not 0 the program proceeds with the following instruction.
Operation	[m] ← [m] − 1 Skip if [m]=0
Affected flag(s)	None

LSDZA [m]	Skip if decrement Data Memory is zero with result in ACC
Description	The contents of the specified Data Memory are first decremented by 1. If the result is 0, the following instruction is skipped. The result is stored in the Accumulator but the specified Data Memory contents remain unchanged. As this requires the insertion of a dummy instruction while the next instruction is fetched, it is a three cycle instruction. If the result is not 0, the program proceeds with the following instruction.
Operation	ACC ← [m] − 1 Skip if ACC=0
Affected flag(s)	None

LSET [m]	Set Data Memory
Description	Each bit of the specified Data Memory is set to 1.
Operation	[m] ← FFH
Affected flag(s)	None

LSET [m].i	Set bit of Data Memory
Description	Bit i of the specified Data Memory is set to 1.
Operation	[m].i ← 1
Affected flag(s)	None

LSIZ [m]	Skip if increment Data Memory is 0
Description	The contents of the specified Data Memory are first incremented by 1. If the result is 0, the following instruction is skipped. As this requires the insertion of a dummy instruction while the next instruction is fetched, it is a three cycle instruction. If the result is not 0 the program proceeds with the following instruction.
Operation	[m] ← [m] + 1 Skip if [m]=0
Affected flag(s)	None

LSIZA [m]	Skip if increment Data Memory is zero with result in ACC
Description	The contents of the specified Data Memory are first incremented by 1. If the result is 0, the following instruction is skipped. The result is stored in the Accumulator but the specified Data Memory contents remain unchanged. As this requires the insertion of a dummy instruction while the next instruction is fetched, it is a three cycle instruction. If the result is not 0 the program proceeds with the following instruction.
Operation	ACC ← [m] + 1 Skip if ACC=0
Affected flag(s)	None

LSNZ [m].i	Skip if bit i of Data Memory is not 0
Description	If bit i of the specified Data Memory is not 0, the following instruction is skipped. As this requires the insertion of a dummy instruction while the next instruction is fetched, it is a three cycle instruction. If the result is 0 the program proceeds with the following instruction.
Operation	Skip if [m].i ≠ 0
Affected flag(s)	None

LSNZ [m]	Skip if Data Memory is not 0
Description	The contents of the specified Data Memory are read out and then written to the specified Data Memory again. If the content of the specified Data Memory is not 0, the following instruction is skipped. As this requires the insertion of a dummy instruction while the next instruction is fetched, it is a three cycle instruction. If the result is 0 the program proceeds with the following instruction.
Operation	Skip if [m] ≠ 0
Affected flag(s)	None

LSUB A,[m]	Subtract Data Memory from ACC
Description	The specified Data Memory is subtracted from the contents of the Accumulator. The result is stored in the Accumulator. Note that if the result of subtraction is negative, the C flag will be cleared to 0, otherwise if the result is positive or zero, the C flag will be set to 1.
Operation	ACC ← ACC − [m]
Affected flag(s)	OV, Z, AC, C, SC, CZ

LSUBM A,[m]	Subtract Data Memory from ACC with result in Data Memory
Description	The specified Data Memory is subtracted from the contents of the Accumulator. The result is stored in the Data Memory. Note that if the result of subtraction is negative, the C flag will be cleared to 0, otherwise if the result is positive or zero, the C flag will be set to 1.
Operation	[m] ← ACC − [m]
Affected flag(s)	OV, Z, AC, C, SC, CZ

LSWAP [m]	Swap nibbles of Data Memory
Description	The low-order and high-order nibbles of the specified Data Memory are interchanged.
Operation	[m].3~[m].0 ↔ [m].7~[m].4
Affected flag(s)	None

LSWAPA [m]	Swap nibbles of Data Memory with result in ACC
Description	The low-order and high-order nibbles of the specified Data Memory are interchanged. The result is stored in the Accumulator. The contents of the Data Memory remain unchanged.
Operation	ACC.3~ACC.0 ← [m].7~[m].4 ACC.7~ACC.4 ← [m].3~[m].0
Affected flag(s)	None

LSZ [m]	Skip if Data Memory is 0
Description	The contents of the specified Data Memory are read out and then written to the specified Data Memory again. If the contents of the specified Data Memory is 0, the following instruction is skipped. As this requires the insertion of a dummy instruction while the next instruction is fetched, it is a three cycle instruction. If the result is not 0 the program proceeds with the following instruction.
Operation	Skip if [m]=0
Affected flag(s)	None

LSZA [m]	Skip if Data Memory is 0 with data movement to ACC
Description	The contents of the specified Data Memory are copied to the Accumulator. If the value is zero, the following instruction is skipped. As this requires the insertion of a dummy instruction while the next instruction is fetched, it is a three cycle instruction. If the result is not 0 the program proceeds with the following instruction.
Operation	ACC ← [m] Skip if [m]=0
Affected flag(s)	None

LSZ [m].i	Skip if bit i of Data Memory is 0
Description	If bit i of the specified Data Memory is 0, the following instruction is skipped. As this requires the insertion of a dummy instruction while the next instruction is fetched, it is a three cycle instruction. If the result is not 0, the program proceeds with the following instruction.
Operation	Skip if [m].i=0
Affected flag(s)	None

LTABRD [m]	Read table (specific page) to TBLH and Data Memory
Description	The low byte of the program code (specific page) addressed by the table pointer (TBHP and TBLP) is moved to the specified Data Memory and the high byte moved to TBLH.
Operation	[m] ← program code (low byte) TBLH ← program code (high byte)
Affected flag(s)	None

LTABRDL [m]	Read table (last page) to TBLH and Data Memory
Description	The low byte of the program code (last page) addressed by the table pointer (TBLP) is moved to the specified Data Memory and the high byte moved to TBLH.
Operation	[m] ← program code (low byte) TBLH ← program code (high byte)
Affected flag(s)	None

LITABRD [m]	Increment table pointer low byte first and read table (specific page) to TBLH and Data Memory
Description	Increment table pointer low byte, TBLP, first and then the program code (specific page) addressed by the table pointer (TBHP and TBLP) is moved to the specified Data Memory and the high byte moved to TBLH.
Operation	[m] ← program code (low byte) TBLH ← program code (high byte)
Affected flag(s)	None

LITABRDL [m]	Increment table pointer low byte first and read table (last page) to TBLH and Data Memory
Description	Increment table pointer low byte, TBLP, first and then the low byte of the program code (last page) addressed by the table pointer (TBLP) is moved to the specified Data Memory and the high byte moved to TBLH.
Operation	[m] ← program code (low byte) TBLH ← program code (high byte)
Affected flag(s)	None

LXOR A,[m]	Logical XOR Data Memory to ACC
Description	Data in the Accumulator and the specified Data Memory perform a bitwise logical XOR operation. The result is stored in the Accumulator.
Operation	ACC ← ACC ″XOR″ [m]
Affected flag(s)	Z

LXORM A,[m]	Logical XOR ACC to Data Memory
Description	Data in the specified Data Memory and the Accumulator perform a bitwise logical XOR operation. The result is stored in the Data Memory.
Operation	[m] ← ACC ″XOR″ [m]
Affected flag(s)	Z

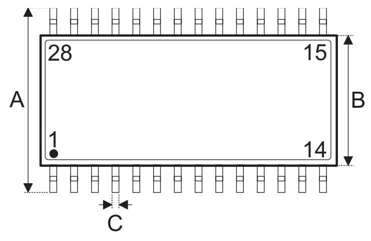
Package Information

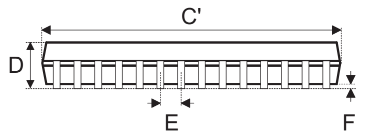
Note that the package information provided here is for consultation purposes only. As this information may be updated at regular intervals users are reminded to consult the Holtek website for the latest version of the Package/Carton Information.

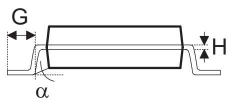
Additional supplementary information with regard to packaging is listed below. Click on the relevant section to be transferred to the relevant website page.

- Package Information (include Outline Dimensions, Product Tape and Reel Specifications)
- The Operation Instruction of Packing Materials
- Carton information

28-pin SSOP (150mil) Outline Dimensions



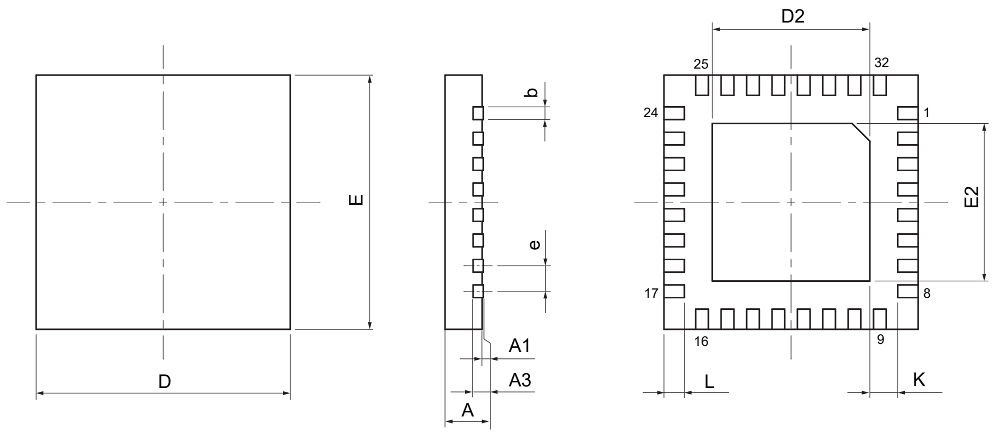




Symbol	Dimensions in inch		
	Min.	Nom.	Max.
A		0.236 BSC	
B		0.154 BSC	
C	0.008	—	0.012
C’		0.390 BSC	
D	—	—	0.069
E		0.025 BSC	
F	0.004	—	0.010
G	0.016	—	0.050
H	0.004	—	0.010
α	0°	—	8°

Symbol	Dimensions in mm		
	Min.	Nom.	Max.
A		6.00 BSC	
B		3.90 BSC	
C	0.20	—	0.30
C’		9.90 BSC	
D	—	—	1.75
E		0.635 BSC	
F	0.10	—	0.25
G	0.41	—	1.27
H	0.10	—	0.25
α	0°	—	8°

SAW Type 32-pin QFN (4mm×4mm×0.75mm) Outline Dimensions



Symbol	Dimensions in inch		
	Min.	Nom.	Max.
A	0.028	0.030	0.031
A1	0.000	0.001	0.002
A3	0.008 REF		
b	0.006	0.008	0.010
D	0.157 BSC		
E	0.157 BSC		
e	0.016 BSC		
D2	0.100	—	0.108
E2	0.100	—	0.108
L	0.010	—	0.018
K	0.008	—	—

Symbol	Dimensions in mm		
	Min.	Nom.	Max.
A	0.70	0.75	0.80
A1	0.00	0.02	0.05
A3	0.203 REF		
b	0.15	0.20	0.25
D	4.00 BSC		
E	4.00 BSC		
e	0.40 BSC		
D2	2.55	—	2.75
E2	2.55	—	2.75
L	0.25	—	0.45
K	0.20	—	—