SNP70032 Programming Guide V1.1

========== CONTENTS ==========

VERSION HISTORY

Version	Date	Description
V1.0	2012/07/18	First Release Version
V1.1	2013/01/31	1. Modify Descroption of I2S Register 2. Remove UART related word

1. General Description

The SNP70032 is a 16-bit DSP processor with SONiX OID decoder function. It runs at 48MHz with 48MIPS high performance processing speed. SNP70032 built-in 64KW ROM, 16KW Program RAM and 16KW Working RAM. However, SNP70032 allows run code at external SPI flash.

Peripherals embedded in the SNP70032 include OID sensor interface, XtraROM interface, SD/MMC controller, USB device, MSP, SPI master/slave interface, PWM Output, Audio ADC and Stereo DAC, I2S and SAR ADC.

2. Features

2.1. DSP

- Built-in 16-bit DSP core
- 48 MIPS CPU Performance under 48MHz, 1 Clock per 1 Instruction.
- Clock Type:
  - 48MHz for system clock
  - 32KHz for system clock
- High Speed Clock Source (pumping from 12MHz → 48MHz by PLL circuit)
  - 12MHz crystal oscillator
- Low Speed Clock Source:
  - 32KHz is be generated from 12MHz (The actually is 31250Hz)
- Operation Mode:
  - Normal mode (hi-speed clock enable)
  - Slow mode (hi-speed clock enable, PLL disable, slow-speed clock disable)
  - Watch mode (chip entry power-down mode and wake-up per 0.5/1 sec automatically)
  - Power down mode (both hi-speed and low-speed clock disable)
- 3 for 16-bit Timers, 1 Watch Dog Timer, 1 RTC
  - Timers with Individual pre-scalar and auto-reload function
  - Event Counter (Combine Timer and Input Pin P0.0~P0.2)
  - Watch Dog Timer (WDT) with 0.25/0.5/1/2-sec period
  - RTC with 0.5/1 sec period
- Interrupt Sources
  - 1 for ADC, 3 for Timers, 1 for RTC, 1 for SPI, 1 for AD, 1 for DA, 1 for I2S, 1 for MSP
  - 3 for External (P0.0~P0.2), 1 for USB, 1 for SD Card, 4 for DMA
- 32KW Internal Program RAM
- Total 16KW Internal RAM memory configuration for Program and working RAM
  - Mode 0: 8KW Program RAM + 8KW Working RAM
  - Mode 1: 4KW Program RAM + 12KW Working RAM
  - Mode 2: 12KW Program RAM + 4KW Working RAM (default)
  - Mode 3: 16KW Working RAM
- Support Barrel Shifter and 16×16 to 32-bit multiplier
- DMA provided (USB/XtraROM/SDC/SPI)

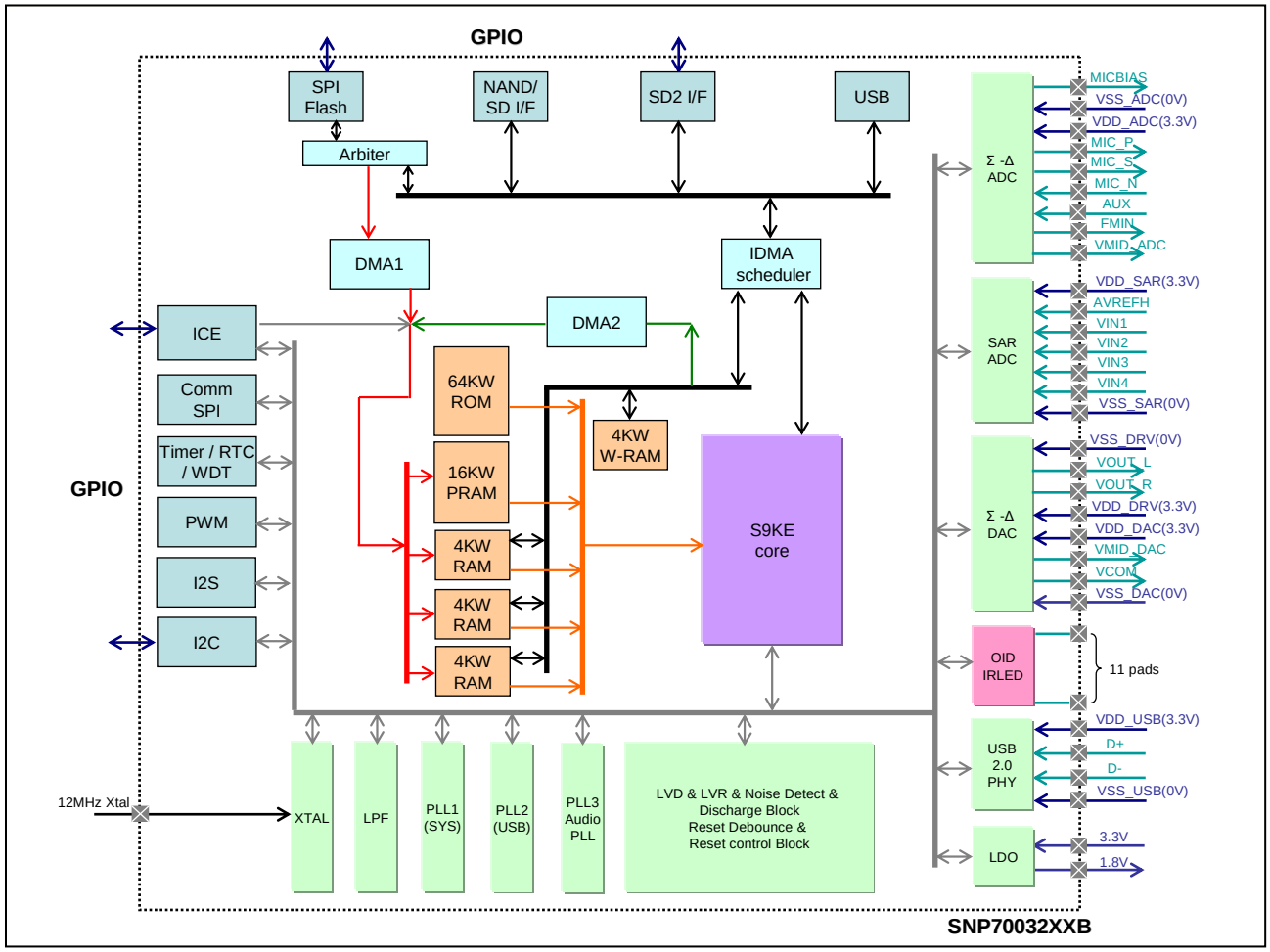
2.2. Peripherals and on-chip resources

- Power Supply: 2.7V ~ 3.6V
- Low voltage detection
- Low power reset
- I/O Pins:
  - 45 I/O pins (P0.0~P0.13, P1.0~P1.15, P3.11~P3.15, P4.0~P4.9)
- 2xSPI (Serial Peripheral Interface)
  - Support 1/2/4 bit mode
  - Maximum 48MHz Clock frequency
  - Support run program on SPI Flash
- MSP
  - Master/Slave mode
  - Support General call address

- PWM Output
  - Provide 4 I/O pins capable for output 256 levels PWM pulse.
- 16-bit Sigma Delta Stereo DAC + Class AB
  - Can drive the L/R Channel Earphone.
  - SNR 90dB
- I2S Interfce
  - Master mode
  - Supports Left/Right justified format
- 16-bit Sigma-Delta ADC with AGC function
  - Differental / Single-end Microphone input option
  - Build-in Microphone Bias Voltage support
- 10-bit SAR ADC
  - 4 input channel (AIN0 ~ AIN3)
  - Input Range 0V~3.3V(VDDA)
  - No missing code : 8 bit
- NAND FLASH controller (just support read function)
- SD/MMC controller
  - Supports SD Card1.0/2.0 commands
  - Supports 1-bit / 4-bit / SPI mode
- USB 2.0 High Speed device
  - Supports Storage and HID Class
  - Supports Control/Bulk/Interrupt interface
- OID Interface
  - Supports dot pattern format : OID_Code_v2
  - Embedded 16 bit-DSP for sensor control and image pattern recognition
  - Light source timing control

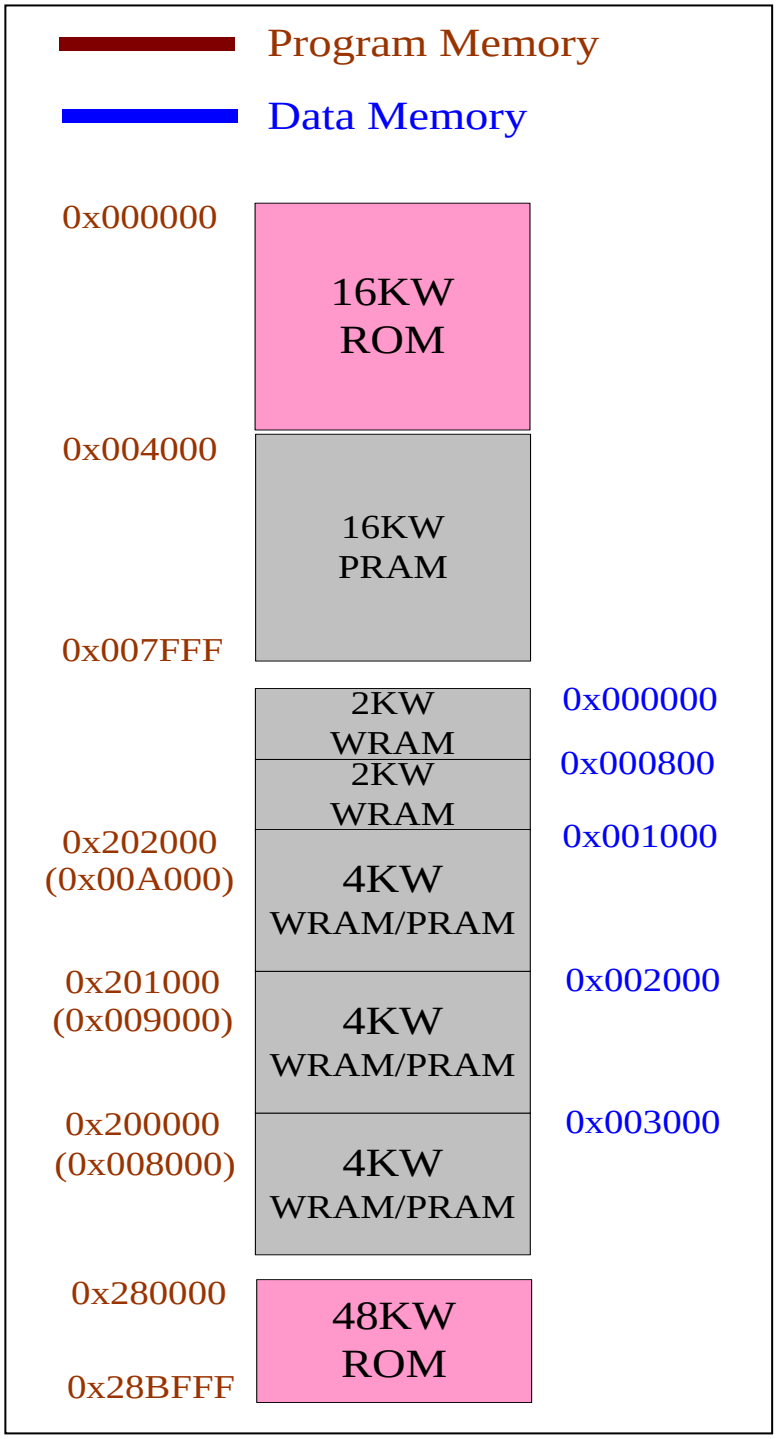
3. Architecture Description

3.1. System Block Diagram



3.2. DSP Memory Map

In SNP70032, not only has 64KW Internal Program ROM size but also has totally 16KW Program RAM size and 16KW User RAM. And the 16KW User RAM can be configured as **Program RAM (PRAM)** or **Working RAM (WRAM)** for different applications. Every 4KW RAM has a dedicated program memory address or an dedicated data memory address.



The following is the system configuration detailed description.

SYSCONF (0x007C) : initial value = 0x0022

Bit	Function	Description	R/W	Reset Value
15:8	--	Reseved	--	All “0”
7	SD2_USE	SD2 Use or not 0: No Use 1: Use	R/W	0
6	SD2_SEL	SD2 Pinout Select 0: SD2 Pinout at P3 1: SD2 Pinout at P1	R/W	0
5:2	--	Reserved	--	1000
1:0	MEM_CONF	Memory Configuration Setting	R/W	10

MEM_CONF : default is select Mode 2

	Mode	Program RAM Size	Working RAM Size
00	Mode 0	8KW	8KW
01	Mode 1	4KW	12KW
10	Mode 2	12KW	4KW
11	Mode 3	--	16KW

3.2.1. Internal working RAM

Address Range	Size (word)	Usage	DSP	DMA2	IDMA
0x0000 ~0x07FF	2K	Working RAM	R/W	R	R/W
0x0800 ~0x0FFF	2K	Working RAM	R/W	R	R/W
0x1000 ~0x1FFF	4K	Working RAM	R/W	R	R/W
0x2000 ~0x2FFF	4K	Working RAM	R/W	R	R/W
0x3000 ~0x3FFF	4K	Working RAM	R/W	R	R/W

MMIO Mapping

0xE000 ~0xE00A	11	USB Control Register	R/W		
0xE100 ~0xE16F	112		R/W		
0xE200 ~0xE283	132		R/W		
0xE300	1		R/W		
0xE310	1		R/W		
0xE800 ~0xE9FF	512	USB Ping-Pong Buffer	R/W		
0xF000 ~0xF0FF	256	NAND Flash / SD Card Register	R/W		

0xF100 ~0xF1FF	256	DMA SCHEDULE Register	R/W		
0xF200 ~0xF206					
0xF300 ~0xF303	4	Fractional PLL Register	R/W		
0xF800 ~0xF8FF	256	SPIF Register	R/W		
0xFE27 ~0xFE2C	6	DMA1 Control Register	R/W		

3.2.2. Program Memory

In SNP70032, Program Memory include Program ROM and Program RAM. The following is the detailed description and usage.

- **Internal Program Memory**

Address Range	Size (Words)	Usage	DSP	DMA1	DMA2	IDMA	ICE
0x00000000 ~ 0x00003FFF	16K	Program ROM	R	--	--	--	--
0x00280000 ~ 0x0028BFFF	48K		R	--	--	--	--
0x00200000 ~ 0x00202FFF (**1)	16K	Program RAM	R	W	W	--	W

(**1) : 0x00200000 ~ 0x00202FFF is mapping to PRAM address 0x00008000 ~ 0x0000AFFF

- **External Program Memory**

Address Range	Size (Words)	Usage	DSP	DMA1	IDMA	ICE
0x00400000 ~ 0x00BFFFFF	8M	CS1 SPI Flash	R/W	R	R/W	R/W

3.2.3. Special Registers (IO SPACE) of DSP

There are 128 words special registers to define peripheral/system register. All defined registers are listed here.

The first 64 words can be pushed to or popped from data ram.

Address	Name	Function Description	Default
	High Byte / Low Byte		Bit15 ~ Bit0
0x0000	SSF	System Status Flag Register	0000,0000,0000,0000
0x0001	SCR	System Control Register	0000,0000,0000,0010
0x0002	Ix0	Indirect Addressing Register x0	0000,0000,0000,0000
0x0003	Ix1	Indirect Addressing Register x1	0000,0000,0000,0000
0x0004	Im00	Ix0/Iy0 Modifier Register 0	0000,0000,0000,0000
0x0005	Im01	Ix0/Iy0 Modifier Register 1	0000,0000,0000,0000
0x0006	Im02	Ix0/Iy0 Modifier Register 2	0000,0000,0000,0001
0x0007	Im03	Ix0/Iy0 Modifier Register 3	1111,1111,1111,1111
0x0008	Im10	Ix1/Iy1 Modifier Register 0	0000,0000,0000,0000
0x0009	Im11	Ix1/Iy1 Modifier Register 1	0000,0000,0000,0000
0x000a	Im12	Ix1/Iy1 Modifier Register 2	0000,0000,0000,0001
0x000b	Im13	Ix1/Iy1 Modifier Register 3	1111,1111,1111,1111
0x000c	OPM_CTRL	Operation mode control	0000,0000,0000,0011
0x000d	RAMBk / 0	RAM Bank Selector (for direct addressing)	1111,1111,0000,0000
0x000e	Ix0Bk	Ix0 Bank Selector	0000,0000,0000,0000
0x000f	Ix1Bk	Ix1 Bank Selector	0000,0000,0000,0000
0x0010	T0	Timer0	0000,0111,0000,0000
0x0011	T1	Timer1	0000,0111,0000,0000
0x0012	T2	Timer2	0000,0111,0000,0000
0x0013	Iy0	Indirect Addressing Register y0	0000,0000,0000,0000
0x0014	Iy1	Indirect Addressing Register y1	0000,0000,0000,0000
0x0015	PCH	Program Counter High	0000,0000,0000,0000
0x0016	PCL	Program Counter Low	0000,0000,0000,0000
0x0017	MMR	MAC Mode Register	0000,0000,0000,0000
0x0018	Sp	Stack Pointer	0000,0001,0000,0000
0x0019	Sign-Extension / MR2	MR2	0000,0000,0000,0000
0x001a	Iy0BK	Iy0 ROM Bank Selector	0000,0000,0000,0000
0x001b	Iy1BK	Iy1 ROM Bank Selector	0000,0000,0000,0000
0x001c	Ix0BKRAM	Ix0 RAM Bank Selector	0000,0000,0000,0000
0x001d	Ix1BKRAM	Ix1 RAM Bank Selector	0000,0000,0000,0000
0x001e	Ix2	Indirect Addressing Register x2	0000,0000,0000,0000
0x001f	Iy2	Indirect Addressing Register y2	0000,0000,0000,0000
0x0020	INTEN	Interrupt channel Enable	0000,0000,0000,0000
0x0021	INTRQ	Interrupt channel Request	0000,0000,0000,0000
0x0022	INTPR	Interrupt channel Priority	0000,0000,0000,0000
0x0023	INTCR	Interrupt channel Clear Request	0000,0000,0000,0000
0x0024	PCR1	Peripheral Control Register 1	0000,0000,0010,0000
0x0025	OPM_CTRL1	Operation mode control 1	0000,0000,0000,0000
0x0026	ADC_FIFOSTATUS	ADC FIFO Status register	0000,0010,0000,0000
0x0027			
0x0028	ADC_DATA	ADC Result Register	0000,0000,0000,0000

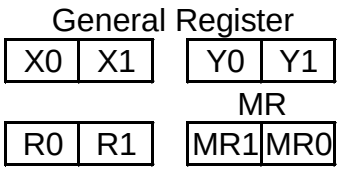
0x0029	WDT	Watch Dog Timer	1000,0000,0000,0000
0x002a	ADC_SET1	Control ADC signal 1	0000,0000,0000,0000
0x002b	ADC_SET2	Control ADC signal 2	0000,0000,0000,0000
0x002c	ImxL	Ix2 Modifier Register Linear Mode	0000,0000,0000,0000
0x002d	ImxC	Ix2 Modifier Register Circular Mode	0000,0000,0000,0000
0x002e	ImyL	Iy2 Modifier Register Linear Mode	0000,0000,0000,0000
0x002f	ImyC	Iy2 Modifier Register Circular Mode	0000,0000,0000,0000
0x0030	P0WKUPEN	P0 wake up enable register	0000,0000,0000,0000
0x0031	P1WKUPEN	P1 wake up enable register	0000,0000,0000,0000
0x0032	INTEN2	Interrupt channel Enable	0000,0000,0000,0000
0x0033	INTRQ2	Interrupt channel Request Flag	0000,0000,0000,0000
0x0034	INTPR2	Interrupt channel Priority	0000,0000,0000,0000
0x0035	INTCR2	Interrupt channel Clear Request	0000,0000,0000,0000
0x0036	IBx	Ix2 Base Address	0000,0000,0000,0000
0x0037	ILx	Ix2 Length	0000,0000,0000,0000
0x0038	IBy	Iy2 Base Address	0000,0000,0000,0000
0x0039	ILy	Iy2 Length	0000,0000,0000,0000
0x003a	IOSW	I/O Byte Swap Register	0000,0000,0000,0000
0x003b	SP1	Stack pointer 2 for OS	0000,0010,0000,0000
0x003c	IOSW2	I/O Byte Swap Register 2	0000,0000,0000,0000
0x003d	EVENT	Timer Event control register	0000,0000,0000,0000
0x003e	ShIdx	Shift amount of Index shift	0000,0000,0000,0000
0x003f	ShV2	Shift result register	0000,0000,0000,0000
0x0040	T1CNTV	Counter Value of T1 (Read only)	0000,0000,0000,0000
0x0041			
0x0042			
0x0043			
0x0044			
0x0045	T0CNT	Timer0 Counter Register	0000,0000,0000,0000
0x0046	T1CNT	Timer1 Counter Register	0000,0000,0000,0000
0x0047	T0CNTV	Counter Value of T0 (Read only)	0000,0000,0000,0000
0x0048	INTEC	Interrupt Edge Control register	0000,0000,0000,0000
0x0049	DAC_SET1	Control DAC signal 1	0000,0000,0000,0000
0x004a	DAC_SET2	Control DAC signal 2	0000,0000,0000,0000
0x004b	DAC_FIFOSTATUS	DAC FIFO Status register	0000,0010,0000,0000
0x004c	T2CNT	Timer2 Counter Register	0000,0000,0000,0000
0x004d	EVENT0CNT	EVENT0 Count Value	0000,0000,0000,0000
0x004e	EVENT1CNT	EVENT1 Count Value	0000,0000,0000,0000
0x004f	EVENT2CNT	EVENT2 Count Value	0000,0000,0000,0000
0x0050	I2SCTRL	I2S control register	0000,0000,0000,0000
0x0051	PWM0	PWM control of P0.3	0000,0000,0000,0000
0x0052	PWM1	PWM control of P0.4	0000,0000,0000,0000
0x0053	PWM2	PWM control of P0.5	0000,0000,0000,0000
0x0054	PWM3	PWM control of P0.6	0000,0000,0000,0000
0x0055	DAOL	Left Channel DA output register	0000,0000,0000,0000
0x0056	DAOR	Right Channel DA output register	0000,0000,0000,0000
0x0057	SPIDADA0	SPI data buffer	0000,0000,XXXX,XXXX
0x0058	SPIDADA1	SPI data buffer	0000,0000,XXXX,XXXX

0x0059	SPIDADA2	SPI data buffer	0000,0000,XXXX,XXXX
0x005A	SPIDADA3	SPI data buffer	0000,0000,XXXX,XXXX
0x005B	SPIDADA4	SPI data buffer	0000,0000,XXXX,XXXX
0x005C	SPIDADA5	SPI data buffer	0000,0000,XXXX,XXXX
0x005D	SPICTRL	SPI Control register	0000,0000,0000,0000
0x005E	SPICSC	SPI Chip Select Control	0000,0000,0000,0000
0x005F	SPITRANSFER	SPI Bit transfer Control	0000,0000,0000,0000
0x0060			
0x0061	SPIBR	SPI Bard Rate register	0000,0000,0000,0000
0x0062	MSPSTAT	MSP Status Register	0000,0000,0000,0000
0x0063	MSPM1	MSP Mode Register 1	0000,0000,0000,0000
0x0064	MSPM2	MSP Mode Register 2	0000,0000,0000,0000
0x0065	MSPBUF	MSP Data Buffer Register	0000,0000,0000,0000
0x0066	MSPADR	MSP Address Register	0000,0000,0000,0000
0x0067	CHIP_ID	Chip ID Register (read only)	0000,1111,1111,1111
0x0068	P0En	I/O Port0 Enable	0000,0000,0000,0000
0x0069	P0	I/O Port0	1111,1111,1111,1111
0x006A	P0M	I/O Port0 Direction	0000,0000,0000,0000
0x006B	P0PH	I/O Port0 Pull-High	1111,1111,1111,1111
0x006C	P1En	I/O Port1 Enable	0000,0000,0000,0000
0x006D	P1	I/O Port1	1111,1111,1111,1111
0x006E	P1M	I/O Port1 Direction	0000,0000,0000,0000
0x006F	P1PH	I/O Port1 Pull-High	1111,1111,1111,1111
0x0070			
0x0071			
0x0072			
0x0073			
0x0074	P3En	I/O Port3 Enable	0000,0000,0000,0000
0x0075	P3	I/O Port3	1111,1000,0000,0000
0x0076	P3M	I/O Port3 Direction	0000,0000,0000,0000
0x0077	P3PH	I/O Port3 Pull-High	1111,1111,1111,1111
0x0078	P4En	I/O Port4 Enable	0000,0000,0000,0000
0x0079	P4	I/O Port4	XXXX,1111,1111,1111
0x007A	P4M	I/O Port4 Direction	0000,0000,0000,0000
0x007B	P4PH	I/O Port4 Pull-High	1111,1111,1111,1111
0x007C	SYSCONF	Internal system configuration	0000,0000,0010,0010
0x007D	ADP	SAR ADC input pin control	0000,0000,0000,0000
0x007E	ADM	SAR ADC Mode control	0000,0000,0000,0000
0x007F	ADR	SAR ADC Result	0000,0000,0000,0000

3.3. DSP Private Functions

3.3.1. General Registers

The DSP each have their own eight 16-bit general registers, namely X0, X1, Y0, Y1, R0, R1, MR0 and MR1. These registers can be used as general purpose of working buffer, compute results from ALU/ MAC/SHIFT, or access ROM's and RAM's data, etc..



3.3.2. System Status Flag

The DSP each have a single System Status Flag register.

SSF (0x0000) : initial value = 0x0000

Bit	Function	Description	R/W	Reset Value
15:6	--	Reserved	--	All "0"
5	IOF	Index Register Overflow Flag (for Data Address Generation) 0: If no index registers (Ix0/Ix1/Iy0/Iy1) overflow after Index modification instruction 1: If any index registers (Ix0/Ix1/Iy0/Iy1) overflow after Index modification instruction	R	0
4	MOF	MAC Overflow Flag (only for MAC) 0: If the MAC doesn't overflow 1: If the MAC overflows	R	0
3	AOF	Arithmetic Overflow Flag (only for ALU) 0: If the ALU doesn't overflow 1: If the ALU overflows	R	0
2	CF	Carry Flag 0: If executed arithmetic addition without occurring carry signal or executed arithmetic subtraction with borrowing signal or executed shift instruction with shifting out logic 0 1: If executed arithmetic addition with occurring carry signal or executed arithmetic subtraction without borrowing signal or executed shift instruction with shifting out logic 1	R	0
1	NF	Negative Flag 0: The ALU output is not negative 1: The ALU output is negative	R	0
0	ZF	Zero Flag 0: The result doesn't equal zero after operation 1: The result equals zero after operation	R	0

3.3.3. MMR (MAC Mode Register)

MMR (0x0017) : initial value = 0x0000

Bit	Function	Description	R/W	Reset Value
15	MacSatEn	MAC Saturation enable bit 0: Disable, 1: Enable	R/W	0
14	AluSatEn	ALU Saturation enable bit 0: Disable, 1: Enable	R/W	0
13	MacE	Mac Extension Instruction 0: Normal, 1: Enable Double Fetch Instruction	R/W	0
12:0	--	Reserved	--	0

3.3.4. Addressing Mode

The DSP provides three addressing modes to access memory data, including immediate addressing, direct addressing and indirect addressing.

3.3.5. Immediate Addressing

One single word instruction can handle 8-bit immediate data. Three different types of addressing mode of manipulating the immediate data are provided: 1. High Byte 2. Low Byte 3. Store Low Byte and Clear High Byte.

Example:

```
X0.h = 0x3a          ; (1 cycle) Low byte is not changed
Y0.l = 0x2f          ; (1 cycle) High byte is not changed
R0 = 0               ; (1 cycle) Low byte is zero and High byte is cleared
R0 = 0x5a            ; (1 cycle) Low byte is “0x5a” and High byte is cleared as (“0x00”)
```

Note:
The compiler recognizes 16-bit data also. However, user’s program will be divided into two equivalent instructions, as follows:

```
X0 = 0x5F3A          ; X0.h = 0x5F
                     ; X0.L = 0x3A
```

3.3.6. Direct Addressing

The direct addressing mode uses address number to access memory location. There is one bank register (RAMBk) to select the RAM bank. Each bank contains 512-word RAM in direct mode. **Please note that direct addressing mode is only used for internal RAM area.** If users want to access external RAM, indirect addressing mode should be used.

RAMBk (0x005D) initial value = 0xFF00

Bit	Function	Description	R/W	Reset Value
15:8	--	Reserved	--	All “1”
7:1	Bank Value	Select Bank number 0000000 => Bank0 : 1111111 => Bank127	R/W	0
0	--	Reserved	--	Always “0”

Example:

```
.Data                   // Define RAM's name and size
/*name     ds    size*/

ORG 0x1000
data1   ds   1          ; Define data1's size is one word, location is 0x1000
data2   ds   16         ; Define data2's size is ten words, location is 0x1001~0x1011

.Code                   // Program segment
X0 = 0x10
RAMBk = X0              ; Set RAM bank = 0x10
X0 = 0x5A
data1 = X0              ; data1{RAM(0x1000)} ← 0x5A
data2+2 = X0            ; data2+2{RAM(0x1003)} ← 0x5A
```

3.3.7. Indirect Addressing

There are 4 general index registers for next data address generation while executing memory access instructions (both RAM and ROM). The next address will be one of the four following results:

1. current address.(Im00/Im10)
2. current address+1. (Im02/Im12)
3.current address-1(Im03/Im13).
4. current address + a modify register (Im01/Im11).).

In the cycle (SRAM take 1, ROM takes 2), the processor finishes all the operations (reading SRAM and updating the index register).

DAG						
RAM Bank 8-bit	ROM Bank 8-bit	Index Register 16-bit	Modify Register 16-bit			
Ix0BKRAM	Ix0BK	Ix0	Im00 (0)	Im02 (1)	Im01	Im03 (-1)
	Iy0BK	Iy0	Im00 (0)	Im02 (1)	Im01	Im03 (-1)

Ix1BKRAM	Ix1BK	Ix1	Im10 (0)	Im12 (1)	Im11	Im13 (-1)
	Iy1BK	Iy1	Im10 (0)	Im12 (1)	Im11	Im13 (-1)
MacE RAM Bank 8-bit	ROM	MacE Index Register 16-bit	MacE Modify Register 16-bit		MacE Base address	MacE Length
8'b0	Not Supported	Ix2	ImxL	ImxC	IBx	ILx
8'b0	Not Supported	Iy2	ImyL	ImyC	IBy	ILy

The DAG contains 16 registers: four index (Ix0, Ix1, Iy0, Iy1) registers and four bank selector (Ix0Bk, Ix1Bk, Iy0Bk, Iy1Bk) registers for storing pointers, eight modify (Im00, …, Im03, Im10, …, Im13) registers for updating pointers. Ix0 and Ix1 can be used by all index operation. Iy0 and Iy1 can only be used when RAM/ROM accessed. Im0x are dedicated to Ix0/Iy0, and Im1x are dedicated to Ix1/Iy1.

The "MacE", Mac Extension Instruction, supports double-fetched and circular buffer. Ix2 and Iy2 for double-fetched source index. The buffer control supports linear mode and circular mode. For circular mode, base address register IBx and IBy are included for buffer base address. Length register ILx and ILy are included for buffer length.

RAM Access

The indirect addressing mode uses RAM(Ix0,m), RAM(Iy0,m), or RAM(Iy1,m) instruction to read/write RAM data. The 8-bit RAM bank register (Ix0BKRAM) together with 16-bit index registers (Ix0) form a 24-bit address for RAM accessing. However, for Iy0 and Iy1 index registers, the RAM bank is fixed to zero by hardware. So Iy0 and Iy1 index registers are only used to 16-bit address for RAM access.

Example:

```
        .DATA                   // Define RAM's name and size

        ORG 0x1010
        Matrix ds   16          ; Define Matrix's size is 16 words, , location is 0x1010~0x101F

        .Code                   // Program segment
              :
        X0 = 0
        Ix0BKRAM = X0           ; Clear RAM Bank of Ix0
        X0 = 0
        Ix0 = (Matrix)          ; Load Matrix's initial location into Ix0 => 0x1010
        R1 = Ix0
        Y0 = 16
        R0 = R1 + Y0
Loop:
        RAM(Ix0,1) = X0         ; RAM(Ix0)=0, Ix0 = Ix0+1
        X0 = X0 + 1
        R1 = Ix0
        R1 = R1 – R0
        Jne   Loop
        Nop
              :
```

ROM Access

For the ROM access function, the bank selector registers (Ix0Bk, Ix1BK, Iy0Bk, Iy1Bk) point to the highest 8-bit address and the index registers (Ix0, Ix1, Iy0, Iy1) point to the lowest 16-bit address. Each ROM bank contains 64K-word ROM size in indirect mode. Note that Iy0Bk and Iy0 can support auto increase/decrease across ROM banks.

Example 1:

```
/*Load ROM Data*/
X0 = data_table$h
Ix0Bk = X0          ; Ix0Bk = data_table's high address=>0x001
Ix0 = data_table    ; Ix0 = data_table's low address=>0x5001
R0 = ROM(Ix0)       ; R0 ← 1,     => Ix0 = 0x5001
      :
X0 = data_table$h
Ix0Bk = X0          ; Ix0Bk = data_table's high address=>0x001
Ix0 = data_table    ; Ix0 = data_table's low address=>0x5001
R0 = ROM(Ix0,1)     ; R0 ← 1, Ix0 = Ix0 + 1 => Ix0 = 0x5002
      :
X0 = data_table$h
Ix0Bk = X0          ; Ix0Bk = data_table's high address=>0x001
Ix0 = data_table    ; Ix0 = data_table's low address=>0x5001
R0 = ROM(Ix0,-1)    ; R0 ← 1, Ix0 = Ix0 – 1 => Ix0 = 0x5000
      :
X0 = data_table$h
Ix0Bk = X0          ; Ix0Bk = data_table's high address=>0x001
Ix0 = data_table    ; Ix0 = data_table's low address=>0x5001
R0 = ROM(Ix0,m)     ; R0 ← 1, Ix0 = Ix0 + lm01
      :
ORG 0x0015001
data_table:
DW   1,2,3,4,5,6,7
```

Example 2:

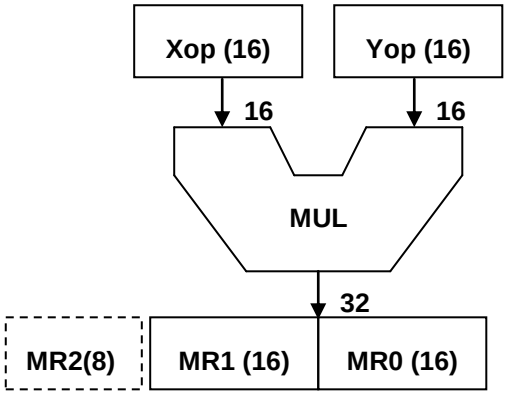
```
X0 = 0x001
Iy0Bk = X0          ; Iy0Bk = 0x001
X0 = 0xFFFF
Iy0 = X0            ; Iy0 = 0xFFFF
R0 = ROM(Iy0,1)     ; R0 ← 1, Iy0 = Iy0 + 1 => Iy0 = 0x0000, Iy0Bk = 0x002
      :
ORG 0x001FFFF
data_table:
DW   1,2,3,4,5,6,7
```

3.3.8. Arithmetic Logic Unit (ALU)

The DSP each have their own arithmetic/logic unit (ALU). The ALU provides a standard set of arithmetic and logical functions. The arithmetic part includes add, subtract, increment and decrement. The logic functions include AND, OR, XOR, NOT, Bit set, bit clear, bit AND, and bit XOR. The instruction set reference in the last section specifies clearly the syntax of ALU instructions.

3.3.9. Multiplier

One 16×16 to 32-bit multiplier is built into the chip. There are 2 operands which come from X and Y operands, separately. Both operands can be treated in fractional or integer representation, but only signed format is allowed. The multiplier provides a simple multiply and multiply-accumulate (MAC) commands. Result is written into MR only. MR is 32-bit.



Standard Functions

X*Y(IS/FS) Multiply X and Y operands.
MR+X*Y(IS/FS) Multiply X and Y operands and add result to MR register.
MR-X*Y(IS/FS) Multiply X and Y operands and subtract result from MR register.

Multiple Functions

Multifunction operations take advantage of the inherent parallelism of this chip's architecture by providing combinations of RAM's read and MAC in a single cycle.

X*Y(IS/FS), X/Y=RAM(IxN) Multiply X and Y operands and load RAM's data
or X/Y=RAM(IxN,1)
or X/Y=RAM(IxN,-1)
or X/Y=RAM(IxN,M)

MR+X*Y(IS/FS), X/Y=RAM(IxN) Multiply X and Y operands and add result to MR register
or X/Y=RAM(IxN,1) and load RAM's data
or X/Y=RAM(IxN,-1)
or X/Y=RAM(IxN,M)
MR-X*Y(IS/FS), X/Y=RAM(IxN) Multiply X and Y operands and subtract result from MR
or X/Y=RAM(IxN,1 register and load RAM's data
or X/Y=RAM(IxN,-1)
or X/Y=RAM(IxN,M)

This chip provides two modes for the MAC: fractional (FS) mode for fractional representation (1.15), and integer (IS) mode for integers (16.0).

Multiple Functions with Double-Fetched

When MMR bit13 MacE enable, the MAC extension instruction with double-fetched will be executed instead of single-fetched instruction. DSP read dual port ram twice for the next operator X and Y and MAC in a single cycle.

X*Y(IS/FS), X=RAM(Ix2,ImXL), Y=RAM(Iy2,ImYL)
X*Y(IS/FS), X=RAM(Ix2,ImXL), Y=RAM(Iy2,ImYC)
X*Y(IS/FS), X=RAM(Ix2,ImXC), Y=RAM(Iy2,ImYL)
X*Y(IS/FS), X=RAM(Ix2,ImXC), Y=RAM(Iy2,ImYC)

MR+X*Y(IS/FS), X=RAM(Ix2,ImXL), Y=RAM(Iy2,ImYL)
MR+X*Y(IS/FS), X=RAM(Ix2,ImXL), Y=RAM(Iy2,ImYC)
MR+X*Y(IS/FS), X=RAM(Ix2,ImXC), Y=RAM(Iy2,ImYL)
MR+X*Y(IS/FS), X=RAM(Ix2,ImXC), Y=RAM(Iy2,ImYC)

MR-X*Y(IS/FS), X=RAM(Ix2,ImXL), Y=RAM(Iy2,ImYL)
MR-X*Y(IS/FS), X=RAM(Ix2,ImXL), Y=RAM(Iy2,ImYC)
MR-X*Y(IS/FS), X=RAM(Ix2,ImXC), Y=RAM(Iy2,ImYL)
MR-X*Y(IS/FS), X=RAM(Ix2,ImXC), Y=RAM(Iy2,ImYC)

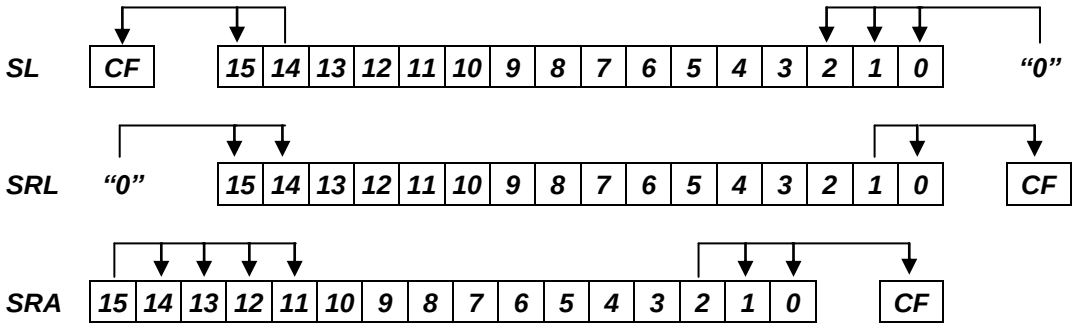
MR2

MR2 is an 8-bit register. MR2 acts as the 8-bit MSB of MAC function and makes the result of MAC 40 bit data. With the help of MR2, large numbers of overflow check procedure routines can be omitted during a series of MAC instructions execute.

Note: X*Y(IS/FS): MR2=Sign Extension (MR1,MR0)

3.3.10. Shifter/Barrel Shifter

This shifter provides an eight bit of shifting functions and a shift with index register shift instruction. These include arithmetic shift and logical shift.



The other shift instruction is index shift instrucion, the number of shift bit of shift instrution is according to ShIdx

ShIdx (0x003e) : initial value = 0x0000

Bit	Function	Description	R/W	Reset Value
15:4	ShOV	Shift overflow. If ShOV doesn't equal "0", shift overflow happened, 16 bits will all be shifted	R	All "0"
3:0	Shldx	Shift index, can shift 0 ~ 15 bits	R/W	0

ShV2 (0x003f) : initial value = 0x0000

Bit	Function	Description	R/W	Reset Value
15:0	ShV2	High byte of shift left instruction, and Low byte of shift right instruction	R/W	All “0”

Example :

```
void main(void)
{
    __asm
    {
        R0 = 8                          // Set Shldx = 8
        IO(0x3E) = R0

        R0 = 0x1234                     // R0 = R0 >> 8
        R0 = SRL. Idx R0                // R0 = 0x0012
        R1 = IO(0x3F)                   // R1 = 0x3400

        R0 = 0x1234                     // R0 = R0 << 8
        R0 = SL. Idx R0                 // R0 = 0x3400
        R1 = IO(0x3F)                   // R1 = 0x0012
    }

}
```

3.3.11. Saturation Control

The resulting registers have a saturation capability which sets the result register to the maximum positive or negative value if an overflow or underflow has occurred. If users set MMR.15 is “1”, the result of MAC/ALU will perform saturation control.

MMR initial value = 0x0000

Bit	Function	Description	R/W	Reset Value
15	MacSatEn	MAC Saturation enable bit 0: Disable, 1: Enable	R/W	0
14	AluSatEn	ALU Saturation enable bit 0: Disable, 1: Enable	R/W	0
13	MacE	Mac Extension Instruction 0: Normal, 1: Enable Double Fetch Instruction	R/W	0
12:0	--	Reserved	--	All “0”

ALU Saturation condition:

SSF.AV	SSF.CF	Result
1	0	16'h7fff
1	1	16'h8000

MAC Saturation condition:

SSF.MV	MR2[7]	Result
1	0	40'h00_7fff_ffff
1	1	40'hff_8000_0000

3.3.12. IOSW

SNP70032 provide a hardware swap function. If a 16 bit value is written into the IOSW register, the hardware will auto exchage the high byte and low byte value.

IOSW (0x003a) : initial value = 0x0000

Bit	Function	Description	R/W	Reset Value
15:8	IOSW[15:8]	The High byte value	R/W	All "0"
7:0	IOSW[7:0]	The Low byte value	R/W	All "0"

IOSW2 (0x003c) : initial value = 0x0000

Bit	Function	Description	R/W	Reset Value
15:8	IOSW2[15:8]	IOSW[15:8] backup	R/W	All "0"
7:0	IOSW2[7:0]	IOSW[7:0] backup	R/W	All "0"

Example :

```
        R0 = 0x1234          // R0 = 0x1234
        IOSW = R0            // Swap R0 value
        R0 = IOSW            // R0 = 0x3421

// Interrupt ISR
        PUSH   IOSW2
               :
               :
        POP    IOSW2
```

3.3.13. Program Counter

The program counter (PC) is a 24-bit binary counter separated into 8-bit in PCH and low-word 16-bits in PCL. It's responsible for pointing to a location in order to fetch an instruction for execution. Normally, the program counter is automatically incremented with each instruction during program execution. And the program is sequential executing. There are 8 exceptions changing the execution flow.

Call/Callff:

Push (current address + 1) into stack. Jump to the target address. Program stack pointers (SP, SP1) are 16 bits. Users must define the RAM word location as its access space.

Ret/Retff:

pop return address from stack, and then jump to it. Update program stack pointer.

Jmp:

Jump to target address. (+/- 2K word address)

Jmpff:

Jump to target address. (over +/- 2K word address)

Interrupt:

If an interrupt request occurs, the SP will +2 and update interrupt enable (GIE) bit. Program branches to the corresponding entry routine of interrupt service routine.

Reti:

Pop return address from the stack (SP), return to the address, and re-open interrupt enable (GIE) bit.

PCH (0x0015) : initial value = 0x0000

Bit	Function	Description	R/W	Reset Value
15:8	--	The High byte value	--	All "0"
7:0	PCH	The Program Counter high word [23:16]	R	All "0"

PCL (0x0016) : initial value = 0x0000

Bit	Function	Description	R/W	Reset Value
15:0	PCL	The Program Counter low word [15:0]	R	All "0"

3.3.14. Conditional Jump

If the condition is matched, the program jumps to target address.

Jeq/Jz	:	If zero
Jne/Jnz	:	If not zero
Jgt	:	If greater than
Jge	:	If greater than or equal
Jlt	:	If less than
Jle	:	If less than or equal
Jav	:	If ALU overflow
Jnav	:	If not ALU overflow
Jac	:	If ALU carry out
Jnac	:	If not ALU carry out
Jmr0s	:	If mr0 is signed
Jmr0ns	:	If mr0 is not signed
Jmv	:	If MAC is overflow
Jnmv	:	If MAC is not overflow

```
Jixv        ;       If index register is overflow
```

3.3.15. Stack Pointer

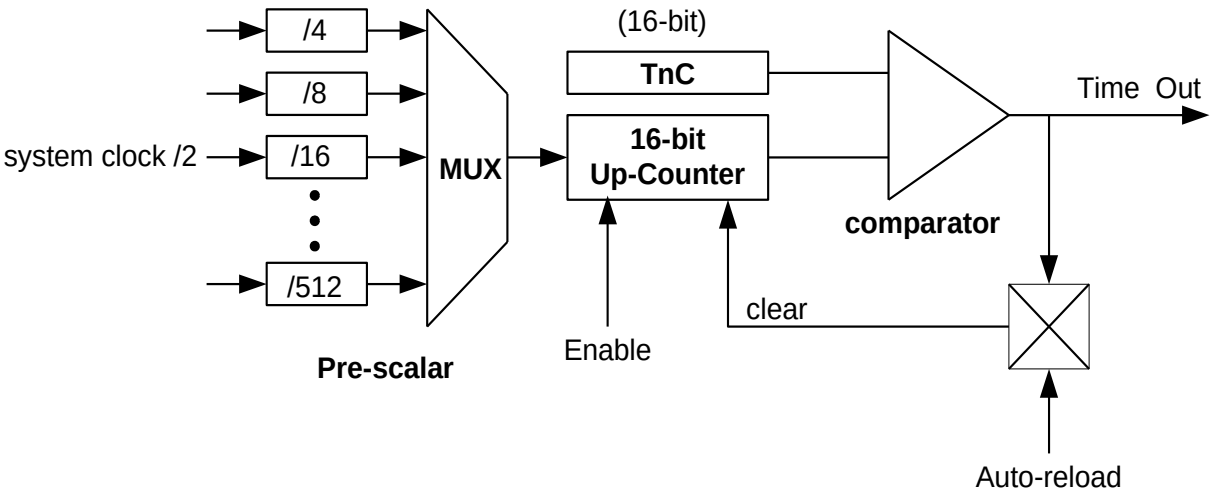
Usually, the user allocates a stack in SRAM to record the program return address and some temporary data. The 16-bit registers (SP) point to the starting address of stack. SP will be increased by 1 after PUSH and decreased by 1 after POP instruction automatically. CALL/RET will increase/decrease SP by 2.

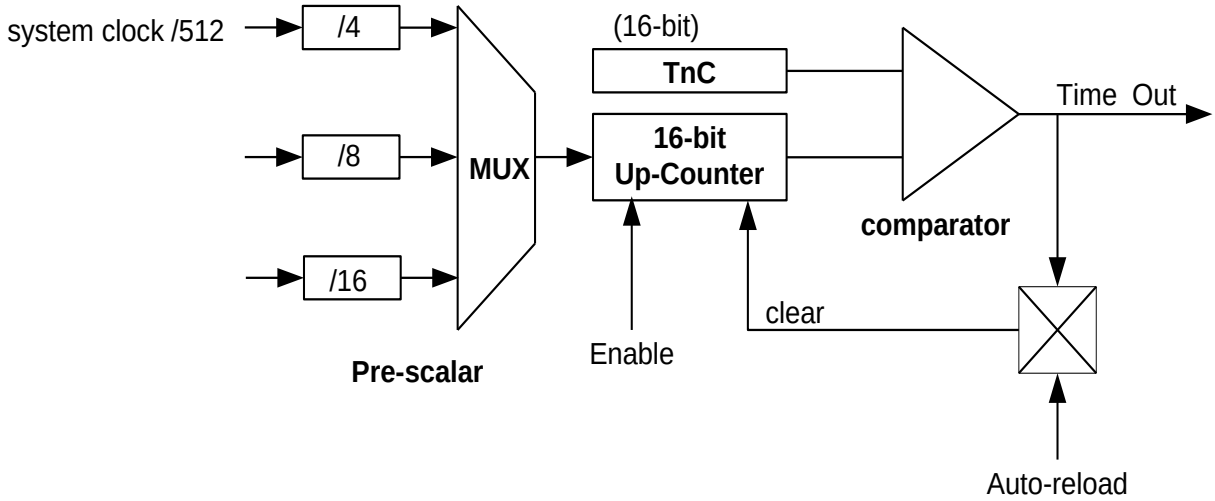
SP (0x0018) : initial value = 0x0100

Bit	Function	Description	R/W	Reset Value
15:0	SP	Stack Pointer value	R/W	All “0”

3.3.16. Timer

The DSP have 3 sets timers. The timers are 16-bit binary up-counting timer with auto-reload function. The 3-bit pre-scalar is used as a clock division. If a successful event occurs (counting value = setting value), it will continue counting and issue a time out signal to trigger Timer-n interrupt to request interrupt service.





T0 (0x0010) / T1 (0x0011) / T2 (0x0012) : initial value = 0x0700

Bit	Function	Description	R/W	Reset Value
15:13	--	Reserved	--	3'b0
12	E	Timer enable bit 0: disable, 1: enable	R/W	0
11	A	Auto-reload control bit 0: None auto-reload, 1: Auto-reload	R/W	0
10:8	Pre-Scalar	Timer internal clock selects bit 000: /4, 001: /8, 010: /16, 011: /32, …, 111: /512 *(If S = 1, just only /4, /8, /16 three mode for used)*	R/W	3'b1
7	S	Select Timer internal clock from system clock 0: system clock/2, 1: system clock/512	R/W	0
6:0	--	Reserved	--	7'b0

T0CNT (0x0045) / T1CNT (0x0046) / T2CNT (0x0047) : initial value = 0x0000

Bit	Function	Description	R/W	Reset Value
15:0	TnC (n=0, 1, 2)	Counting register. This register is used to store the counter value	R	All "0"

EVENT (0x003d) : initial value = 0x0000

Bit	Function	Description	R/W	Reset Value
15:6	--	Reserved	--	All "0"
5	T2_mode	Select Timer2 mode 0: Timer mode, 1: Event mode	R/W	0
4	Event2 trig mode	Select Event2 trigger mode 0: Count P0.2 falling edge trigger 1: Count P0.2 raising edge trigger then save to EVENT2CNT	R/W	0
3	T1_mode	Select Timer1 mode 0: Timer mode, 1: Event mode	R/W	0
2	Event1 trig mode	Select Event1 trigger mode 0: Count P0.1 falling edge trigger 1: Count P0.1 raising edge trigger then save to EVENT1CNT	R/W	0
1	T0_mode	Select Timer0 mode 0: Timer mode, 1: Event mode	R/W	0

0	Event0 trig mode	Select Event0 trigger mode 0: Count P0.0 falling edge trigger 1: Count P0.0 raising edge trigger then save to EVENT0CNT	R/W	0

EVENT0CNT (0x004d) / EVENT1CNT (0x004e) / EVENT2CNT (0x004f) : initial value = 0x0000

Bit	Function	Description	R/W	Reset Value
15:0	EVENTnC (n=0, 1, 2)	EVENT counting register	R	All “0”

Example :

```
void main(void)
{
    _setSR(SFR_T0, _getSR(SFR_T0) | 0x1F00);          // Set T0 Auto reload, Pre scalar 256(/256)
                                                      // Enable T0
    _setSR(SFR_T0CNT, 0x0FFF);                        // Set T0 Counter value.

    _setSR(SFR_INTEN, _getSR(SFR_INTEN) | 0x0080);    // T0 interrupt Enable
    _setSR(SFR_INTEN, _getSR(SFR_INTEN) | 0x8000);    // Global interrupt Enable
}

void __interrupt [0x0018] T0INT(void)   // T0 interrupt
{
    _IObitSET(SFR_INTCR, 7);                          // Clear T0 interrupt request.
}
```

3.3.17. Real-time clock

To realize the watch function, **0.5s ~ 1s** RTC (real time clock) is built in the chip. The clock source of RTC comes from the system clock.

RTC_EN = 1 will enable the real time clock counting. And INTEN.RTCIEN = 1 will activate the RTC interrupt when RTC occurs 0.5 ~ 1s overflow.

PCR1 (0x0024) : initial value = 0x0020

Bit	Function	Description	R/W	Reset Value
15	LPF_BYPASS	12MHz if bypass from LPF 0: No bypass, 1: Always bypass	R/W	0
14	PAD_DRVC	Select P1.1 ~ P1.5 driving current 0: 16mA, 1: 12mA	R/W	0
13:12	--	Reserved	--	0

11	LVD_PDC	LVD Power Down Control 0: LVD ON, 1: LVD OFF	R/W	0
10	LVD_DISC	LVD Discharge 0: No discharge, 1: Discharge	R/W	0
9	BAT_DET	Battery Voltage Dectect (refer to LVD_SEL bits) 0: Battery voltage >= 2.8V 1: Battery voltage < 2.8V	R	0
8	**RTC_TS**	**RTC Time Select** **0: 0.5S, 1: 1S**	**R/W**	**0**
7	**RTC_SS**	**RTC Source Select** **0: From 12M/48MHz Clock, 1: From 32K OSC**	**R/W**	**0**
6:5	BAT_DET_LEVEL	Adjust Battery Voltage Detect Level 00: 2.5V, 01: 2.8V, 10: 2.9V	R/W	01
4	--	Reserved	--	0
3	**RTC_EN**	**RTC function Enable** **0: disable, 1: enable**	**R/W**	**0**
2	LVD_2ms	0: RESET_ON, 1: RESET_OFF	R/W	0
1:0	--	Reserved	--	00

Example :

```
void main(void)
{
    _IObitSET(SFR_PCR1, 8);                 // Set RTC time select is 1 second
    _IObitSET(SFR_PCR1, 7);                 // Set RTC source from 32KHz XTAL
    _IObitSET(SFR_PCR1, 3);                 // Enable RTC

    _IObitSET(SFR_INTEN2, 12);              // RTC interrupt Enable
    _IObitSET(SFR_INTEN, 15);               // Global interrupt Enable
}

void __interrupt [0x0058] RTCINT(void)// RTC interrupt
{
    _IObitSET(SFR_INTCR2, 12);              // Clear RTC interrupt request
}
```

3.3.18. Watch Dog Timer

The DSP has a single Watchdog Timer (WDT). The WDT is a free-running counter that generates a reset signal if the counter overflows. The WDT is a kind of safety device to prevent the program from being in the out of control state because of external factors such as power supply noise, software dead loop, or runaway program. ***The WDT is always enabled in the SNP70032, so users must add instructions at the proper location of the program to***

clear WDT clear flag (WDT_CLR).

WDT (0x0029) : initial value = 0x8000

Bit	Function	Description	R/W	Reset Value
15	WDT_EN	Watch Dog Timer Enable 0: disable, 1: enable	R/W	1
14	WDT_CLR	Watch Dog Timer Clear bit 0: No Clear WDT counter, 1: Clear WDT counter	R/W	0
13	WDT_TR	Watch Dog Timer Reset. When chip reset by WDT, this bit will be 1	R	0
12:2	--	Reserved	--	0
1:0	WDT_OPS	Watch Dog Timer Overflow Period Select 00: 0.25S, 01: 0.5S, 10: 1S, 11: 2S	R/W	00

Users are suggested to add the instructions to clear WDT counter in main function as the following example.

Example:

```
void main(void)
{
    _setSR(SFR_WDT, _getSR(SFR_WDT) | (SET_BIT1 | SET_BIT0));    // Set WDT_OPS is 2 second

    while(1)
    {
        _setSR(SFR_WDT, _getSR(SFR_ WDT) | SET_BIT14);    // Clear WDT counter
    }
}
```

3.3.19. Interrupt of DSP

At the moment when DSP enters the interrupt service routine, the GIE bit (in INTEN) will be cleared to "0“ for blanking other interrupt. During this stage, other enabled interrupt sources still issue their requests however the requests are queued in INTRQ. GIE will be restored to “1” when DSP exits ISR. Other valid interrupts in the queue will be granted and served immediately.

Interrupt Vector	Priority	Entry Location	Descriptions
Reset	x	0x000000	Reset
reserved		0x000010	
AD	4	0x000014	AD FIFO interrupt
T0	5	0x000018	T0 overflow
P0.0	6	0x00001C	Falling/Raising edge of P0.0
T1	7	0x000020	T1 overflow
P0.1	8	0x000024	Falling/Raising edge of P0.1
T2	9	0x000028	T2 overflow
P0.2	10	0x00002C	Falling/Raising edge of P0.2
reserved		0x000030	
DA	11	0x000034	DA FIFO interrupt
SPI	3	0x000038	SPI Interrupt
MSP(I2C)	2	0x00003C	MSP Interrupt
I2S	1	0x000040	I2S Interrupt
reserved		0x000044	
reserved		0x000048	
reserved		0x00004C	
USB	12	0x000050	USB Interrupt
reserved		0x000054	
RTC	13	0x000058	RTC overflow
SD	14	0x00005C	SD Card Interrupt
reserved		0x000060	
DMA_SD_RW	15	0x000064	DMA_SD_R & DMA_SD_W Interrupt
SAR_AD	16	0x000068	SAR ADC interrupt
DMA_SD2_RW	17	0x00006C	DMA_SD2_R & DMA_SD2_W Interrupt
reserved		0x000070	
reserved		0x000074	
DMA_DEV_RW	18	0x000078	DMA_DEV_R & DMA_DEV_W Interrupt

Note:

1. **To execute interrupt service routine (ISR) properly, users must place ISR in entry label of interrupt.**
2. **When any interrupt has happened, the system will jump to entry label location to execute user’s ISR.**
3. **If the interrupt is not used, users must put “RETI” instruction in entry label of interrupt.**

INTEC (0x0048) : initial value = 0x0000

Bit	Function	Description	R/W	Reset Value
15:3	--	Reserved	--	All “0”

2	P02IEC	P0.2 Edge control bit 0: Falling edge, 1: Raising edge	R/W	0
1	P01IEC	P0.1 Edge control bit 0: Falling edge, 1: Raising edge	R/W	0
0	P00IEC	P0.0 Edge control bit 0: Falling edge, 1: Raising edge	R/W	0

- **Interrupt Enable Register**

INTEN (0x0020) initial value = 0x0000

Bit	Function	Description	R/W	Reset Value
15	GIE	Golbal Interrput Enable bit 0: disable, 1: enable	R/W	0
14:13	--	Reserved	--	0
12	I2S	I2S Interrupt Enable control bit 0: disable, 1: enable	R/W	0
11	MSP	MSP Interrupt Enable control bit 0: disable, 1: enable	R/W	0
10	SPI	SPI Interrupt Enable control bit 0: disable, 1: enable	R/W	0
9	--	Reserved	--	0
8	ADC	ADC FIFO Interrupt Enable contol bit 0: disable, 1: enable	R/W	0
7	T0	Timer0 Interrupt Enable control bit 0: disable, 1: enable	R/W	0
6	P00	P0.0 Interrupt Enable control bit 0: disable, 1: enable	R/W	0
5	T1	Timer1 Interrupt Enable control bit 0: disable, 1: enable	R/W	0
4	P01	P0.1 Interrupt Enable control bit 0: disable, 1: enable	R/W	0
3	T2	Timer2 Interrupt Enable control bit 0: disable, 1: enable	R/W	0
2	P02	P0.2 Interrupt Enable control bit 0: disable, 1: enable	R/W	0
1	--	Reserved	--	0
0	DAC	DAC FIFO Interrupt Enable contol bit 0: disable, 1: enable	R/W	0

INTEN2 (0x0032) : initial value = 0x0000

Bit	Function	Description	R/W	Reset Value
15	--	Reserved	--	0
14	USB	USB Interrupt Enable control bit 0: disable, 1: enable	R/W	0
13	--	Reserved	--	0
12	RTC	RTC Interrupt Enable control bit 0: disable, 1: enable	R/W	0
11	SD	SD Card Interrupt Enable control bit 0: disable, 1: enable	R/W	0
10	--	Reserved	--	0
9	DMA_SD_RW	SD Card DMA Write to / Read from RAM Interrupt Enable contol bit 0: disable, 1: enable	R/W	0
8	SAR_ADC	SAR ADC Interrupt Enable control bit 0: disable, 1: enable	R/W	0
7	DMA_SD2_RW	SD Card 2 (SD2) DMA Write to / Read from RAM Interrupt Enable contol bit 0: disable, 1: enable	R/W	0
6:5	--	Reserved	--	0
4	DMA_DEV_RW	Device USB DMA Write to / Read from RAM Interrupt Enable control bit 0: disable, 1: enable	R/W	0
3:0	--	Reserved	--	0

- **Interrupt Request Register**

INTRQ (0x0021) : initial value = 0x0000

Bit	Function	Description	R/W	Reset Value
15:13	--	Reserved	--	0
12	I2S	I2S Interrupt Request status 0: not requested, 1: requested	R	0
11	MSP	MSP Interrupt Request status 0: not requested, 1: requested	R	0
10	SPI	SPI Interrupt Request status 0: not requested, 1: requested	R	0
9	--	Reserved	--	0

8	ADC	ADC FIFO Interrupt Request status 0: not requested, 1: requested	R	0
7	T0	Timer0 Interrupt Request status 0: not requested, 1: requested	R	0
6	P00	P0.0 Interrupt Request status 0: not requested, 1: requested	R	0
5	T1	Timer1 Interrupt Request status 0: not requested, 1: requested	R	0
4	P01	P0.1 Interrupt Request status 0: not requested, 1: requested	R	0
3	T2	Timer2 Interrupt Request status 0: not requested, 1: requested	R	0
2	P02	P0.2 Interrupt Request status 0: not requested, 1: requested	R	0
1	--	Reserved	--	0
0	DAC	DAC FIFO Interrupt Request status 0: not requested, 1: requested	R	0

INTRQ2 (0x0033) : initial value = 0x0000

Bit	Function	Description	R/W	Reset Value
15	--	Reserved	--	0
14	USB	USB Interrupt Request status 0: not requested, 1: requested	R	0
13	--	Reserved	--	0
12	RTC	RTC Interrupt Request status 0: not requested, 1: requested	R	0
11	SD	SD Card Interrupt Request status 0: not requested, 1: requested	R	0
10	--	Reserved	--	0
9	DMA_SD_RW	SD Card DMA Write to / Read from RAM Interrupt Request status 0: not requested, 1: requested	R	0
8	SAR_ADC	SAR ADC Interrupt Request status 0: not requested, 1: requested	R	0
7	DMA_SD2_RW	SD Card 2 (SD2) DMA Write to / Read from RAM Interrupt Request status 0: not requested, 1: requested	R	0
6:5	--	Reserved	--	0
4	DMA_DEV_RW	Device USB DMA Write to / Read from RAM	R	0

		Interrupt Request status 0: not requested, 1: requested		
3:0	--	Reserved	--	0

- **Interrupt Clear Request Register**

INTCR (0x0023) : initial value = 0x0000

Bit	Function	Description	R/W	Reset Value
15:13	--	Reserved	--	0
12	I2S	I2S Interrupt Request bit clear 1: Clear request	R/W	0
11	MSP	MSP Interrupt Request bit clear 1: Clear request	R/W	0
10	SPI	SPI Interrupt Request bit clear 1: Clear request	R/W	0
9	--	Reserved	--	0
8	ADC	ADC FIFO Interrupt Request bit clear 1: Clear request	R/W	0
7	T0	Timer0 Interrupt Request bit clear 1: Clear request	R/W	0
6	P00	P0.0 Interrupt Request bit clear 1: Clear request	R/W	0
5	T1	Timer1 Interrupt Request bit clear 1: Clear request	R/W	0
4	P01	P0.1 Interrupt Request bit clear 1: Clear request	R/W	0
3	T2	Timer2 Interrupt Request bit clear 1: Clear request	R/W	0
2	P02	P0.2 Interrupt Request bit clear 1: Clear request	R/W	0
1	--	Reserved	--	0
0	DAC	DAC FIFO Interrupt Request bit clear 1: Clear request	R/W	0

INTCR2 (0x0035) : initial value = 0x0000

Bit	Function	Description	R/W	Reset Value
15	--	Reserved	--	0
14	USB	USB Interrupt Request bit clear 1: Clear request	R/W	0

13	--	Reserved	--	0
12	RTC	RTC Interrupt Request bit clear 1: Clear request	R/W	0
11	SD	SD Card Interrupt Request bit clear 1: Clear request	R/W	0
10	--	Reserved	--	0
9	DMA_SD_RW	SD Card DMA Write to / Read from RAM Interrupt Request bit clear 1: Clear request	R/W	0
8	SAR_ADC	SAR ADC Interrupt Request bit clear 1: Clear request	R/W	0
7	DMA_SD2_RW	SD Card 2 (SD2) DMA Write to / Read from RAM Interrupt Request bit clear 1: Clear request	R/W	0
6:5	--	Reserved	--	0
4	DMA_DEV_RW	Device USB DMA Write to / Read from RAM Interrupt Request bit clear 1: Clear request	R/W	0
3:0	--	Reserved	--	0

NOTE:

1. ***Be sure to set each INTCR flag to 1 in each interrupt service routine.***
2. ***H/W will reset the INTCR flag to 0 after clearing the INTRQ flag.***

Example :

```
void __interrupt [0x0028] T2 (void)        // User entry label of T2 interrupt
{
        _setSR(SFR_INTCR, SET_BIT3);        // Set bit 3 of INTCR to 1 (Enalbe T2 interrupt clear request)
}
```

- **Interrupt Priority and Interrupt Priority Register**

INTPR (0x0022) : initial value = 0x0000

Bit	Function	Description	R/W	Reset Value
15:13	--	Reserved	--	0
12	I2S	I2S Interrupt Priority bit 0: low priority, 1: high priority	R/W	0
11	MSP	MSP Interrupt Priority bit 0: low priority, 1: high priority	R/W	0
10	SPI	SPI Interrupt Priority bit	R/W	0

		0: low priority, 1: high priority		
9	--	Reserved	--	0
8	ADC	ADC FIFO Interrupt Priority bit 0: low priority, 1: high priority	R/W	0
7	T0	Timer0 Interrupt Priority bit 0: low priority, 1: high priority	R/W	0
6	P00	P0.0 Interrupt Priority bit 0: low priority, 1: high priority	R/W	0
5	T1	Timer1 Interrupt Priority bit 0: low priority, 1: high priority	R/W	0
4	P01	P0.1 Interrupt Priority bit 0: low priority, 1: high priority	R/W	0
3	T2	Timer2 Interrupt Priority bit 0: low priority, 1: high priority	R/W	0
2	P02	P0.2 Interrupt Priority bit 0: low priority, 1: high priority	R/W	0
1	--	Reserved	--	0
0	DAC	DAC FIFO Interrupt Priority bit 0: low priority, 1: high priority	R/W	0

INTPR2 (0x0034) initial value = 0x0000

Bit	Function	Description	R/W	Reset Value
15	--	Reserved	--	0
14	USB	USB Interrupt Priority bit 0: low priority, 1: high priority	R/W	0
13	--	Reserved	--	0
12	RTC	RTC Interrupt Priority bit 0: low priority, 1: high priority	R/W	0
11	SD	SD Card Interrupt Priority bit 0: low priority, 1: high priority	R/W	0
10	--	Reserved	--	0
9	DMA_SD_RW	SD Card DMA Write to / Read from RAM Interrupt Priority bit 0: low priority, 1: high priority	R/W	0
8	SAR_ADC	SAR ADC Interrupt Priority bit 0: low priority, 1: high priority	R/W	0
7	DMA_SD2_RW	SD Card 2 (SD2) DMA Write to / Read from RAM Interrupt Priority bit 0: low priority, 1: high priority	R/W	0

6:5	--	Reserved	--	0
4	DMA_DEV_RW	Device USB DMA Write to / Read from RAM Interrupt Priority bit 0: low priority, 1: high priority	R/W	0
3:0	--	Reserved	--	0

When two or more interrupt sources issue requests at the same time, the DSP will process the one with the highest priority and suspend the others until the highest one is finished. *An implicit priority sequence is:* ***I2S > MSP > SPI > AD_INT > T0 > P0.0 > T1 > P0.1 > T2 > P0.2 > DA_INT > USB > RTC > SD > DMA_SD_RW > SAR_AD > DMA_SD2_RW > DMA_DEV_RW*** INTPR helps to reorder the sequence. If a bit of INTPR is 1, then the corresponding interrupt source will be moved up to a higher priority.

3.3.20. DMA Transfers

This chip provides Direct Memory Access (DMA) for fast data transfer. The registers are defined at Memory Mapping addresses. The following describes DMA transfer paths for applications.

SD Card / SD Card 2 to WRAM

Reg_DMA_SD_W_ADDR (0xF100) : initial value = 0x0000

Bit	Function	Description	R/W	Reset Value
15:0	DMA_SD_W_ ADDR[15:0]	WRAM Address	R/W	0

Reg_DMA_SD_W_LEN (0xF101) : initial value = 0x0000

Bit	Function	Description	R/W	Reset Value
15:0	DMA_SD_W_ LEN[15:0]	DMA Data length [15:0]	R/W	0

Reg_DMA_SD_W_CTRL (0xF102) : initial value = 0x0000

Bit	Function	Description	R/W	Reset Value
15	DMA_SD_W_ TRG	Trigger to transfer data from SD to RAM, hardware auto clear after finish 1: Trigger DMA start	R/W	0
14:10	--	Reserved	--	0
9	DMA_SD_W_ PRAM	DMA destination address is WRAM or PRAM 0: WRAM, 1: PRAM	R/W	0
8	DMA_SD_W_RS	WRAM address select 0: 0~4KW, 1: 4~16KW	R/W	0
7:2	--	Reserved	--	0
1:0	DMA_SD_W_LE N[17:16]	DMA Data length [17:16]	R/W	0

Reg_DMA_SD2_W_ADDR (0xF136) : initial value = 0x0000

Bit	Function	Description	R/W	Reset Value
15:0	DMA_SD2_W_ ADDR[15:0]	WRAM Address	R/W	0

Reg_DMA_SD2_W_LEN (0xF137) : initial value = 0x0000

Bit	Function	Description	R/W	Reset Value
15:0	DMA_SD2_W_ LEN[15:0]	DMA Data length [15:0]	R/W	0

Reg_DMA_SD2_W_CTRL (0xF138) : initial value = 0x0000

Bit	Function	Description	R/W	Reset Value
15	DMA_SD2_W_	Trigger to transfer data from SD2 to RAM,	R/W	0

	TRG	hardware auto clear after finish 1: Trigger DMA start		
14:10	--	Reserved	--	0
9	DMA_SD2_W_PRAM	DMA destination address is WRAM or PRAM 0: WRAM, 1: PRAM	R/W	0
8	DMA_SD2_W_RS	WRAM address select 0: 0~4KW, 1: 4~16KW	R/W	0
7:2	--	Reserved	--	0
1:0	DMA_SD2_W_LEN[17:16]	DMA Data length [17:16]	R/W	0

WRAM to SD Card

Reg_DMA_SD_R_ADDR (0xF103) : initial value = 0x0000

Bit	Function	Description	R/W	Reset Value
15:0	DMA_SD_R_ADDR[15:0]	WRAM Address	R/W	0

Reg_DMA_SD_R_LEN (0xF104) : initial value = 0x0000

Bit	Function	Description	R/W	Reset Value
15:0	DMA_SD_R_LEN[15:0]	DMA Data length [15:0]	R/W	0

Reg_DMA_SD_R_CTRL (0xF105) : initial value = 0x0000

Bit	Function	Description	R/W	Reset Value
15	DMA_SD_R_TRG	Trigger to transfer data from RAM to SD, hardware auto clear after finish 1: Trigger DMA start	R/W	0
14:9	--	Reserved	--	0
8	DMA_SD_R_RS	WRAM address select 0: 0~4KW, 1: 4~16KW	R/W	0
7:2	--	Reserved	--	0
1:0	DMA_SD_R_LEN[17:16]	DMA Data length [17:16]	R/W	0

Reg_DMA_SD2_R_ADDR (0xF139) : initial value = 0x0000

Bit	Function	Description	R/W	Reset Value
15:0	DMA_SD2_R_ADDR[15:0]	WRAM Address	R/W	0

Reg_DMA_SD2_R_LEN (0xF13A) : initial value = 0x0000

Bit	Function	Description	R/W	Reset Value
15:0	DMA_SD2_R_LEN[15:0]	DMA Data length [15:0]	R/W	0

Reg_DMA_SD2_R_CTRL (0xF13B) : initial value = 0x0000

Bit	Function	Description	R/W	Reset Value
15	DMA_SD2_R_TRG	Trigger to transfer data from RAM to SD2, hardware auto clear after finish 1: Trigger DMA start	R/W	0
14:9	--	Reserved	--	0
8	DMA_SD2_R_RS	WRAM address select 0: 0~4KW, 1: 4~16KW	R/W	0
7:2	--	Reserved	--	0
1:0	DMA_SD2_R_LEN[17:16]	DMA Data length [17:16]	R/W	0

USB Device to WRAM

Reg_DMA_USB_DEV_W_ADDR (0xF112) : initial value = 0x0000

Bit	Function	Description	R/W	Reset Value
15:0	DMA_USB_DEV_W_ADDR[15:0]	WRAM Address	R/W	0

Reg_DMA_USB_DEV_W_LEN (0xF113) : initial value = 0x0000

Bit	Function	Description	R/W	Reset Value
15:0	DMA_USB_DEV_W_LEN[15:0]	DMA Data length [15:0]	R/W	0

Reg_DMA_USB_DEV_W_CTRL (0xF114) : initial value = 0x0000

Bit	Function	Description	R/W	Reset Value
15	DMA_USB_DEV_W_TRG	Trigger to transfer data from USB Device to RAM, hardware auto clear after finish 1: Trigger DMA start	R/W	0
14:9	--	Reserved	--	0
8	DMA_USB_DEV_W_RS	WRAM address select 0: 0~4KW, 1: 4~16KW	R/W	0
7:2	--	Reserved	--	0
1:0	DMA_USB_DEV_W_LEN[17:16]	DMA Data length [17:16]	R/W	0

WRAM to USB Device

Reg_DMA_USB_DEV_R_ADDR (0xF115) : initial value = 0x0000

Bit	Function	Description	R/W	Reset Value
15:0	DMA_USB_DEV_R_ADDR[15:0]	WRAM Address	R/W	0

Reg_DMA_USB_DEV_R_LEN (0xF116) : initial value = 0x0000

Bit	Function	Description	R/W	Reset Value
15:0	DMA_USB_DEV_R_LEN[15:0]	DMA Data length [15:0]	R/W	0

Reg_DMA_USB_DEV_R_CTRL (0xF117) : initial value = 0x0000

Bit	Function	Description	R/W	Reset Value
15	DMA_USB_DEV_R_TRG	Trigger to transfer data from RAM to USB Device, hardware auto clear after finish 1: Trigger DMA start	R/W	0
14:9	--	Reserved	--	0
8	DMA_USB_DEV_R_RS	WRAM address select 0: 0~4KW, 1: 4~16KW	R/W	0
7:2	--	Reserved	--	0
1:0	DMA_USB_DEV_R_LEN[17:16]	DMA Data length [17:16]	R/W	0

EXTF to WARM

Reg_DMA_EXTF_W_ADDR (0xF121) : initial value = 0x0000

Bit	Function	Description	R/W	Reset Value
15:0	DMA_EXTF_W_ADDR[15:0]	WRAM Address	R/W	0

Reg_DMA_EXTF_W_LEN (0xF122) : initial value = 0x0000

Bit	Function	Description	R/W	Reset Value
15:0	DMA_EXTF_W_LEN[15:0]	DMA Data length [15:0]	R/W	0

Reg_DMA_EXTF_W_CTRL (0xF123) : initial value = 0x0000

Bit	Function	Description	R/W	Reset Value
15	DMA_EXTF_W_TRG	Trigger to transfer data from External Flash or 8080 LCM to RAM, hardware auto clear after finish 1: Trigger DMA start	R/W	0

14:9	--	Reserved	--	0
8	DMA_EXTF_W_RS	WRAM address select 0: 0~4KW, 1: 4~16KW	R/W	0
7:2	--	Reserved	--	0
1:0	DMA_EXTF_W_LEN[17:16]	DMA Data length [17:16]	R/W	0

Reg_DMA_EXTF_RW_ADDR_H (0xF127) : initial value = 0x0000

Bit	Function	Description	R/W	Reset Value
15:0	--	Reserved	--	0
7:0	EXTF_RW_ADDR [23:16]	EXTF Flash Address [23:16] [23]=0 => CS1	R/W	All “0”

Reg_DMA_EXTF_RW_ADDR_L (0xF128) : initial value = 0x0000

Bit	Function	Description	R/W	Reset Value
15:0	EXTF_RW_ADDR [15:0]	EXTF Flash Address [15:0]	R/W	All “0”

WRAM to EXTF

Reg_DMA_EXTF_R_ADDR (0xF124) : initial value = 0x0000

Bit	Function	Description	R/W	Reset Value
15:0	DMA_EXTF_R_ADDR[15:0]	WRAM Address	R/W	0

Reg_DMA_EXTF_R_LEN (0xF125) : initial value = 0x0000

Bit	Function	Description	R/W	Reset Value
15:0	DMA_EXTF_R_LEN[15:0]	DMA Data length [15:0]	R/W	0

Reg_DMA_EXTF_R_CTRL (0xF126) : initial value = 0x0000

Bit	Function	Description	R/W	Reset Value
15	DMA_EXTF_R_TRG	Trigger to transfer data from RAM to External Flash or 8080 LCM, hardware auto clear after finish 1: Trigger DMA start	R/W	0
14:9	--	Reserved	--	0
8	DMA_EXTF_R_RS	WRAM address select 0: 0~4KW, 1: 4~16KW	R/W	0

7:2	--	Reserved	--	0
1:0	DMA_EXTF_R_LEN[17:16]	DMA Data length [17:16]	R/W	0

Reg_DMA_EXTF_RW_ADDR_H (0xF127) : initial value = 0x0000

Bit	Function	Description	R/W	Reset Value
15:0	--	Reserved	--	0
7:0	EXTF_RW_ADDR [23:16]	EXTF Flash Address [23:16] [23]=0 => CS1	R/W	All “0”

Reg_DMA_EXTF_RW_ADDR_L (0xF128) : initial value = 0x0000

Bit	Function	Description	R/W	Reset Value
15:0	EXTF_RW_ADDR [15:0]	EXTF Flash Address [15:0]	R/W	All “0”

External Flash to PRAM (DMA1)

Reg_DMA_EXTF_ADDR_H (0xFE27) : initial value = 0x0000

Bit	Function	Description	R/W	Reset Value
15:0	--	Reserved	--	0
7:0	EXTF_ADDR [23:16]	EXTF Flash Address [23:16] [23]=0 => CS1	R/W	All “0”

Reg_DMA_EXTF_ADDR_L (0xFE28) : initial value = 0x0000

Bit	Function	Description	R/W	Reset Value
15:0	EXTF_ADDR [15:0]	EXTF Flash Address [15:0]	R/W	All “0”

Reg_DMA1_PRAM_ADDR_H (0xFE29) : initial value = 0x0000

Bit	Function	Description	R/W	Reset Value
15:8	--	Reserved	--	All “0”
7:0	DMA1_PRAM_ADDR[23:16]	PRAM Address [23:16]	R/W	All “0”

Reg_DMA1_PRAM_ADDR_L (0xFE2A) : initial value = 0x0000

Bit	Function	Description	R/W	Reset Value
15:0	DMA1_PRAM_ADDR[15:0]	PRAM Address [15:0]	R/W	All “0”

Reg_DMA1_LEN (0xFE2B) : initial value = 0x0000

Bit	Function	Description	R/W	Reset Value
15:13	--	Reserved	--	All “0”
12:0	DMA1_LEN[12:0]	DMA Data length [12:0]	R/W	All “0”

Reg_DMA1_CTRL (0xFE2C) : initial value = 0x0000

Bit	Function	Description	R/W	Reset Value
15:1	---	DMA Data length [15:0]	--	All “0”
0	DMA1_TRG	Trigger to transfer data from EXTF to PRAM, hardware auto clear after finish 1: Trigger DMA start	R/W	0

Example : //DMA1 sample code

```
void main(void)
{
    Reg_DMA1_EXTF_ADDR_H = 0x0040;          // Set External storage address is 0x400000
    Reg_DMA1_EXTF_ADDR_L = 0x0000;
    Reg_DMA1_PRAM_ADDR_H = 0x0020;      // Set PRAM address is 0x201000
    Reg_DMA1_PRAM_ADDR_L = 0x1000;
    Reg_DMA1_LEN = 0x300;                   // Set length is 0x300
    Reg_DMA1_CTRL = 1;                      // Start DMA1

    while(Reg_DMA1_CTRL & 0x0001);          //Wait DMA1 stop
}
```

WRAM to PRAM (DMA2)

Reg_DMA2_WRAM_ADDR (0xF132) : initial value = 0x0000

Bit	Function	Description	R/W	Reset Value
15:0	DMA2_WRAM_ADDR[15:0]	WRAM Address	R/W	All “0”

Reg_DMA2_LEN (0xF133) : initial value = 0x0000

Bit	Function	Description	R/W	Reset Value
15:0	DMA2_LEN[15:0]	DMA Data length [15:0]	R/W	All “0”

Reg_DMA2_CTRL (0xF134) : initial value = 0x0000

Bit	Function	Description	R/W	Reset Value
15	DMA2_R_TRG	Trigger to transfer data from WRAM to PRAM, hardware auto clear after finish 1: Trigger DMA start	R/W	0
14:9	--	Reserved	--	0
8	DMA2_R_RS	WRAM address select	R/W	0

		0: 0~4KW, 1: 4~16KW		
7:2	--	Reserved	--	0
1:0	DMA2_R_LEN[17:16]	DMA Data length [17:16]	R/W	00

Reg_DMA2_PRAM_ADDR (0xF135) : initial value = 0x0000

Bit	Function	Description	R/W	Reset Value
15:0	DMA2_PRAM_ADDR[15:0]	PRAM Address [15:0]	R/W	All "0"

Example : //DMA2 sample code

```
void main(void)
{
    Reg_DMA2_WRAM_ADDR = 0x0100;    // Set WRAM address is 0x0100
    Reg_DMA2_PRAM_ADDR = 0x4000;    // Set PRAM address is 0x4000
    Reg_DMA2_LEN = 0x300;           // Set length is 0x300
    Reg_DMA2_CTRL = 1;              // Start DMA2

    while(Reg_DMA2_CTRL & 0x0001);  //Wait DMA1 stop
}
```

Huffman Deoder to WRAM

Reg_DMA_HFD_W_ADDR (0xF10C) : initial value = 0x0000

Bit	Function	Description	R/W	Reset Value
15:0	DMA_HFD_W_ADDR[15:0]	WRAM Address	R/W	0

Reg_DMA_HFD_W_LEN (0xF10D) : initial value = 0x0000

Bit	Function	Description	R/W	Reset Value
15:0	DMA_HFD_W_LEN[15:0]	DMA Data length [15:0]	R/W	0

Reg_DMA_HFD_W_CTRL (0xF10E) : initial value = 0x0000

Bit	Function	Description	R/W	Reset Value
15	DMA_HFD_W_TRG	Trigger to transfer data from Huffman Decoder to RAM, hardware auto clear after finish 1: Trigger DMA start	R/W	0
14:9	--	Reserved	--	0
8	DMA_HFD_W_RS	WRAM address select 1: 4~16KW (Always is "1")	R/W	1
7:2	--	Reserved	--	0

1:0	DMA_HFD_W_LEN[17:16]	DMA Data length [17:16]	R/W	0

WRAM to Huffman Deoder

Reg_DMA_HFD_R_ADDR (0xF10F) : initial value = 0x0000

Bit	Function	Description	R/W	Reset Value
15:0	DMA_HFD_R_ADDR[15:0]	WRAM Address	R/W	0

Reg_DMA_HFD_R_LEN (0xF110) : initial value = 0x0000

Bit	Function	Description	R/W	Reset Value
15:0	DMA_HFD_R_LEN[15:0]	DMA Data length [15:0]	R/W	0

Reg_DMA_HFD_R_CTRL (0xF111) : initial value = 0x0000

Bit	Function	Description	R/W	Reset Value
15	DMA_HFD_R_TRG	Trigger to transfer data from RAM to Huffman Decoder, hardware auto clear after finish 1: Trigger DMA start	R/W	0
14:9	--	Reserved	--	0
8	DMA_HFD_R_RS	WRAM address select 1: 4~16KW (Always is “1”)	R/W	1
7:2	--	Reserved	--	0
1:0	DMA_HFD_R_LEN[17:16]	DMA Data length [17:16]	R/W	0

3.3.21. Instruction Set Table for DSP

Operation	Mnemonic		ZF	SF	CF	AV	MV	Commands	Clock	Word
Call	Call <absolute address>		-	-	-	-	-	32K Words in a Program Bank	2	1
Jump	Jmp <relative address>		-	-	-	-	-	+/- 2K words in one relative jump	2	1
Jump Condition	Jeq/jz <relative address>	if zero	-	-	-	-	-	+/- 128 words in one relative jump	2 (take) 1 (not)	1
	Jne/Jnz <relative address>	if not zero								
	Jgt <relative address>	if greater than								
	Jge <relative address>	if greater than or equal								
	Jlt <relative address>	if less than								
	Jle <relative address>	if less than or equal								
	Jav <relative address>	if ALU overflow								
	Jnav <relative address>	if not ALU overflow								
	Jac <relative address>	if ALU carry out								
	jnac < relative address>	if not ALU carry out								
	Jmr0s <relative address>	if mr0 is signed (mr0[15] = 1)								
	Jmr0ns <relative address>	if mr0 is not signed (mr0[15] = 0)								
	Jmv <relative address>	if MAC is overflow								
	Jnmv <relative address>	if MAC is not overflow								
Jump Condition	Jfeq/jz <relative address>	if zero	-	-	-	-	-	+/- 2K words in one relative jump	3 (take) 2 (not)	2
	Jfne/Jnz <relative address>	if not zero								
	Jfgt <relative address>	if greater than								
	Jfge <relative address>	if greater than or equal								
	Jflt <relative address>	if less than								
	Jfle <relative address>	if less than or equal								
	Jfav <relative address>	if ALU overflow								
	Jfnav <relative address>	if not ALU overflow								
	Jfac <relative address>	if ALU carry out								
	Jfnac < relative address>	if not ALU carry out								
	Jfmr0s <relative address>	if mr0 is signed (mr0[15] = 1)								
	Jfmr0ns <relative address>	if mr0 is not signed (mr0[15] = 0)								
	Jfmv <relative address>	if MAC is overflow								
	Jfnmv <relative address>	if MAC is not overflow								

Operation	Mnemonic		ZF	SF	CF	AV	MV	Commands	Clock	Word
Jump Condition	Jffeq/jz <relative address> Jffne/Jnz <relative address> Jffgt <relative address> Jffge <relative address> Jfflt <relative address> Jffle <relative address> Jffav <relative address> Jffnav <relative address> Jffac <relative address> Jffnac < relative address> Jffmr0s <relative address> Jffmr0ns <relative address> Jffmv <relative address> Jffnmv <relative address>	if zero if not zero if greater than if greater than or equal if less than if less than or equal if ALU overflow if not ALU overflow if ALU carry out if not ALU carry out if mr0 is signed (mr0[15] = 1) if mr0 is not signed (mr0[15] = 0) if MAC is overflow if MAC is not overflow	-	-	-	-	-	+/- 16M words in one relative jump	4(take) 2(not)	3
RW SRAM (direct)	| R | = RAMname RAMname = | R |	R=X0/X1/Y0/Y1/R0/R1/MR0/MR1	-	-	-	-	-	512 Words in a Memory Bank	1	1
Load Immediate	| R | .h = <immediate> | R | .l = <immediate> | R | = <immediate>	R=X0/X1/Y0/Y1/R0/R1/Ix0						Load Byte Load Word	1 2	1 2
RW SRAM (indirect)	| R | = RAM(IN) | R | = RAM(IN, 1) | R | = RAM(IN, -1) | R | = RAM(IN, m) RAM(IN) = | R | RAM(IN, 1) = | R | RAM(IN, -1) = | R | RAM(IN, m) = | R |	R=X0/X1/Y0/Y1/R0/R1/MR0/MR1 IN = Ix0/Iy0/Iy1	-	-	-	-	-	64K Words in a Memory Bank	1	1
Load ROM (indirect)	| R | = ROM(IN) | R | = ROM(IN, 1) | R | = ROM(IN, -1) | R | = ROM(IN, m)	R=X0/X1/Y0/Y1/R0/R1/MR0/MR1 IN = Ix0/Iy0/Iy1	-	-	-	-	-	64K Words in a Memory Bank	2	1
I/O	| R | = IO(<address>) IO(<address>) = | R |	R = X0/X1/R0/R1	-	-	-	-	-	128 words in Special Register	1	1

Operation	Mnemonic		ZF	SF	CF	AV	MV	Commands	Clock	Word
Push/Pop I/O	Push IO(<address>) Pop IO(<address>)		-	-	-	-	-	128 words in Special Register	1	1
AU(1)	R = R + 1 R = R – 1 R = R + Yop R = R + Yop + C R = R – Yop R = R – Yop + C – 1 R = - R + Yop R = - R + Yop + C – 1	R=X0/X1/Y0/Y1/R0/R1/MR0/MR1 Yop = Y0/Y1/R0/R1	*	*	*	*	-		1	1
AU(2)	RAM(IxN, Im) = Xop + 1 RAM(IxN, Im) = Xop – 1 RAM(IxN, Im) = Xop + Yop RAM(IxN, Im) = Xop + Yop + C RAM(IxN, Im) = Xop – Yop RAM(IxN, Im) = Xop – Yop + C – 1 RAM(IxN, Im) = -Xop + Yop RAM(IxN, Im) = -Xop + Yop + C – 1	Xop = X0/X1/R0/R1 Yop = Y0/Y1/R0/R1 IxN = Ix0/Iy0/Iy1 Im =/ /1/-1/m	*	*	*	*	-		1	1
LU(1)	| R | = Xop AND Yop | R | = Xop OR Yop | R | = Xop XOR Yop | R | = NOT Xop	R=X0/X1/Y0/Y1/R0/R1/MR0/MR1 Xop = X0/X1/R0/R1 Yop = Y0/Y1/R0/R1	*	*	*	*	-		1	1
LU(2)	| Rx | = BSET.x | Yop | | Rx | = BCLR.x | Yop| | Rx | = BTST.x | Yop | | Rx | = BTOG.x | Yop|	Rx= R0/R1 Yop = Y0/Y1/R0/R1 x = 0 ~ 15	*	*	*	*	-		1	1

Operation	Mnemonic		ZF	SF	CF	AV	MV	Commands	Clock	Word
MAC	MR = Xop * Yop (IS) MR = MR + Xop * Yop (IS) MR = MR – Yop * Yop (IS) MR = Xop * Yop (FS) MR = MR + Xop * Yop (FS) MR = MR – Yop * Yop (FS)	Xop = X0/X1 Yop = Y0/Y1	-	-	-	-	*	IS = Integer Signed FS= Fractional Signed	1	1
	MR = Xop * Yop (IS), |DR| = RAM(IxN, |,1,-1|) MR = MR + Xop * Yop (IS), |DR| = RAM(IxN, |,1,-1|) MR = MR - Xop * Yop (IS), |DR| = RAM(IxN, |,1,-1|) MR = Xop * Yop (FS), |DR| = RAM(IxN, |,1,-1|) MR = MR + Xop * Yop (FS), |DR| = RAM(IxN, |,1,-1|) MR = MR - Xop * Yop (FS), |DR| = RAM(IxN, |,1,-1|)	Xop = X0/X1 Yop = Y0/Y1 DR = X0/X1/Y0/Y1 IxN = Ix0	-	-	-	-	*	(multiple function) IS= Integer Signed FS= Fractional Signed	1	1
Internal Reg Move	| R | = | R |	R=X0/X1/Y0/Y1/R0/R1/MR0/MR1	-	-	-	-	-		1	1
Push/Pop	Push | R | Pop | R |	R=X0/X1/Y0/Y1/R0/R1/MR0/MR1	-	-	-	-	-		1	1
Shift	| Rx | = SL R | Rx | = SRL R | Rx | = SRA R	Rx = R0/R1 R=X0/X1/Y0/Y1/R0/R1/MR0/MR1	*	*	*	*	-		1	1
Callff	Callff <Absolute Address>		-	-	-	-	-	Call Far Far (24-bit absolute address)	3	2
Jumpff	Jumpff <Absolute Address>		-	-	-	-	-	Jump Far Far (24-bit absolute address)	3	2
ret/reti/retff	Ret Reti Retff	; Return from call subroutine ; Return from interrupt service routine ; Return from callff subroutine	-	-	-	-	-		2 2 3	1
NOP		; No operation	-	-	-	-	-		1	1

Note:
"-": Does not change the status flag
"*": Could possibly change the status flag

● Call

Syntax:

```
Call      <address (15bits)>          ; First 32K words
Callff    <address (24bits)>          ; Whole 16M words region
```

Descriptions:

Save (current_address+1) into system call stack, update system call stack pointer, and jump to the target address (subroutine address).

Status Generated:

ZF	SF	CF	AV	MV	IV
-	-	-	-	-	-

Example:

```
Call subroutine
```

● Jump

Syntax:

```
jmp <relative address>              ; unconditional jump
jmpff<absolute address>             ; unconditional jump(whole 16M)

<jcc => +/- 128 words offset, takes 1 instruction, 1 cycle>
jeq/jz <relative address>           ; jump if equal
jne/jnz <relative address>          ; jump if not equal
jgt <relative address>              ; jump if greater than
jge <relative address>              ; jump if greater than or equal
jlt <relative address>              ; jump if less than
jle <relative address>              ; jump if less than or equal
jav <relative address>              ; jump if ALU overflow
jnav <relative address>             ; jump if not ALU overflow
jac <relative address>              ; jump if ALU carry
jnac <relative address>             ; jump if not ALU carry
jmr0s <relative address>            ; jump if mr0 signed
jmr0ns <relative address>           ; jump if mr0 not signed
jmv <relative address>              ; jump if MAC overflow
jnmv <relative address>             ; jump if not MAC overflow
```

<jfcc => +/- 2K words offset, takes 2 instruction, 2 cycle>

jfeq/jfz <relative address>	; jump if equal
jfne/jfnz <relative address>	; jump if not equal
jfgt <relative address>	; jump if greater than
jfge <relative address>	; jump if greater than or equal
jflt <relative address>	; jump if less than
jfle <relative address>	; jump if less than or equal
jfav <relative address>	; jump if ALU overflow
jfnav <relative address>	; jump if not ALU overflow
jfac <relative address>	; jump if ALU carry
jfnac <relative address>	; jump if not ALU carry
jfrm0s <relative address>	; jump if mr0 signed
jfmr0ns <relative address>	; jump if mr0 not signed
jfmv <relative address>	; jump if MAC overflow
jfnmv <relative address>	; jump if not MAC overflow

<jffcc => 16M words absolute address, takes 3 instruction, 3 cycle(taken). 2 cycle (not taken>

jffeq/jffz <relative address>	; jump if equal
jffne/jffnz <relative address>	; jump if not equal
jffgt <relative address>	; jump if greater than
jffge <relative address>	; jump if greater than or equal
jfflt <relative address>	; jump if less than
jffle <relative address>	; jump if less than or equal
jffav <relative address>	; jump if ALU overflow
jffnav <relative address>	; jump if not ALU overflow
jffac <relative address>	; jump if ALU carry
jffnac <relative address>	; jump if not ALU carry
jffmr0s <relative address>	; jump if mr0 signed
jffmr0ns <relative address>	; jump if mr0 not signed
jffmv <relative address>	; jump if MAC overflow
jffnmv <relative address>	; jump if not MAC overflow

Description:

Jump to the target address

Jump/jumpcc/jumpfcc: (Current Address + signed offset).

Jumpff

Status Generated:

ZF	SF	CF	AV	MV	IV
-	-	-	-	-	-

Permission Condition:

See Syntax Jump if condition match, else sequential execute.

Example:

```
Jmp       next_target_addr
Jeq       next_target_addr
Jgt       next_target_addr
Jmpff     next_target_addr
```

● Read/Write RAM (direct)

Syntax:

```
| R | = RAMname                    ; Define the RAMName, RAM's address and
RAMname = | R |                    ; Size in Data segment.
```

Description:

Access SRAM data with direct addressing

Status Generated:

ZF	SF	CF	AV	MV	IV
-	-	-	-	-	-

Permission Register:

X0, X1, Y0, Y1, R0, R1, MR0, MR1

Example:

```
X0 = RAMName1
```

RAMName2 = R0

● Load Immediate

Syntax:

```
| R |.H = <immediate>          ; High byte load immediate / Low byte no change
| R |.L = <immediate>          ; High byte no change / Low byte load immediate
| R | = <immediate>            ; Load one word data immediate
```

Descriptions:

Load immediate into register.

Status Generated:

ZF	SF	CF	AV	MV	IV
-	-	-	-	-	-

Permission Load Style:

High byte only(.h)
Low byte only(.l)
All word()

Permission Register:

X0, X1, Y0, Y1, R0, R1, Ix0

Example:

```
X0.h = 0x00
R0.l = 0xaf
R0 = 0xA102
Ix0 = 0x1234
```

● Read/Write RAM (indirect)

Syntax:

```
| R | = RAM(IN, Im)            ; Load SRAM(IN), then (IN <= IN + Im)
RAM(IN, Im) = | R |            ; Store SRAM(IN), then (IN <= IN + Im)
| R | = RAM(IN)                ; Load SRAM(IN)
| R | = RAM(IN, 1) ;           ; Load SRAM(IN), then (IN <= IN + 1)
| R | = RAM(IN, -1)            ; Load SRAM(IN), then (IN <= IN – 1)
RAM(IN) = | R |                ; Load SRAM(IN)
RAM(IN, 1) = | R|              ; Load SRAM(IN), then (IN <= IN + 1)
RAM(IN, -1) = | R |            ; Load SRAM(IN), then (IN <= IN - 1)
```

Descriptions:

Read/Write SRAM data from/to register.

Status Generated:

ZF	SF	CF	AV	MV	IV
-	-	-	-	-	*

Permission Load Style:
Next data address = data address
Next data address = data address + 1
Next data address = data address - 1
Next data address = data address + Modifier Register

Permission Register:
X0, X1, Y0, Y1, R0, R1, MR0, MR1

Permission modifier Register (Im): 1/-1/ /m

Permission IN:
Ix0/Iy0/Iy1

Example:

```
X0 = RAM(Ix0)
Y0 = RAM(Iy1, 1)
RAM(Ix0, -1) = R1
RAM(Iy0, m) = R0
```

● Read ROM (indirect)

Syntax:

```
| R | = ROM (IN, Im)      ; Load ROM(IN), then (IN <= IN + Im)
| R | = ROM(IN)           ; Load ROM(IN)
| R | = ROM(IN, 1)        ; Load ROM(IN), then (IN <= IN + 1)
| R | = ROM(IN, -1)       ; Load ROM(IN), then (IN <= IN - 1)
```

Descriptions:
Read ROM data to register.

Status Generated:

ZF	SF	CF	AV	MV	IV
-	-	-	-	-	*

Permission Registers:
X0, X1, Y0, Y1, R0, R1, MR0, MR1

Permission modifier Register (Im): 1/-1/ /m

Permission IN:
Ix0/Iy0/Iy1

Example:

```
X0 = ROM(Ix0)
Y0 = ROM(Iy0, 1)
R0 = ROM(Iy1, -1)
MR1 = ROM(Ix0, m)
```

● IO

Syntax:

```
| R | = IO(<address>)          ; Load IO
| R | = IOName                 ; Load IO
IO(<address>) = | R |          ; Store IO
IOName= | R |                  ; Store IO
```

Descriptions:

Read/write I/O register data from/to register.

Status Generated:

ZF	SF	CF	AV	MV	IV
-	-	-	-	-	-

Permission Register:

X0, X1, R0, R1

Example:

```
X0 = IO(0x00)
IO(0x0C) = R1
X0 = SSF          ; It is identical to the instruction “X0 = IO(0x00)”
PCR1 = R1         ; It is identical to the instruction “IO(0x24) = R1”
```

● Push/Pop I/O

Syntax:

```
Push IO(<address>)          ; Push IO
Push IOName                 ; Push IO
Pop IO(<address>)           ; Pop IO
Pop IOName                  ; Pop IO
```

Descriptions:

Read/write I/O register data from/to Data Stack.

Status Generated:

ZF	SF	CF	AV	MV	IV
-	-	-	-	-	-

Example:

```
Push IO(0x00)
Pop IO(0x02)
Push SSF          ; It is identical to the instruction “Push IO(0x00)”
Pop Ix0           ; It is identical to the instruction “Pop IO(0x02)”
```

● AU(1)

Syntax:

| R | = | R | .OP. Yop

Descriptions:

Operate R operand with Y operand. Write the result into R

Status Generated:

ZF	SF	CF	AV	MV	IV
*	*	*	*	-	-

Permission AU:

R + 1	R - Y
R – 1	R – Y + C - 1
R + Y	-R + Y
R + Y + C	-R + Y + C - 1

Permission Register:

X0, X1, Y0, Y1, R0, R1, MR0, MR1

Permission Yop:

Y0, Y1, R0, R1

Example:

```
R0 = X0 + Y0
R1 = X0 + R0
X0 = R1 + Y0
X0 = X0 + R0
Y0 = Y0 + Y1
MR0 = MR0 + 1
```

● AU(2)

Syntax:

RAM(IxN, Im) = Xop .OP. Yop
RAM(IxN, Im) = Xop .OP. Yop

Descriptions:

Operate X operand with Y operand. Write the result into RAM

Status Generated:

ZF	SF	CF	AV	MV	IV
*	*	*	*	-	-

Permission AU:

X + 1	X - Y
X – 1	X – Y + C -1
X + Y	-X + Y
X + Y + C	-X + Y + C -1

Permission Xop:

X0, X1, Y0, Y1, R0, R1, MR0, MR1

Permission Yop:

Y0, Y1, R0, R1

Permission modifier Register (Im): 1/-1/ /m

Permission IxN:

Ix0/Iy0/Iy1

Example:

RAM(Ix0) = X0 + Y0
RAM(Ix0, 1) = X0 - R0
RAM(Iy0, -1) = X0 – R1

● LU(1)

Syntax:

| R | = Xop .OP. Yop

Descriptions:

Operate X operand with Y operand. Write the result into result register.

Status Generated:

ZF	SF	CF	AV	MV	IV
*	*	*	*	-	-

Permission LU:

Xop AND	Yop
Xop OR	Yop
Xop XOR	Yop
NOT	Xop

Permission Register:
X0, X1, Y0, Y1, R0, R1, MR0, MR1

Permission Xop:
X0, X1, R0, R1

Permission Yop:
Y0, Y1, R0, R1

Example:

```
R0 = X0 AND Y0
R1 = X0 OR R1
X0 = R1 XOR Y1
Y0 = X0 AND R1
R0 = NOT X1
```

● LU(2)

Syntax:

```
| R0/R1 | = BSET.x    | Yop |          ; Bit x Set as 1
| R0/R1 | = BCLR.x    | Yop |          ; Bit x Set as 0
| R0/R1 | = BTST.x    | Yop |          ; Bit x test 1
| R0/R1 | = BTOG.x    | Yop |          ; Bit x XOR
```

Descriptions:
Bit Operation

Status Generated

ZF	SF	CF	AV	MV	IV
*	*	*	*	-	-

Permission Yop:
Y0, Y1, R0, R1

Permission x:
x = 0~15

Example:

```
R0 = BSET.2    Y0        ; R0 = (0x0004) OR Yop
R0 = BCLR.9    Y1        ; R0 = (0xfdff) AND Yop
R1 = BTST.4    R0        ; R1 = (0x0010) AND Yop
R1 = BTOG.6    R1        ; R1 = (0x0040) XOR Yop
```

● MAC

Syntax:

| MR | = Xop * Yop (IS/FS	

| MR | = MR +/- Xop * Yop (IS/FS)	
| MR | = Xop * Yop (IS/FS), | Xop/Yop | = RAM(| Ix0 | , | , 1, -1, m|)	
| MR | = MR +/- Xop * Yop (IS/FS), | Xop/Yop | = RAM(| Ix0 | , | , 1, -1, m|)	

Descriptions:

Operate X operand with Y operand. Write the result into result register (MR)

Status Generated:

ZF	SF	CF	AV	MV	IV
-	-	-	-	*	-

Permission MAC:

Xop * Yop (IS)
MR +/- Xop * Yop (IS)
MR +/- Xop * Yop (IS), Xop/Yop = RAM (Ix0, 1/-1/ /m)
Xop * Yop (FS)
MR +/- Xop * Yop (FS)
MR +/- Xop * Yop (FS), Xop/Yop = RAM (Ix0, 1/-1/ /m)

Permission modifier Register (Im): 1/-1/ /m

Permission X:

X0, X1

Permission Y:

Y0, Y1

Example:

```
; Simple MAC
MR = X0 * Y0(IS)                    ; Integer Signed
MR = MR + X0 * Y1 (IS)              ; Integer Signed
MR = MR – X1 * Y0 (IS)              ; Integer Signed
MR = X0 * Y0 (FS)                   ; Fractional Signed
MR = MR + X0 * Y0 (FS)              ; Fractional Signed

; Multiple – Function
MR = X0 * Y0 (IS), X0 = RAM(Ix0, 1)
MR = MR + X1 * Y1(IS)

; A General Case of Complex Variable Multiplication
MR = MR + X0 * Y1 (FS), Y1 = RAM(Ix0, 1)      ; Fractional Signed
MR = MR + X1 * Y0 (FS), X1 = RAM(Ix0, -1)     ; Fractional Signed
; (Write MR)
MR = MR + X0 * Y0 (FS), Y0 = RAM(Ix0, 1)      ; Fractional Signed
MR = MR + X0 * Y0 (FS), X0 = RAM(IX0, -1)     ; Fractional Signed
; (Write MR)
```

- **Internal Register Move**

Syntax:

| R | = | R |

Descriptions:

Move internal register.

Status Generated:

ZF	SF	CF	AV	MV	IV
-	-	-	-	-	-

Permission Registers:

X0, X1, Y0, Y1, R0, R1, MR0, MR1

Example:

```
X0 = R1
Y0 = MR0
X1 = Y1
MR1 = R0
R0 = Y0
Y1 = X0
```

● Push/Pop register

Syntax:

```
Push   | R |
Pop    | R |
```

Descriptions:

Push: Write register to memory at stack pointer address. Add 1 to stack pointer.
Pop: Read memory at stack pointer address to register. Subtract 1 from stack pointer.

Status Generated:

ZF	SF	CF	AV	MV	IV
-	-	-	-	-	-

Permission Registers:

X0, X1, Y0, Y1, R0, R1, MR0, MR1

Example:

```
Push X0
Push R0
Pop R1
Pop Y1
```

● Shift

Syntax:

```
| R0 / R1 | = SL      | R |          ; Left Shift by 1 bit
| R0 / R1 | = SRL     | R |          ; Right Logic Shift by 1 bit
| R0 / R1 | = SRA     | R |          ; Right Arithmetic Shift by 1 bit
```

Descriptions:

Shift register context, then write into result register.

Status Generated:

ZF	SF	CF	AV	MV	IV
*	*	*	*	-	-

Permission Registers:

X0, X1, Y0, Y1, R0, R1, MR0, MR1

Example:

```
R0 = SL Y0
R1 = SRA R1
R1 = SRL MR0
```

● Ret/Reti/Retff

Syntax:

```
RET                          ; Pop 1 word return address
RETI                         ; Pop 2 word return address
RETFF                        ; Pop 2 word return address
```

Description:

Pop program return address from program stack. Minus program stack. Jump to the return address.

Status Generated:

ZF	SF	CF	AV	MV	IV
-	-	-	-	-	-

Example:

```
Ret
Reti
Retff
```

● NOP

Syntax:

```
NOP                          ; No operation
```

Descriptions:

No operation.

Status Generated:

ZF	SF	CF	AV	MV	IV
-	-	-	-	-	-

3.4. Peripherals

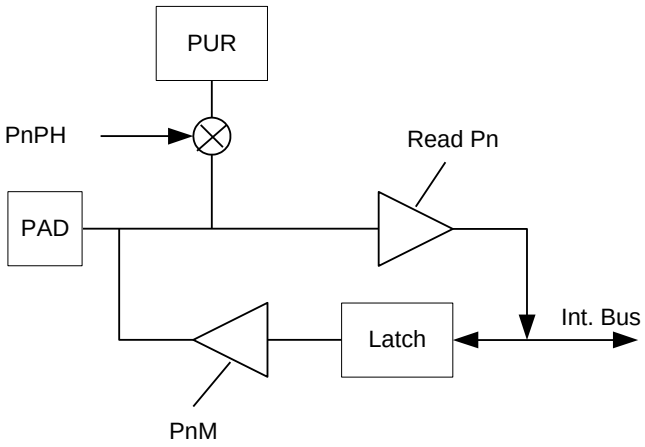
3.4.1. Multi-function of I/O

						SNP70032XXB2FG (LQFP100)	SNP70032XXB1FG (LQFP80)
PORT0	P0.0	INT	INT0			●	●
	P0.1		INT0			●	
	P0.2		INT0			●	
	P0.3	PWM	PWMIO#0			●	●
	P0.4		PWMIO#1			●	●
	P0.5		PWMIO#2			●	
	P0.6		PWMIO#3			●	
	P0.7	SPI#2	SPISCK2	ICE	ICE_SCK	●	●
	P0.8		SPIMISO2		ICE_CSB	●	●
	P0.9		SPIMOSI2		ICE_MOSI	●	●
	P0.10		SPICS2		ICE_MISO	●	●
	P0.11		SPID2			●	
	P0.12		SPID3			●	
	P0.13					●	
PORT1	P1.0	SPI#1	SPICS1			●	●
	P1.1		SPISCK			●	●
	P1.2		SPIMISO			●	●
	P1.3		SPIMOSI			●	●
	P1.4		SPIED2			●	●
	P1.5		SPIED3			●	●
	P1.6	I2C	MSP_CLK	I2S	I2S_WS	●	
	P1.7		MSP_DAT		I2S_SCL	●	
	P1.8				I2S_SD	●	
	P1.9				I2S_MCLK	●	
	P1.10	SD#2	SDCLK#2			●	
	P1.11		SDCMD#2			●	
	P1.12		SDD0#2			●	
	P1.13		SDD1#2			●	
	P1.14		SDD2#2			●	
	P1.15		SDD3#2			●	
PORT3	P3.11	Xtra ROM	NFCS	SD#2	SDCLK#2	●	●
	P3.12		R/B		SDCMD#2	●	●
	P3.13		NFALE		SDD0#2	●	●
	P3.14				SDD1#2	●	●

	P3.15		NFRE		SDD2#2	●	●
PORT4	P4.0		NFWP	SD#1	SDCLK	●	●
	P4.1		NFCLE		SDCMD	●	●
	P4.2		NFD0		SDD0	●	●
	P4.3		NFD1		SDD1	●	●
	P4.4		NFD2		SDD2	●	●
	P4.5		NFD3		SDD3	●	●
	P4.6		NFD4		SDD3#2	●	●
	P4.7		NFD5			●	●
	P4.8		NFD6			●	●
	P4.9		NFD7			●	●
	P4.12	SAR	AIN0			●	●
	P4.13	ADC	AIN1			●	
	P4.14		AIN2			●	
	P4.15		AIN3			●	

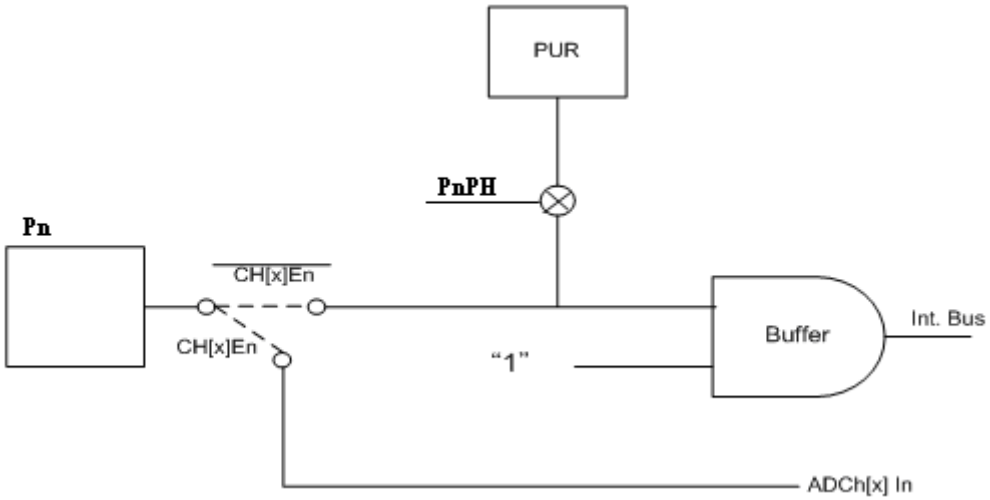
3.5. I/O Port

The I/O Ports are managed by the DSP. The direction and input pull high resistor of each pin can be individually programmed by the Port Mode and Port Pull-High registers.. These ports work as input ports without pull high resisters after the system resets. P0.0~P0.13, and P1.0~P1.15 can wake the chip up from the Stop mode if wake up function is enabled.



I/O Port

Note: The pull high resistor must be disabled manually for output mode.



Port 4.12 ~ Port 4.15 internal circuit.

3.5.1. Port Registers

P0En (0x0068) : initial value = 0x0000

Bit	Function	Description	R/W	Reset Value
15:0	P0.15En ~ P0.0En	Set P0.15 ~ P0.0 enable bit 0: disable, 1:enble	R/W	All “0”

P0 (0x0069) : initial value = 0xFFFF

Bit	Function	Description	R/W	Reset Value
15:0	P0.15 ~ P0.0	Set P0.15 ~ P0.0 port value 0: low, 1:high	R/W	All “1”

P0M (0x006A) : initial value = 0x0000

Bit	Function	Description	R/W	Reset Value
15:0	P0.15M ~ P0.0M	Set P0.15 ~ P0.0 port direction 0: input, 1:output	R/W	All “0”

.

P0PH (0x006B) : initial value = 0xFFFF

Bit	Function	Description	R/W	Reset Value
15:0	P0.15PH ~ P0.0PH	Enable P0.15 ~ P0.0 port pull-high resistor 0: disable, 1:enable	R/W	All “1”

Once Port has been set as output mode, user must disable the port pull-high resistor by himself.

P1En (0x006C) : initial value = 0x0000

Bit	Function	Description	R/W	Reset Value
15:0	P1.15En ~ P1.0En	Set P1.15 ~ P1.0 enable bit 0: disable, 1:enble	R/W	All “0”

P1 (0x006D) : initial value = 0xFFFF

Bit	Function	Description	R/W	Reset Value
15:0	P1.15 ~ P1.0	Set P1.15 ~ P1.0 port value 0: low, 1:high	R/W	All “1”

P1M (0x006E) : initial value = 0x0000

Bit	Function	Description	R/W	Reset Value
15:0	P1.15M ~ P1.0M	Set P1.15 ~ P1.0 port direction 0: input, 1:output	R/W	All “0”

.

P1PH (0x006F) : initial value = 0xFFFF

Bit	Function	Description	R/W	Reset Value
15:0	P1.15PH ~ P1.0PH	Enable P1.15 ~ P1.0 port pull-high resistor 0: disable, 1:enable	R/W	All “1”

Once Port has been set as output mode, user must disable the port pull-high resistor by himself.

P3En (0x0074) : initial value = 0x0000

Bit	Function	Description	R/W	Reset Value
15:11	P3.15En ~ P3.11En	Set P3.15 ~ P3.11 enable bit 0: disable, 1:enble	R/W	All “0”

※ P3.10En ~ P3.0En is always “0”

P3 (0x0075) : initial value = 0xF800

Bit	Function	Description	R/W	Reset Value
15:11	P3.15 ~ P3.11	Set P3.15 ~ P3.11 port value 0: low, 1:high	R/W	All “1”

※ P3.10 ~ P3.0 is always “0”

P3M (0x0076) : initial value = 0x0000

Bit	Function	Description	R/W	Reset Value
15:11	P3.15M ~ P3.11M	Set P3.15 ~ P3.11 port direction 0: input, 1:output	R/W	All “0”

※ P3.10M ~ P3.0M is always “0”

P3PH (0x0077) : initial value = 0xF800

Bit	Function	Description	R/W	Reset Value
15:11	P3.15PH ~ P3.11PH	Enable P3.15 ~ P3.11 port pull-high resistor 0: disable, 1:enable	R/W	All “1”

※ P3.10PH ~ P3.0PH is always “0”

Once Port has been set as output mode, user must disable the port pull-high resistor by himself.

P4En (0x0078) : initial value = 0x0000

Bit	Function	Description	R/W	Reset Value
15:0	P4.15En ~ P4.0En	Set P4.15 ~ P4.0 enable bit 0: disable, 1:enble	R/W	All “0”

※ P4.11En ~ P4.10En is always “0”

P4 (0x0079) : initial value = 0xF3FF

Bit	Function	Description	R/W	Reset Value
15:0	P4.15 ~ P4.0	Set P4.15 ~ P4.0 port value 0: low, 1:high	R/W	All “1”

※ P4.11 ~ P4.10 is always “0”

P4M (0x007A) : initial value = 0x0000

Bit	Function	Description	R/W	Reset Value
15:0	P4.15M ~ P4.0M	Set P4.15 ~ P4.0 port direction 0: input, 1:output	R/W	All “0”

※ P4.11M ~ P4.10M is always “0”

P4PH (0x007B) : initial value = 0xF3FF

Bit	Function	Description	R/W	Reset Value
15:0	P4.15PH ~ P4.0PH	Enable P4.15 ~ P4.0 port pull-high resistor 0: disable, 1:enable	R/W	All “1”

※ P4.11PH ~ P4.10PH is always “0”

Once Port has been set as output mode, user must disable the port pull-high resistor by himself.

Note: All pull high resistors are 55KΩ @3V.

Example:

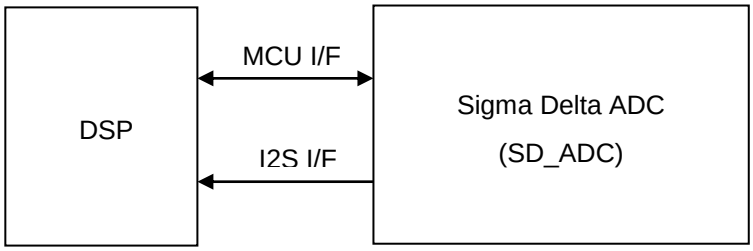
```
void main(void)
{
    _setSR(SFR_P1En, 0xFFFF);     //P1 enable
    _setSR(SFR_P1M, 0x0000);      //P1 input
    _setSR(SFR_P1PH, 0xFFFF);     //P1 pull high
}
```

3.5.2. P4 and SAR AD converter

P4.12~P4.15 share the same pins with SAR ADC AIN0~AIN3. If **CH[x]En** of ADM2 is set to 1, the corresponding Input port of P4.12~P4.15 will work as an input channel of SAR ADC.

3.6. ADC

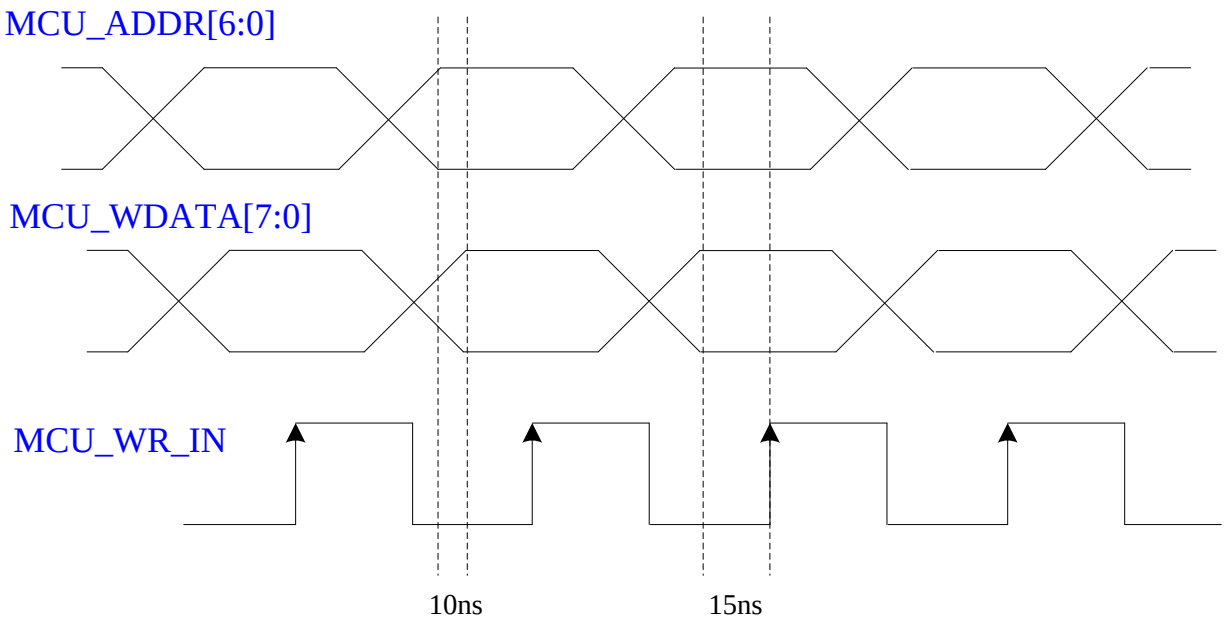
The SNP70032 provides one set of high performance single channel Analog-to-Digital Converter (ADC) for micro-phone applications with typical SNR of 90dB. Along with built-in Microphone Preamp, PGA and AGC function, the MCU supports I2S interface to link to DSP core.



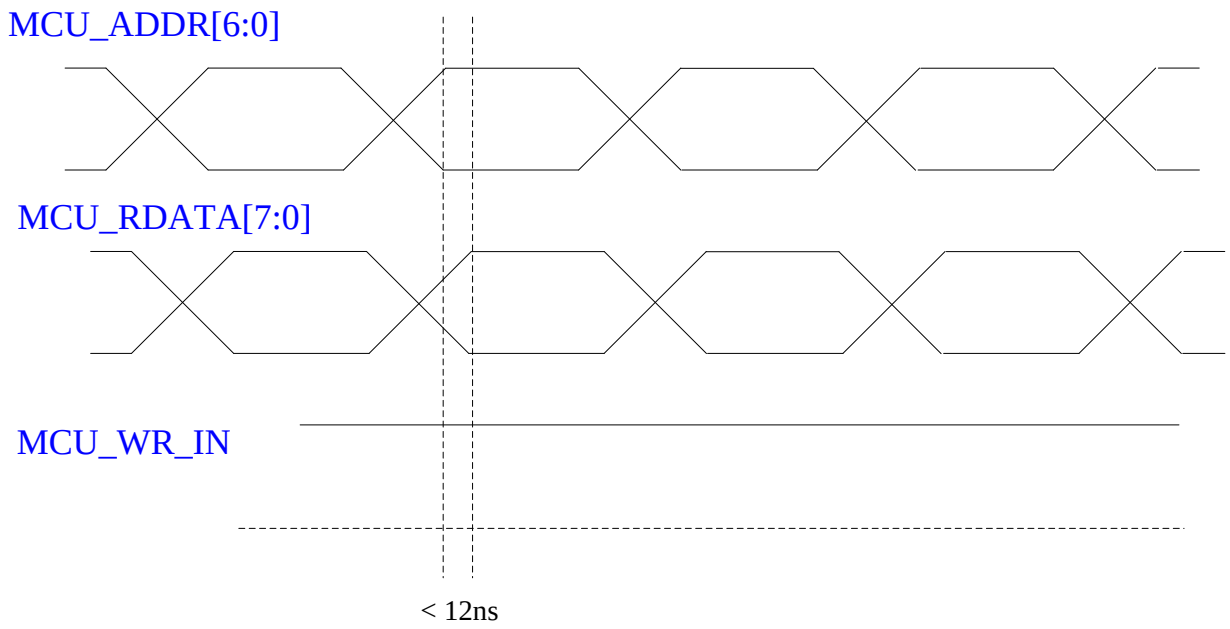
3.6.1. MCU Interface

The SNP70032 DSP core uses MCU interface to access the sigma-delta ADC internal registers and uses the I2S interface to transfer ADC output data into a FIFO. The following is the timing diagram.

MCU Write Mode



MCU Read Mode



ADCSET1 (0x002A) : initial value = 0x0000

Bit	Function	Description	R/W	Reset Value
15	MCU_WR_IN	MCU Interface Read/Write control bit 0: write, 1:read	R/W	0
14:7	--	Reserved	--	All “0”
6:0	MCU_ADDR[6:0]	MCU Interface Address bit	R/W	All “0”

ADCSET2 (0x002B) : initial value = 0x0000

Bit	Function	Description	R/W	Reset Value
15:8	--	Reserved	--	All “0”
7:0	MCU_DATA[7:0]	MCU Interface Data bit	R/W	All “0”

3.6.2. FIFO Interrupt

The SNP70032 ADC output data is sent to the DSP core via the I2S interface. H/W will automatically place the data into a 16 level FIFO. An interrupt will be issued when the FIFO timer has timed out. Data is then ready for reading out of the FIFO from the ADC_FIFO register.

ADC_FIFOStatus (0x0026) : Initial Value = 0x0200

Bit	Function	Description	R/W	Reset Value
15	TRG_ADC	Enable Sigma Delta ADC controller	R/W	0

		0: disable, 1: enable		
14:11	--	Reserved	--	All “0”
10:9	ADC_INT_MODE	ADC interrupt triggered is by FIFO status 00: interrupt trigger by FIFO full 01: interrupt trigger by FIFO half 10: interrupt trigger by FIFO empty 11: N/A	R/W	01
8	--	Reserved	--	0
7:4	ADC_SR	Set ADC Sampling Rate (KHz) 0000: 32, 0001: 16, 0010: 8, 0011: 4 0100: 44.1, 0101: 22.05, 0110: 11.025 0111: 5.5125 1000: 48, 1001: 24, 1010: 12, 1011: 6 1100 ~ 1111: N/A	R/W	0000
3	RST_FIFO	Reset FIFO Pointer 1: reset FIFO	R/W	0
2	FIFO_FULL	Info FIFO is full 0: FIFO not full, 1: FIFO full	R	0
1	FIFO_HALF	Info FIFO is half full 0: FIFO not half, 1: FIFO half	R	0
0	FIFO_EPT	Info FIFO is empty 0: FIFO not empty, 1: FIFO empty	R	0

ADC_FIFO (0x0028) : initial value = 0x0000

Bit	Function	Description	R/W	Reset Value
15:0	ADC_DATA	ADC input data	R	All “0”

Example :

```
u16 Read_ADC_REG(u16 ADC_reg_addr)
{
    u16 temp;
    _setSR(SFR_ADCSET1, ADC_reg_addr);          // Set register address
    _IObitCLR(SFR_ADCSET1, 15);                 // Set write_en is "0" (enable)
    temp = _getSR(SFR_ADCSET2);                 // Read the assignment register's value
    return temp;
}
```

```
u16 Write_ADC_REG(u16 ADC_reg_addr, u16 ADC_value)
{
	u16 temp;
	_IObitCLR(SFR_ADCSET1, 15);		// Set write_en is "0" (enable)
	_setSR(SFR_ADCSET1, ADC_reg_addr);	// Address
	_setSR(SFR_ADCSET2, ADC_value);		// Value
	_IObitSET(SFR_ADCSET1, 15);		// Set write_en is "1" (disable)
}

void main (void)
{
	_IObitSET(SFR_ADCFIFO, 15);		// Enable ADC Interface
	Write_ADC_REG(0x65, 0x00);		// Set Volume
	Write_ADC_REG(0x6B, 0xB0);		// Enable ADC Analog IP
	Write_ADC_REG(0x6C, 0x1F);		// Enable ADC Analog IP
	Write_ADC_REG(0x63, 0x36);		// Set ADC Gain

	_IObitSET(SFR_ADCFIFO, 3);		//Reset FIFO pointer

	_IObitCLR(SFR_ADCFIFO, 7);		// Set Sampling rate is 16KHz
	_IObitCLR(SFR_ADCFIFO, 6);
	_IObitCLR(SFR_ADCFIFO, 5);
	_IObitSET(SFR_ADCFIFO, 4);

	_IObitSET(SFR_INTEN, 8);		// Enable SD_ADC interrupt
	_IObitSET(SFR_INTEN, 15);		// Enable GIE

	while(1)
	{
		_IObitSET(SFR_WDT, 14);		// Clear WDT
	}
}

void __interrupt [0x0014] ADINT(void)   // ADC interrupt
{
	_IObitSET(SFR_INTCR, 8);

	for(i=0; i<8; i++)
	{
		BUF0[ADC_index] = _getSR(SFR_ADDATA);
		ADC_index++;
	}
}
```

3.6.3. ADC internal Register Table

Register Addr	BIT	LABEL	Default Value	Description
0x54	7:0	LB_H	0x40	AGC Control. Low bound setting for output amplitude of ADC: High byte
0x55	7:0	LB_L	0x00	AGC Control. Low bound setting for output amplitude of ADC: Low byte
0x56	7:0	HB_H	0x60	AGC Control. High bound setting for output amplitude of ADC: High byte
0x57	7:0	HB_L	0x00	AGC Control. High bound setting for output amplitude of ADC: Low byte
0x58	7:4	Reserved	0x00	
	3:0	NOR_POD	0x05	AGC Control. The period of gain update at normal mode when AGC is on. Fs : Sampling rate 0000: 1/Fs x 2^(0) 0001: 1/Fs x 2^(1) ……. 1110: 1/Fs x 2^(14) 1111: 1/Fs x 2^(15)
0x59	7:4	Reserved	0x00	
	3:0	MUTE_POD	0x0B	AGC Control. The period of gain update at mute mode when AGC is on. Fs : Sampling rate 0000: 1/Fs x 2^(0) 0001: 1/Fs x 2^(1) ……. 1110: 1/Fs x 2^(14) 1111: 1/Fs x 2^(15)
0x5A	7:0	SEARCH_TH_H	0x03	AGC Control Threshold for activating AGC. High byte.
0x5B	7:0	SEARCH_TH_L	0x00	AGC Control Threshold for activating AGC. Low byte.
0x5C	7:0	MUTE_TH_H	0x10	AGC Control Threshold for inactivating AGC. High byte
0x5D	7:0	MUTE_TH_L	0x00	AGC Control Threshold for inactivating AGC. Low byte
0x5E	7:4	Reserved	0x00	
	3:0	MUTE_CAL_POD	0x09	AGC Control.

				The calculating period for inactivating AGC. Fs : Sampling rate 0000: 256/Fs x 2^(0) 0001: 256/Fs x 2^(1) ……. 1110: 256/Fs x 2^(14) 1111: 256/Fs x 2^(15)
0x5F	7:4	Reserved	0x00	
	3:0	SAT_TH	0x03	AGC Control. Threshold for ADC saturation condition.
0x60	7:4	Reserved	0x00	
	3:0	SAT_POD	0x0A	AGC Control. The detection period for ADC saturation condition. Fs : Sampling rate 0000: 1/Fs x 2^(0) 0001: 1/Fs x 2^(1) ……. 1110: 1/Fs x 2^(14) 1111: 1/Fs x 2^(15)
0x61	7	AGC_OFF	0x01	AGC Control AGC function 0: Enable 1: Disable
	6:5	BOOST_SET_VAL	0x03	AGC Control Boost setting value at normal mode when AGC is on. 00: +0dB 01:+12dB 10: +20dB 11:+30dB
	4:0	PGA_SET_VAL	0x10	AGC Control PGA setting value at normal mode when AGC is on (1.5dB/step). 00000: Mute 00001: -12dB …… 01001: 0dB …… 11110: +31.5dB 11111: +33dB
0x62	7:3	Reserved	0x00	

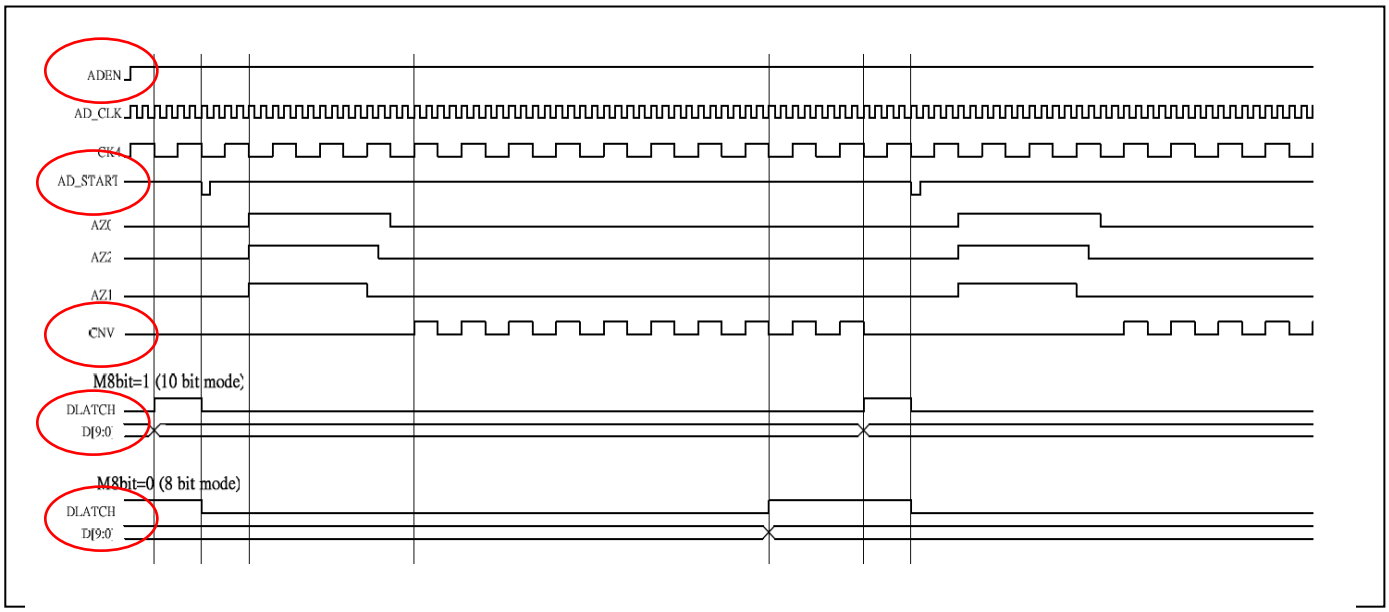
	2	ACTIVE	0x01	Digital Audio Interface Control. 0: Disable 1: Enable
	1:0	IWL	0x03	Word length of DAI. 00: 16-bits 01: 18-bits 10: 20-bits 11: 24-bits
0x63	7	Reserved	0x00	
	6:5	BOOST	0x00	Boost setting value when AGC is off 00: +0dB 01: +12dB 10: +20dB 11: +30dB
	4:0	PGA	0x09	PGA setting value when AGC is off 00000: Mute 00001: -12dB …… 01001: 0dB …… 11110: +31.5dB 11111: +33dB
0x64	7:6	Reserved	0x00	
	5	FILTER_OFF	0x01	Filter function for noise suppression 0: Enable 1: Disable
	4	ADCHPD	0x01	HPF function for offset cancellation 0: Enable 1: Disable
	3:0	HPGAIN	0x0B	HPF GAIN
0x65	7:4	VOL_CTRL	0x00	Digital Volume attenuation control. At the normal mode when AGC is on or off. 0000: 0dB 0001: -3dB 0010: -6dB 0011: -9dB 0100: -12dB 0101: -15dB 0110: -18dB 0111: -21dB 1000: -24dB 1001: -27dB 1010: -30dB 1011: -36dB 1100: -42dB 1101: -48dB 1110: -54dB 1111: -78dB
	3:0	MUTE_CTRL	0x00	Digital Volume attenuation control. At the mute mode when

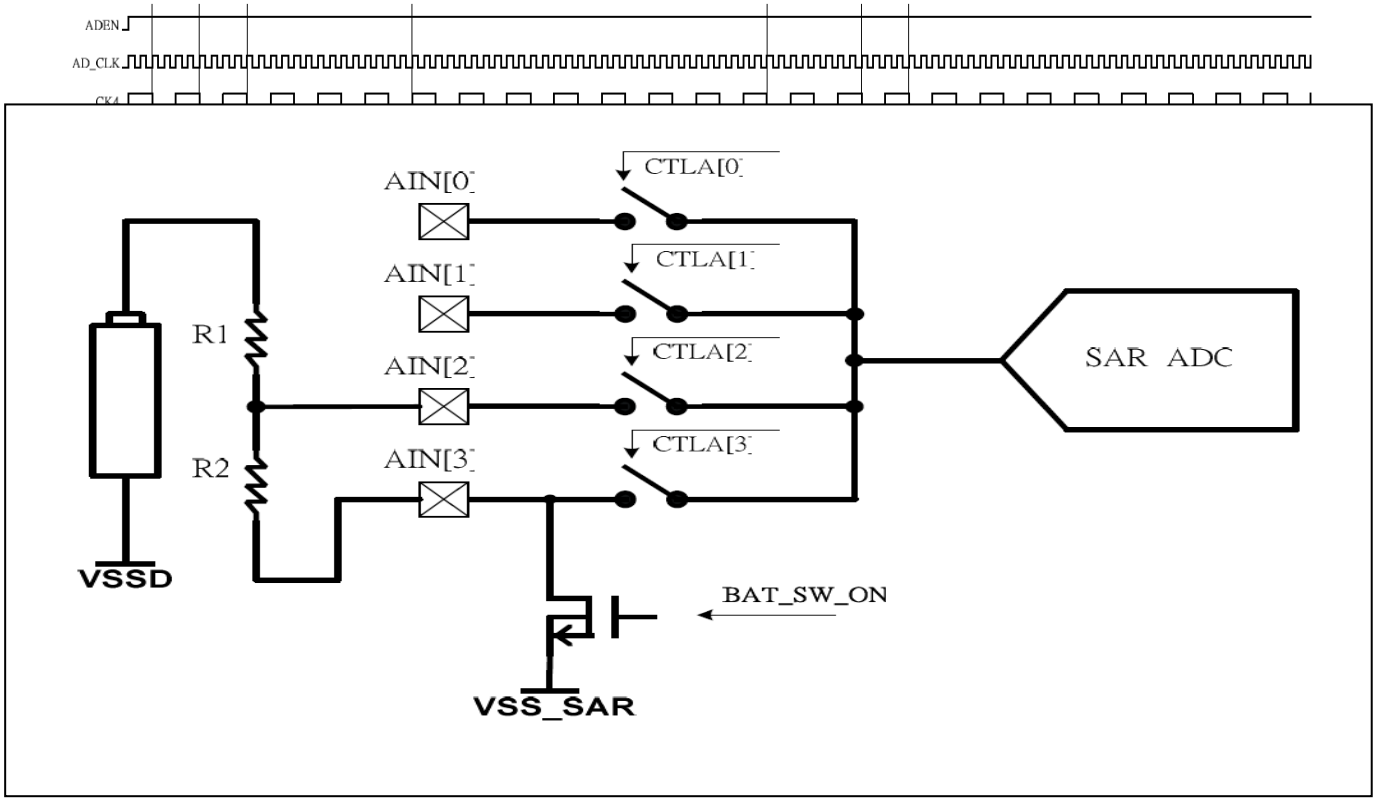
				AGC is on. 0000: 0dB 0001: -3dB 0010: -6dB 0011: -9dB 0100: -12dB 0101: -15dB 0110: -18dB 0111: -21dB 1000: -24dB 1001: -27dB 1010: -30dB 1011: -36dB 1100: -42dB 1101: -48dB 1110: -54dB 1111: -78dB
0x66	7	Reserved	0x00	
	6:5	BOOST_MUTE_VAL	0x03	AGC Control. Boost setting value at mute mode when AGC is on. 00: +0dB 01: +12dB 10: +20dB 11: +30dB
	4:0	PGA_MUTE_VAL	0x10	AGC Control. PGA setting value at mute mode when AGC is on. 00000: Mute 00001: -12dB …… 01001: 0dB …… 11110: +31.5dB 11111: +33dB
0x67	7:0	VOL_MUTE_POD	0x12	The updating period for Digital Volume attenuation at the normal/mute mode when AGC is on. Bit[7:4]: for normal mode. Bit[3:0]: for mute mode. Fs : Sampling rate 0000: 1/Fs x 2^(0) 0001: 1/Fs x 2^(1) ……. 1110: 1/Fs x 2^(14) 1111: 1/Fs x 2^(15)
0x68	7:0	Reserved	0x00	
0x69	7:0	Reserved	0x00	
0x6A	7:0	Reserved	0x00	
0x6B	7	ADC_EN	0	ADC power-on enable, active High
	6	LINE_EN	0	LINVOL power-on enable, active High
	5	MICBT_EN	0	MICBOOST power-on enable, active High

	4	PGA_EN	0	PGA power-on enable, active High
	3	DEMPD	0	DEM disable, active High
	2:0	LINVOL	1	Lin-in gain select (8-level) 000: -6dB 001: +0dB 010: +3dB 011: +6dB 100: +9dB 101: +12dB 110: +15dB 111: +20dB
0x6C	7:5	Reserved	0x00	
	4	DX_EN	0	SDM data to IC external enable, active High
	3	IREF_EN	0	IREF circuit enable, active High
	2	VREF_EN	0	VREF circuit enable, active High
	1	MICB_EN	0	Microphone bias enable, active High
	0	CK_EN	0	CKGEN enable, active High
0x6D	7	Reserved	0	
	6:5	SEL_IB	0	Internal Bias Current select 00: 30uA 01: 40uA 10: 40uA 11: 50uA
	4	SEL_MICB	0	Microphone Bias Output select 0: 0.8*VA 1: 0.9*VA
	3	SEL_LINE	0	Line input structure select 0: AUX 1: FMIN
	2	SEL_MIC	0	Microphone input structure select 0: Differential 1: Single-end
	1	SEL_MIX_LINE	0	Line input path to mixer enable 0: Disable 1: Enable
	0	SEL_MIX_MIC	1	Microphone input path to mixer enable 0: Disable 1: Enable
0x6E	7	Reserved	0x00	
	6:5	BOOST AGC	-	Boost setting value when AGC is on. ***Read Only***
	4:0	PGA AGC	-	PGA setting value when AGC is on. ***Read only***

3.7. SAR ADC

This analog to digital converter has 4-input sources with up to 1024-step resolution to transfer analog signal into 10-bits digital data. The sequence of ADC operation is to select an input source (AIN0 ~ AIN3) first, then set CHS and START bit to “1” to start conversion. When the conversion is complete, the ADC circuit will set START bit to “0” and the final value output will be in the ADR register. The SNP70032 provide an interrupt to inform user program that an ADC result is ready. The interrupt event is optional.





ADP (0x007D) : initial value = 0x0000

Bit	Function	Description	R/W	Reset Value

15:12	CHS	analog input channel select bit 0000 : all analog input disable 1000 : AIN0 is analog input 1001 : AIN1 is analog input 1010 : AIN2 is analog input 1011 : AIN3 is analog input 1100 : Input is VSS_SAR (analog) 1101 : Input is AVREFH (analog)	R/W	0000
11:8	--	Reserved	--	000
7	AIN3_DI	1: AIN3 is digital input pin (AIN3 isn't analog input)	R/W	0
6	AIN2_DI	1: AIN2 is digital input pin (AIN2 isn't analog input)	R/W	0
5	AIN1_DI	1: AIN1 is digital input pin (AIN1 isn't analog input)	R/W	0
4	AIN0_DI	1: AIN0 is digital input pin (AIN0 isn't analog input)	R/W	0
3	AIN3_PH	1: AIN3 pull-high enable (AIN3 is digital input)	R/W	0
2	AIN2_PH	1: AIN2 pull-high enable (AIN2 is digital input)	R/W	0
1	AIN1_PH	1: AIN1 pull-high enable (AIN1 is digital input)	R/W	0
0	AIN0_PH	1: AIN0 pull-high enable (AIN0 is digital input)	R/W	0

ADM (0x007E) : initial value = 0x0000

Bit	Function	Description	R/W	Reset Value
15	SAR_ADC_EN	SAR ADC Enable control bit 0: disable, 1: enable	R/W	0
14	ADC_LEN	SAR ADC Resolution select 0: 8-bit, 1: 10-bit	R/W	0
13	BAT_SW_ON	1: AIN3 will auto tie to GND by H/W	R/W	0
12	--	Reserved	--	0
11:9	CONV_RATE_SEL[2:0]	Set the SAR ADC conversion rate (CR) ADC_LEN = 0 (8-bit solution) CR = 250K / (2 ^ CONV_RATE_ SEL) / 12 ADC_LEN = 1 (10-bit solution) CR = 250K / (2 ^ CONV_RATE_ SEL) / 14	R/W	000
8:4	--	Reserved	--	0
3	AUTO	1: HW auto clear start signal	R/W	0
2	--	Reserved	--	0
1	VALID	SAR ADC output data if valid 0: invalid, 1: valid	R	0
0	START	SAR ADC Start control bit 0: disable, 1: enable	R/W	0

ADR (0x007F) : initial value = 0x0000 (if ADC_LEN = 0)

Bit	Function	Description	R/W	Reset Value
15:8	ADC_DATA	SAR ADC output data	R	All “0”
7:0	--	H/W always fill in “0”	R	All “0”

ADR (0x007F) : initial value = 0x0000 (if ADC_LEN = 1)

Bit	Function	Description	R/W	Reset Value
15:6	ADC_DATA	SAR ADC output data	R	All “0”
5:0	--	H/W always fill in “0”	R	All “0”

Example :

```
void main (void)
{
        _IObitSET(SFR_ADP, 15);                 // Select AIN3
        _IObitCLR(SFR_ADP, 14);
        _IObitSET(SFR_ADP, 13);
        _IObitSET(SFR_ADP, 12);

        _IObitSET(SFR_ADM, 14);                 // Select resolution is 10-bit

        _IObitCLR(SFR_ADM, 11);                 // Select conversion rate is 17.85KHz
        _IObitCLR(SFR_ADM, 10);
        _IObitCLR(SFR_ADM, 9);

        _IObitSET(SFR_ADM, 15);                 // SAR ADC enable

        _IObitSET(SFR_ADM, 0);                  // SAR ADC start

        _IObitSET(SFR_INTEN2, 8);               // Enable SAR ADC Interrupt
        _IObitSET(SFR_INTEN, 15);               // Enable GIE
}

void __interrupt [0x0068] SARADINT(void)        // SAR ADC interrupt
{
        _IObitSET(SFR_INTCR2, 8);
        BUF0[ADC_index] = _getSR(SFR_ADR);
        ADC_index++;
        if(ADC_index == 0x100)
        {
                ADC_index = 0;
        }
        _IObitSET(SFR_ADM, 0);                  // SAR ADC Start
}
```

3.8. MSP (Main Serial Port)

The MSP (Main Serial Port) is a serial communication interface for data exchanging from one MCU to another MCU or other hardware peripherals. These peripheral devices may be serial EEPROM, A/D converters, Display device, etc. The MSP module can operate in one of two modes

- Full Master Mode
- Slave Mode (with general address call)

The MSP features include the following:

2-wire synchronous data transfer / receiver.

Master (SCL is clock output) or Slave (SC is clock input) operation.

SCL, SDA are programmable open-drain output pin for multiple salve devices application.

Support 400K clock rate @ Fcpu=48MIPs.

End-of-Transfer/Receiver interrupt.

3.8.1. MSP STATUS REGISTER

MSPSTAT (0x0062) : initial value = 0x0000

Bit	Function	Description	R/W	Reset Value
15:7	--	Reserved	--	0
6	CKE	Slave Clock Edge Control bit In Slave Mode: Receive Address or Data byte 0: Latch Data on SCL Rising Edge 1: Latch Data on SCL Falling Edge	R/W	0
5	D/A	Data / Address bit 0: Indicates the last byte received or transmitted was address 1: Indicates the last byte received or transmitted was data	R	0
4	P	Stop bit 0: Stop bit was not detected 1: Indicates that a stop bit has been detected last ***It will be cleared when Start bit was detected***	R	0
3	S	Start bit 0: Start bit was not detected. 1: Indicates that a start bit has been detected last ***It will be cleared when Stop bit was detected***	R	0
2	RD/WR	Read/Write bit information This bit holds the R/W bit information following the	R	0

		last address match. This bit is only valid from the address match to the next start bit, stop bit, or not ACK bit. Slave mode 0: Write 1: Read Master mode 0: Transmit is not in progress. 1: Transmit is in progress Or this bit with SEN, RSEN, PEN, RCEN, or ACKEN will indicate if the MSP is in IDLE mode.		
1	--	Reserved	--	0
0	BF	Buffer Full Status bit Receive 1: Receive complete, MSPBUF is full 0: Receive not complete, MSPBUF is empty Transmit 1: Data Transmit in progress (does not include the ACK and stop bits), MSPBUF is full 0: Data Transmit complete (does not include the ACK and stop bits), MSPBUF is empty	R	0

3.8.2. MSP MODE REGISTER 1

MSPM1 (0x0063) : initial value = 0x0000

Bit	Function	Description	R/W	Reset Value
15:8	--	Reserved	--	0
7	WCOL	Write Collision Detect bit Master Mode 0: No collision 1: A write to the SSPBUF register was attempted while the MSP conditions were not valid for a transmission to be started Slave Mode 0: No collision 1: The SSPBUF register is written while it is still transmitting the previous word (must be cleared in software)	R/W	0

6	WSPOV	Receive Overflow Indicator bit 1: A byte is received while the SSPBUF register is still holding the previous byte. SSPOV is a “don’t care” in transmit mode. SSPOV must be cleared in software in either mode. (must be cleared in software) 0: No overflow.	R/W	0
5	WSPENB	Master Serial Communication Enable. 0: Disables serial port and configures these pins as I/O port pins 1: Enables serial port and configures SCL & SDA pins as MSP port pins	R/W	0
4	CKP	SCL Clock Priority Control bit Slave mode 0: Hold SCL keeping Low. (Ensure data setup time and Slave device ready.) 1: Release SCL Clock (Slave Transistor mode CKP function always enable, Slave Receiver CPK function control by SLRXCKP) Master mode: Unused	R/W	0
3	SLRXCKP	Slave Receiver mode SCL Clock Priority Control bit Slave Receiver mode 0: Enable CKP function. 1: Disable CKP function. Master and Slave Transistor mode Unused.	R/W	0
2:1	--	Reserved	--	0
0	MSPC	MSP mode Control register 0: MSP operated on Master mode. 1: MSP operated Slave mode, 7-bit address	R/W	0

3.8.3. MSP MODE REGISTER 2

MSPM2 (0x0064) : initial value = 0x0000

Bit	Function	Description	R/W	Reset Value
15:8	--	Reserved	--	0
7	GCEN	General Call Enable bit (In Slave mode only) 0: General call address disabled	R/W	0

		1: Enable interrupt when a general call address (0000h) is received.		
6	ACKSTAT	Acknowledge Status bit (In master mode only) master transmit mode 0: Acknowledge was received from slave 1: Acknowledge was not received from slave	R/W	0
5	ACKDT	Acknowledge Data bit. (In master mode only) master receive mode Value that will be transmitted when the user initiates an Acknowledge sequence at the end of a receive. 0: Send ACK during Acknowledge 1: Send NACK during Acknowledge	R/W	0
4	ACKEN	Acknowledge Sequence Enable bit (In MSP master mode only) master receive mode 0: Acknowledge sequence idle 1: Initiate Acknowledge sequence on SDA and SCL pins, and transmit AKDT data bit. Auto clear by hardware ***If the MSP module is not in the idle mode, this bit may not be set (no spooling), and the MSPBUF may not be written (or writes to the MSPBUF are disabled)***	R/W	0
3	RCEN	Receive Enable bit (In master mode only) 0: Receive idle 1: Enables Receive mode for MSP ***If the MSP module is not in the idle mode, this bit may not be set (no spooling), and the MSPBUF may not be written (or writes to the MSPBUF are disabled)***	R/W	0
2	PEN	Stop Condition Enable bit (In master mode only) 0: Stop condition idle 1: Initiate Stop condition on SDA and SCL pins. Automatically cleared by hardware. ***If the MSP module is not in the idle mode, this bit may not be set (no spooling), and the MSPBUF may not be written (or writes to the MSPBUF are disabled)***	R/W	0

1	RSEN	Repeated Start Condition Enabled bit (In master mode only) 0: Repeated Start condition idle. 1: Initiate Repeated Start condition on SDA and SCL pins. Automatically cleared by hardware ***If the MSP module is not in the idle mode, this bit may not be set (no spooling), and the MSPBUF may not be written (or writes to the MSPBUF are disabled)***	R/W	0
0	SEN	Start Condition Enabled bit (In master mode only) 0: Start condition idle 1: Initiate Start condition on SDA and SCL pins. Automatically cleared by hardware ***If the MSP module is not in the idle mode, this bit may not be set (no spooling), and the SSPBUF may not be written (or writes to the SSPBUF are disabled)***	R/W	0

3.8.4. MSP Buffer register

MSPBUF(0x0065) : initial value = 0x0000

Bit	Function	Description	R/W	Reset Value
15:8	--	Reserved	--	0
7:0	MSPBUF[7:0]	MSP Buffer	R/W	0

3.8.5. MSP ADDRESS register

MSPADR(0x0066) : initial value = 0x0000

Bit	Function	Description	R/W	Reset Value
15:8	--	Reserved	--	0
7:0	MSPADR[7:0]	MSP Address	R/W	0

NOTE: In Master Mode the MSPADR register holds the baud rate generator reload value.
In Slave Mode the MSPADR register holds the slave device address, only MSPADR [7:1] are relevant.

3.8.6. Slave Mode Operation

When an address is matched or after a data transfer and address match is received, the hardware will automatically generate an acknowledge (ACK_) signal, and load MSPBUF (MSP buffer register) with the received data from MSPSR.

There are some conditions that will cause MSP function to not respond with an ACK_ signal:

1. Data Buffer already full: BF=1 (MSPSTAT bit 0), when another transfer was received.
2. Data Overflow: MSPOV=1 (MSPM1 bit 6), when another transfer was received

When BF=1, it means MSPBUF data has still not been read by the MCU, so the MSPSR will not load data into MSPBUF, but MSPIRQ and MSPOV bit will still set to 1. BF bit will be clear automatically when reading the MSPBUF register. MSPOV bit must be cleared by Software.

3.8.7. Addressing

When MSP Slave function has been enabled, it will wait until a START signal occurs. Following the START signal, 8-bits received data will shift into the MSPSR register. The data of MSPSR[7:1] is compared with MSPADR[7:1] on the falling edge of eight SCL pulses. If the address are the same, the BF and SSPOV bit are both clear, the following events occur:

1. MSPSR register is loaded into MSPBUF on the falling edge of eight SCL pulse.
2. Buffer full bit (BF) is set to 1, on the falling edge of eight SCL pulse.
3. An ACK_ signal is generated.
4. MSP interrupt request MSPIRQ is set on the falling edge of ninth SCL pulse.

Status when Data is Received		MSPSP→ MSPBUF	Reply an ACK_ signal	Set MSPIRQ
BF	MSPOV			
0	0	Yes	Yes	Yes
*0	*1	Yes	No	Yes
1	0	No	No	Yes
1	1	No	No	Yes

Data Received Action Table

Note: BF=0, MSPOV=1 shows the software is not set properly to clear Overflow register.

3.8.8. Slave Receiving

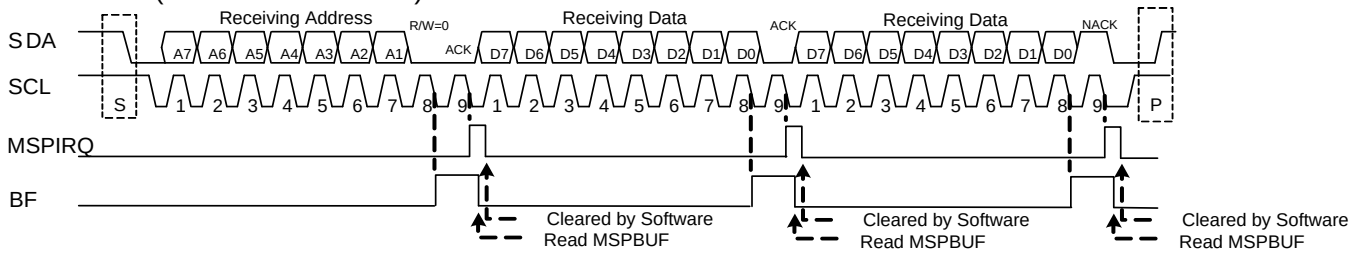
When the R/W bit of address byte =0 and the address is matched, the R/W bit of MSPSTAT is cleared. The address will be loaded into MSPBUF. After replying an ACK_ signal, MSP will receive data every 8 clocks. The CKP function enable or disable (Default) is controlled by SLRXCKP bit and data latch edge -Rising edge (Default) or Falling edge is controlled by CPE bit.

When an overflow occurs, no acknowledge signal is replied which means either BF=1 or MSPOV=1.

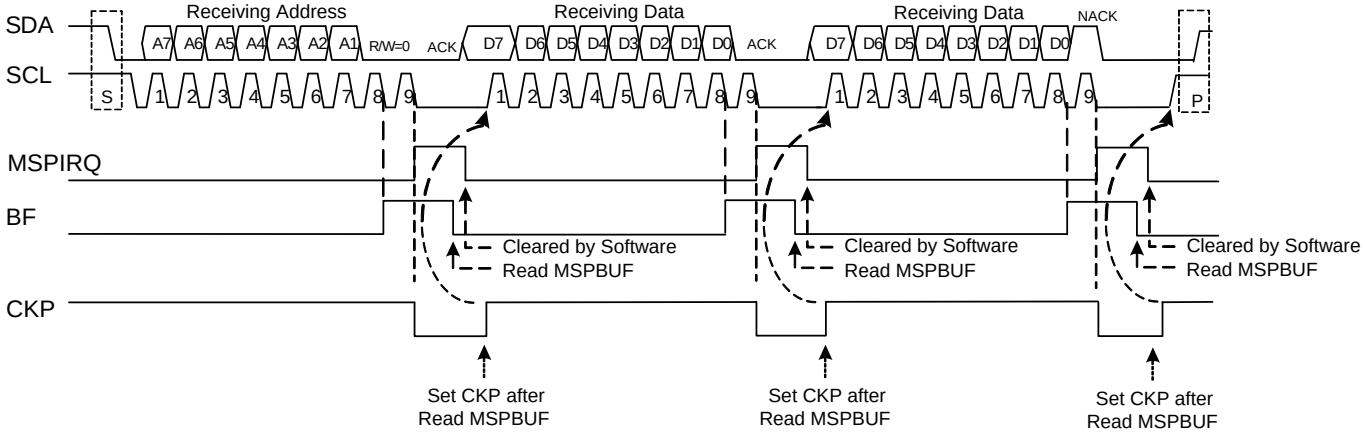
MSP interrupt is generated in every data transfer. The MSPIRQ bit must be cleared by software.

Following is the Slave Receiving Diagram

SLRXCKP=1 (Disable CKP function)



SLRXCKP=0 (Enable CKP function)

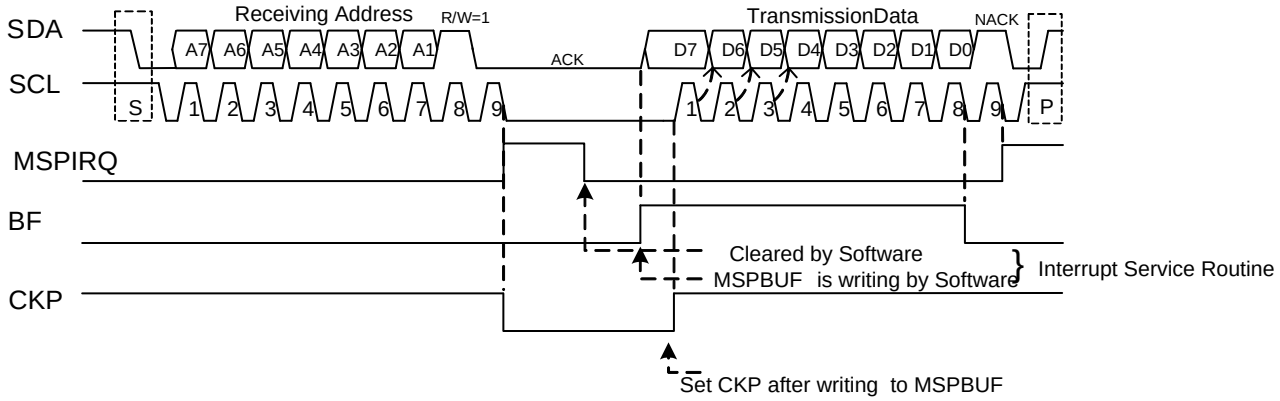


3.8.9. Slave Transmission

After an address match, MSPSTAT bit 2 R/W will be set. The received address will be loaded into MSPBUF and an ACK_ will be sent at the ninth clock then SCL will be held low. Transmission data will be loaded into MSPBUF which also loads to MSPSR register. The Master should monitor SCL pin signal. The slave device may hold the master by keeping CKP low. After load MSPBUF, set CKP bit, MSPBUF data will shift out on the falling edge on SCL signal. This will ensure the SDA signal is valid on the SCL high duty.

An MSP interrupt is generated on every byte transmission. The MSPIRQ will be set on the ninth clock of SCL. Clear MSPIRQ by software. MSPSTAT register can monitor the status of data transmission.

In Slave transmission mode, an ACK_ signal from master-receiver is latched on rising edge of ninth clock of SCL. If ACK_ = high, transmission is complete. Slave device will reset logic and waiting for another START signal. If ACK_= low, slave must load MSPBUF which also MSPSR, and set CKP=1 to start data transmission again.



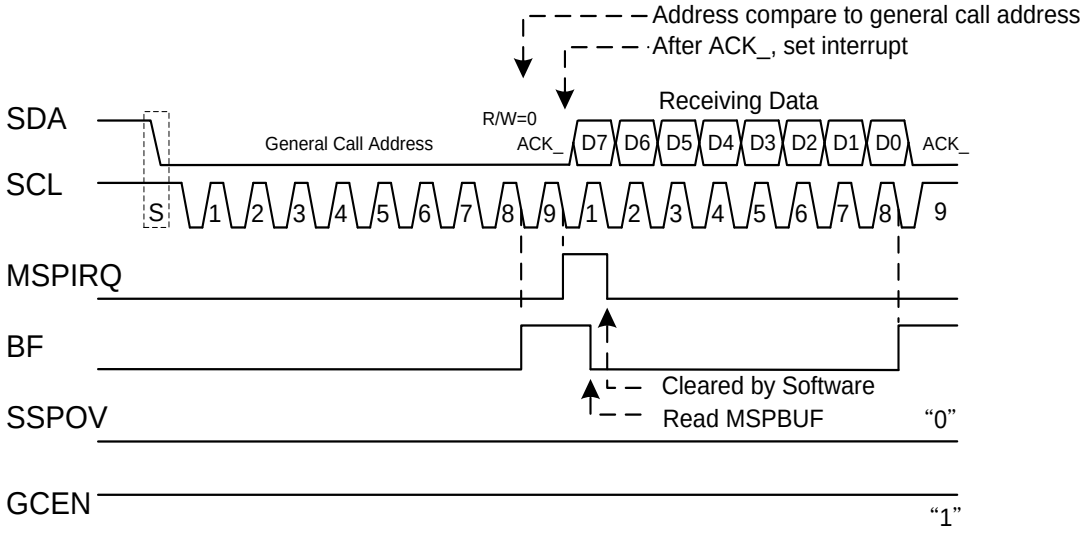
MSP Slave Transmission Timing Diagram

3.8.10. General Call Address

In MSP bus, the first 7-byte is the Slave address. Only the address match MSPADDR the Slave will response an ACK_. The exception is the general call address which can address all slave devices. When this address occur, all devices should response an acknowledge signal.

The general call address is a special address which is reserved as all “0” of 7-bytes address. The general call address function is control by GCEN bit. Set this bit will enable general call address and clear it will disable. When GECN=1, following a START signal, 8-bit will shift into MSPSR and the address is compared with MSPADD and also the general call address which fixed by hardware.

If the general call address matches, the MSPSR data is transferred into MSPBUF, the BF flag bit is set, and in the falling edge of the ninth clock (ACK_) MSPIRQ flag set for interrupt request. In the interrupt service routine, reading MSPBUF can check if the address is the general call address or device specific.



General Call Address Timing Diagram

3.8.11. Master Mode

Master mode of MSP operation from a START signal and end by STOP signal.

The START (S) and STOP (P) bit are clear when reset or MSP function disabled.

In Master mode the SCL and SDA line are controlled by MSP hardware.

Following events will set MSP interrupt request (MSPIRQ), if MSPIEN set, interrupt occurs.
- Data byte transmitted or received
- Acknowledge Transmit.

3.8.12. Mater Mode Support

Master mode enable when MSPC and MSPENB bit set. Once the Master mode enabled, the user had following six options.
- Send a START signal on SCL and SDA line.
- Send a Repeat START signal on SCL and SDA line.
- Write to MSPBUF register for Data or Address byte transmission
- Send a STOP signal on SCL and SDA line.
- Configuration MSP port for receive data
- Send an Acknowledge at the end of a received byte of data.

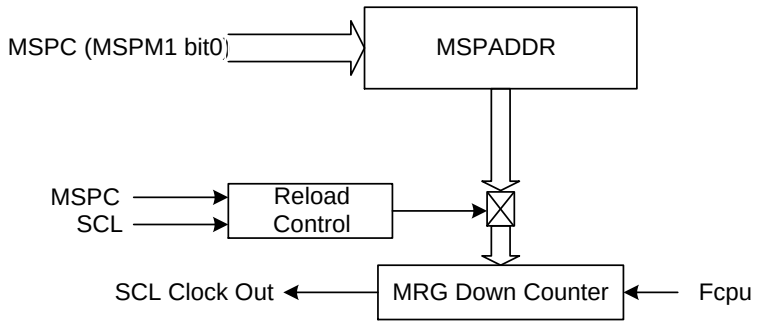
3.8.13. MSP Rate Generator

In Master Mode, the MSP rate generator's reload value is located in the lower 7 bit of MSPADDR register. When MRG is loaded with the register, the MRG count down to 0 and stop until another reload has taken place. In MSP mater mode MRG reload from MSPADDR automatically. If Clock Arbitration occur for instance (SCL pin keep low by Slave device), the MRG will reload when SCL pin is detected High.

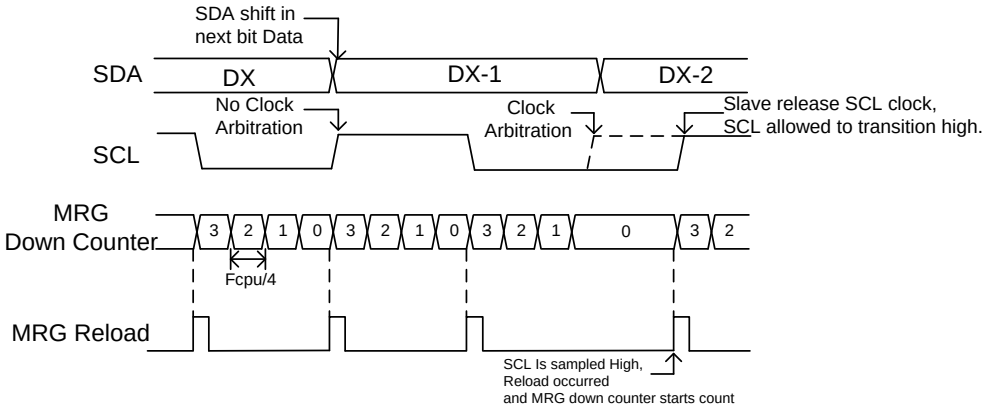
SCL clock rate = Fcpu / (MSPADDR*2)

For example, if we want to set 400Khz in 48Mhz Fcpu, the MSPADDR register have to set to 0x3Ch.

MSPADDR=48Mhz / (400Khz*2) = 60



MSP Rate Generator Block Diagram



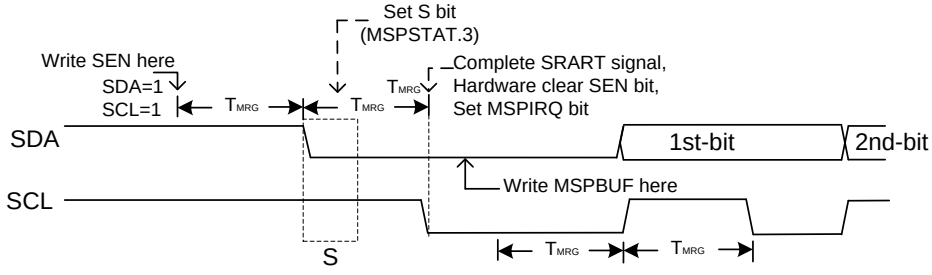
MRG Timing Diagram with and without Clock Arbitration (MSPADRR=0x03)

3.8.14. MSP Mater START Condition

To generate a START signal, user sets SEN bit (MSPM2.0). When SDA and SCL pin are both sampled High, MSP rate generator reload MSPADDR[6:0], and starts down counter. When SDA and SCL are both sampled high and MRG overflow, SDA pin is drive low. When SCL sampled high, and SDA transmitted from High to Low is the START signal and will set S bit (MSPSTAT.3). MRG reload again and start down counter. SEN bit will be clear automatically when MRG overflow, the MRG is suspend leaving SDA line held low, and START condition is complete.

WCOL Status Flag

If user write to MSPBUF when START condition processing, then WCOL is set and the content of MSPBUF data is un-changed. (the writer doesn't occur)



START Condition Timing Diagram

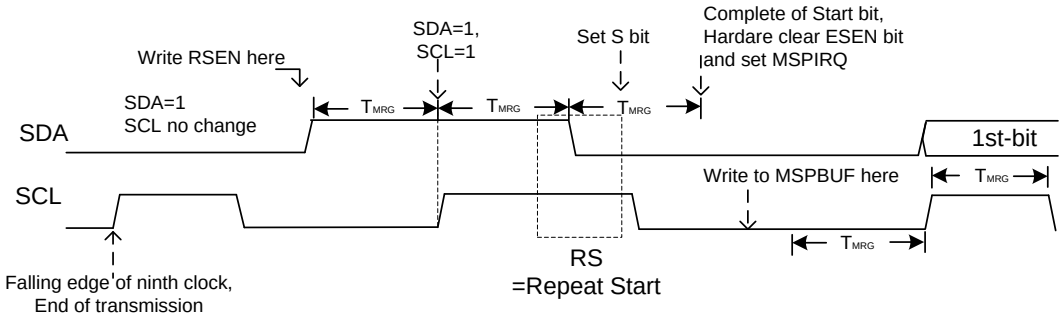
3.8.15. MSP Master mode Repeat START Condition

When MSP logic module is idle and RSEN set to 1, Repeat Start progress occurs. RSEN set and SCL pin is sampled low, MSPADDR[6:0] data reload to MSP rate generator and start down counter. The SDA pin is release to high in one MSP rate generate counter (T_{MRG}). When the MRG is overflow, if SDA is sampled high. SCL will be brought high. When SCL is sampled high, MSPADDR reload to MRG and start down counter. SDA and SCL must keep high in one T_{MRG} period. In the next T_{MRG} period, SDA will be brought low when SCL is sampled high, then RSEN will clear automatically by hardware and MRG will not reload, leaving SDA pin held low. Once detect SDA and SCL occur START condition, the S bit will be set (MSPSTAT.3). MSPIRQ will not set until MRG overflow.

- Note: 1. While any other event is progress, Set RSEN will take no effect.
- Note:2. A bus collision during the Repeat Start condition occur:
  - ***SDA is sampled low when SCL goes from low to high***

WCOL Status Flag

If user write to MSPBUF when Repeat START condition processing, then WCOL is set and the content of MSPBUF data is un-changed. (the writer doesn't occur)



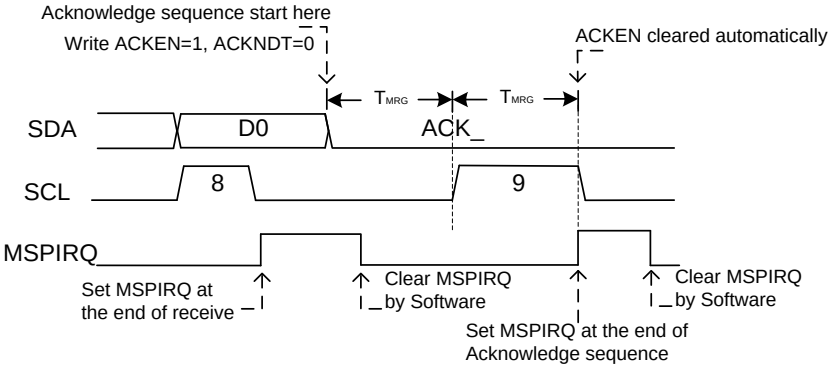
Repeat Start Condition Timing Diagram

3.8.16. Acknowledge Sequence Timing

An acknowledge sequence is enabled when set ACKEN (MSPM2.4). SCL is pulled low when set ACKEN and the content of the acknowledge data bit is present on SDA pin. If user whished to reply a acknowledge signal , ACKDT bit should be cleared. If not, set ACKDT bit before starting a acknowledge sequence. SCL pin will be release (brought high) when MSP rate generator overflow. MSP rate generator start a T_{MRG} period down counter, when SCL is sampled high. After this period, SCL is pulled low, and ACKEN bit is clear automatically by hardware. When next MRG overflow again, MSP goes into idle mode.

WCOL Status Flag

If user write to MSPBUF when Acknowledge sequence processing, then WCOL bit is set and the content of MSPBUF data is un-changed. (the writer doesn't occur)



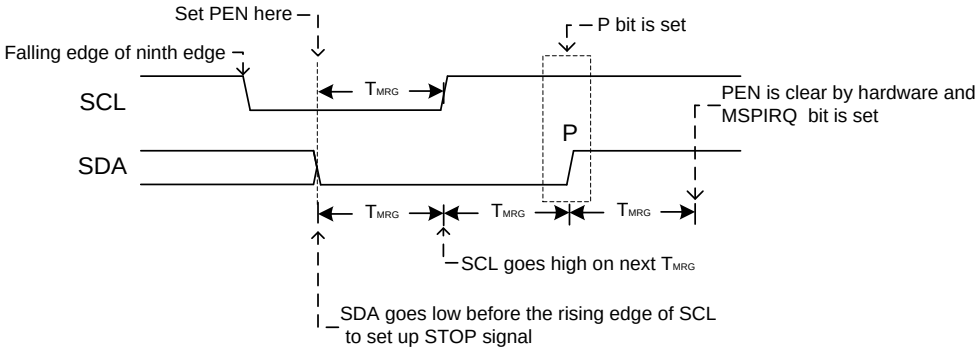
Acknowledge Sequence Timing Diagram

3.8.17. STOP Condition Timing

At the end of received/transmitted, a STOP signal present on SDA pin by setting the STOP bit register, PEN (MSPM2.1). At the end of receive/transmit, SCL goes low on the failing edge of ninth clock. Master will set SDA go low, when set PEN bit. When SDA is sampled low, MSP rate generator is reloaded and start count down to 0. When MRG overflow, SCL pin is pull high. After one T_{MRG} period, SDA goes to High. When SDA is sampled high while SCL is high, bit P is set. PEN bit is clear after next one T_{MRG} period, and MSPIRQ is set.

WCOL Status Flag

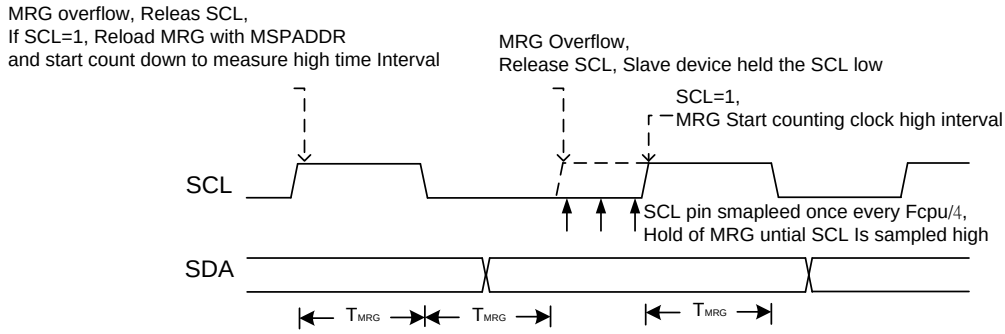
If user write to MSPBUF when a STOP condition is processing, then WCOL bit is set and the content of MSPBUF data is un-changed. (the writer doesn't occur)



STOP condition sequence Timing Diagram

3.8.18. Clock Arbitration

Clock arbitration occurs when the master, during any receive, transmit or Repeat START, STOP condition that SCL pin allowed to float high. When SCL pin is allowed float high, the master rate generator (MRG) suspended from counting until the SCL pin is actually sampled high. When SCL is sampled high, the MRG is reloaded with the content of MSPADDR[6:0], and start down counter. This ensure that SCL high time will always be at least one MRG overflow time in the event that the clock is held low by an external device.



Clock Arbitration sequence Timing Diagram

3.8.19. Master Mode Transmission

Transmission a data byte, 7-bit address or the eight bit data is accomplished by simply write to MSPBUF register. This operation will set the Buffer Full flag BF and allow MSP rate generator start counting.

After write to MSPBUF, each bit of address will be shifted out on the falling edge of SCL until 7-bit address and R/W_ bit are complete. On the failing edge of eighth clock, the master will pull low SDA fort slave device respond with an acknowledge. On the ninth clock falling edge, SDA is sampled to indicate the address already accept by slave device. The status of the ACK bit is load into ACKSTAT status bit. Then MSPIRQ bit is set, the BF bit is clear and the MRG is hold off until another write to the MSPBUF occurs, holding SCL low and allow SDA floating.

BF Status Flag

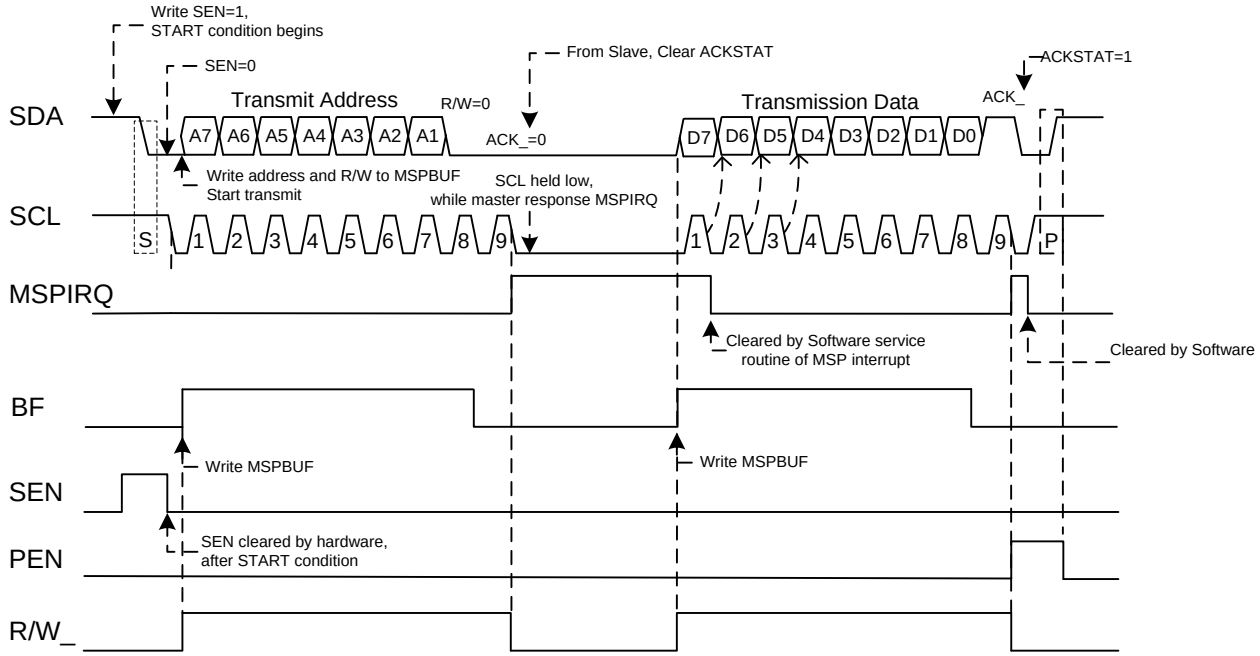
In transmission mode, the BF bit is set when user writes to MSPBUF and is cleared automatically when all 8 bit data are shift out.

WCOL Flag

If user write to MSPBUF during Transmission sequence in progress, the WCOL bit is set and the content of MSPBUF data will unchanged.

ACKSTAT Status Flag

In transmission mode, the ACKSTAT bit is cleared when the slave has sent an acknowledge (ACK_=0), and is set when slave does not acknowledge (ACK_=1). A slave send an acknowledge when it has recognized its address (including general call), or when the slave has properly received the data.



MSP Master Transmission Mode Timing Diagram

3.8.20. Master Mode Receiving

Master receiving mode is enable by set RCEN bit.

The MRG start counting and when SCL change state from low to high, the data is shifted into MSPSR. After the falling edge of eighth clock, the receive enable bit (RCEN) is clear automatically, the contents of MSP are load into MSPBUF, the BF flag is set, the MSPIRQ flag is set and MRG counter is suspended fro, counting, holding SCL low. The MSP is now in IDLE mode and awaiting the next operation command. When the MSPBUF data is read by Software, the BF flag is cleat automatically. By setting ACKEN bit, user can send an acknowledge bit at the end of receiving.

BF Status Flag

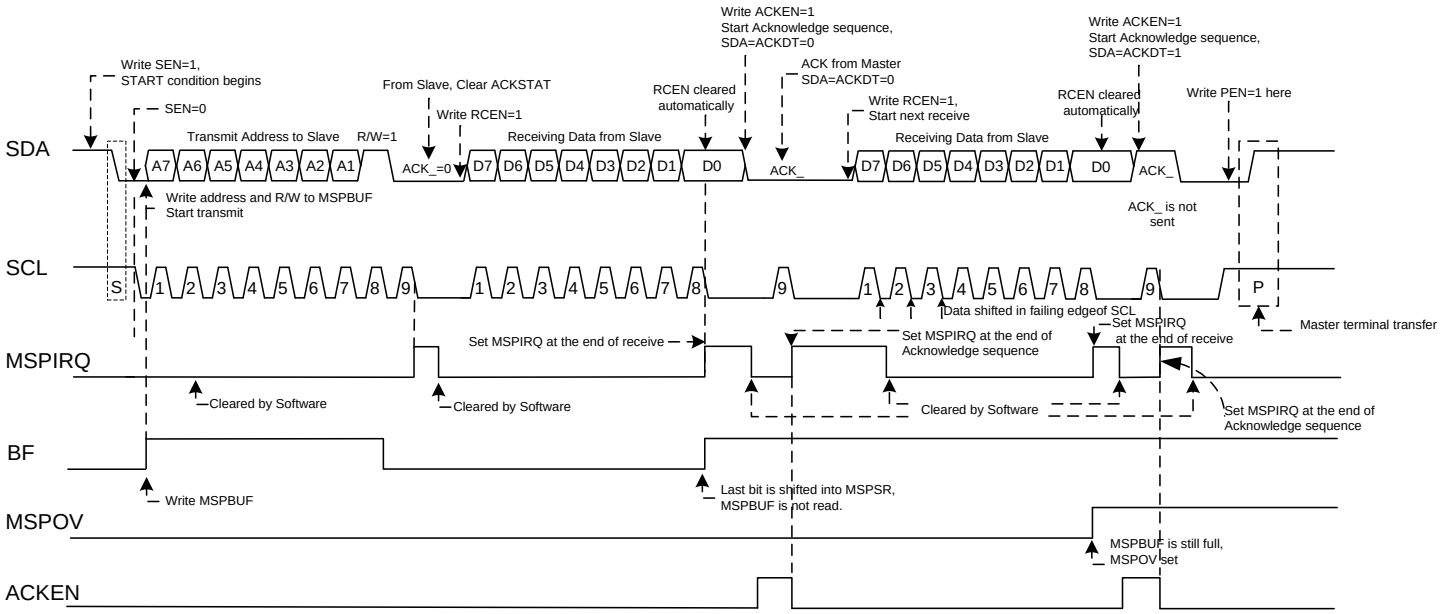
In Reception mode, the BF bit is set when an address or data byte is loaded into MSPBUF from MSPSR. It is cleared automatically when MSPBUF is read.

MSPOV Flag

In receive operation, the MSPOV bit is set when another 8-bit are received into MSPSR, and the BF bit is already set from previous reception

WCOL Flag

If user write to MSPBUF when a receive is already progress, the WCOL bit is set and the content of MSPBUF data will unchanged.



MSP Master Receiving Mode Timing Diagram

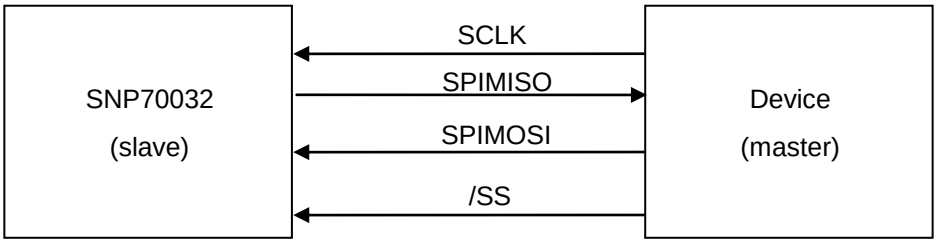
3.9. SPI Interface

The SPI is managed by the DSP. The SPI (serial peripheral interface) is a synchronous serial bus that provides good support for communication with SPI-compatible peripheral devices.

The SPI peripheral is a synchronous, 7-wire interface consisting of two data pins (SPIMOSI and SPIMISO); two additional pins (SPIED3 and SPIED2) for 4-bit mode access, one slave select pins (/SS1); and a synchronous clock pin (SCLK). The two data pins permit full-duplex and half-duplex operation to other SPI-compatible devices. The SPI also includes programmable baud rates, clock phase (CPHA), and clock polarity (CPOL).

Devices communicate using a master/slave relationship, in which the master initiates the data transfer by writing to the transmit data buffer (SPIDATA0~5) and bits to be transfer (SPITRANSFER). When the master generates a clock and selects a slave device, data may be transferred in both directions simultaneously. In the master SPI, the bits are sent out of the SPIMOSI pin and received in the SPIMISO pin. The bits to be shifted out are stored in the SPI internal shift register SPIDATA[0:5], and are sent out most significant bit (bit 7) first . When bit 7 of the master is shifted out through SPIMOSI pin, a bit from bit 7 of the slave is being shifted into bit 0 of the master via the SPIMISO pin. After 8 clock pulses or shifts, this bit will eventually end up in bit 7 of the master. *If there is no more data written to the transmit data buffer (SPITxDR), the SPI enters idle state, /SS is idle state refers to user' setting, and SCLK stops until another data is written.*

Master Mode



Slave Mode

3.9.1. Interface Pins

1. SPI Clock Pin (SCLK)

SPI clock signal (SCLK) is driven by the master and controls the rate at which data is transferred. The master may transmit data at a variety of baud rates. SCLK cycles once for each bit transmitted. This pin is shared with P0.7

SCLK is used to synchronize shifting out and shifting in data driven on the SPIMISO and SPIMOSI pins. According with user' setting, data is shifted out on one edge of the clock and sampled on the other clock edge. The clock polarity and clock phase relative to data are programmable in the SPI Control (SPICTRL) registers.

2. SPI Slave Select Pin (/SS)

SPI Slave Select signal (CS1) from the master is an active low output signal used to enable a slave device. User can choose if /SS is high active or not by setting SPICSC register. This pin is shared with P0.10

3. SPI Transmit Data Pin (SPIMISO)

Data shifts out by SPIMISO pin. This pin is shared with P0.8

4. SPI Receive Data Pin (SPIMOSI)

Data shifts in by SPIMOSI pin. This pin is shared with P0.9

3.9.2. SPI Control Register (SPICTRL)

The SPI Control registers set the bit transfer rate for a master device.

SPICTRL (0x005D) : initial value = 0x0000

Bit	Function	Description	R/W	Reset Value
15	RWMODE	SPI read / write mode control bit 0: write, 1: read	R/W	0
14	MSMODE	SPI master / slave mode control bit 0: master, 1: slave	R/W	0
13:12	SPIBIT_MODE	Select SPI transmission bit 00: 1 bit transfer mode (Normal) 01: 2 bit transfer mode (Dual) 10: 4 bit transfer mode (Quad) 11: Invalid	R/W	00
11:10	--	Reserved	--	00
9	CPHA	Clock Phase select 0: Data change at clock falling, latch at clock rising when CPOL = 0; Data change at clock rising, latch at clock falling when CPOL = 1 1: Data change at clock rising, latch at clock falling when CPOL = 0; Data change at clock falling, latch at clock rising when CPOL = 1	R/W	0

8	CPOL	Clock idle polarity 0: Clock idles at low, 1: Clock idles at high	R/W	0
7	SPIEN	SPI control enable control bit 0: disable, 1: enable	R/W	0
6:1	--	Reserved	--	All “0”
0	START	Set this bit to START SPI transfer/received, auto clear after operation finished 0: Stop, 1: Start	R/W	0

Please note that data is transferred MSB first.

SPI Transfer Format

	CPOL	CPHA
Mode 0	0	0
Mode 1	0	1
Mode 2	1	0
Mode 3	1	1

SPI (1/2/4 bit) I/O Mode

SPIBIT_MODE	Master			Slave		
	RWMode	I/O Name	I/O Mode	RWMode	I/O Name	I/O Mode
00 (1 bit)	R	SPIED3	-	R	SPIED3	-
		SPIED2	-		SPIED2	-
		SPIMISO	I		SPIMISO	O
		SPIMOSI	O		SPIMOSI	I
	W	SPIED3	-	W	SPIED3	-
		SPIED2	-		SPIED2	-
		SPIMISO	I		SPIMISO	O
		SPIMOSI	O		SPIMOSI	I
01 (2 bit)	R	SPIED3	-	R	SPIED3	-
		SPIED2	-		SPIED2	-
		SPIMISO	I		SPIMISO	I
		SPIMOSI	I		SPIMOSI	I
	W	SPIED3	-	W	SPIED3	-
		SPIED2	-		SPIED2	-
		SPIMISO	O		SPIMISO	O
		SPIMOSI	O		SPIMOSI	O
10 (4 bit)	R	SPIED3	I	R	SPIED3	I
		SPIED2	I		SPIED2	I
		SPIMISO	I		SPIMISO	I
		SPIMOSI	I		SPIMOSI	I

	W	SPIED3	O	W	SPIED3	O
		SPIED2	O		SPIED2	O
		SPIMISO	O		SPIMISO	O
		SPIMOSI	O		SPIMOSI	O

DAUL Read/Write Data Format:

Clock	0	1	2	3	4	5	6	7
SPIMISO	Bit 7	Bit 5	Bit 3	Bit 1	Bit 7	Bit 5	Bit 3	Bit 1
SPIMOSI	Bit 6	Bit 4	Bit 2	Bit 0	Bit 6	Bit 4	Bit 2	Bit 0

QUAD Read/Write Data Format:

Clock	0	1	2	3	4	5	6	7
SPIED3	Bit 7	Bit 3	Bit 7	Bit 3	Bit 7	Bit 3	Bit 7	Bit 3
SPIED2	Bit 6	Bit 2	Bit 6	Bit 2	Bit 6	Bit 2	Bit 6	Bit 2
SPIMISO	Bit 5	Bit 1	Bit 5	Bit 1	Bit 5	Bit 1	Bit 5	Bit 1
SPIMOSI	Bit 4	Bit 0	Bit 4	Bit 0	Bit 4	Bit 0	Bit 4	Bit 0

3.9.3. SPI Baud Rate Register (SPIBR)

The SPI Baud Rate (SPIBR) registers set the bit transfer rate for a master device.

SPIBR (0x0061) : initial value = 0x0000

Bit	Function	Description	R/W	Reset Value
15:11	--	Reserved	--	0
10:8	SPIPRS	Set SPI baud rate parameter 1	R/W	000
7:0	SPIDIV	Set SPI baud rate parameter 2	R/W	All “0”

SPIPRS	Pre-Scale
000	2
001	4
010	8
011	16
100	32
101	64
110	128
111	256

The SPI baud rate frequency is determined by the following formula

=> SPI baud rate = (System clock) / (Pre_Scale*(SPIDIV+1))

***Note: When SPI is Slave mode, the maximum running speed is 12MHz*.**

3.9.4. SPIDATABuf (SPI Transmit and Receive Data Buffer)

SPIDATA0 (0x0057) ~ SPIDATA5 (0x005C) : initial value = 0x00XX

Bit	Function	Description	R/W	Reset Value
15:8	--	Reserved	--	0
7:0	DATA[7:0]	SPI data	R/W	X

These registers are used as the transfer and receive data buffer for SPI. For each operation, all the data on those registers will be shifted out to the SPITxD pin one by one, at the same time, data will be shifted in from the SPIRxD pin. After the operation has finished, user can examine the data from these registers.

3.9.5. SPI bit Transfer (SPITRANSFER)

This register is used to identify the amount of bits from the data buffer to be transferred.

SPITRANSFER (0x005F) : initial value = 0x0000

Bit	Function	Description	R/W	Reset Value
15:6	--	Reserved	--	0
5:0	SPI_TF[5:0]	SPI Transfer bit number	R/W	All “0’

Note: When SPI is Slave mode, the register has to be set to indicate how many bits of data will be received.

3.9.6. SPI Chip Select Control (SPICSC)

User can control enable and polarity of chip select pin by Hardware or software. In Hardware control CS, the CS polarity is specified by the CSPOL control bit, which selects an active high or active low. In software control CS, user can use SW_CS control bit to control one cycle running time of SPI. The chip selects (CS1) control bit is to choose the CS pin is controlled by HW or SW. On the other hand, in slave mode, CS1 is Chip select pin of SNP70032 and it always low active.

SPICSC (0x005E) : initial value = 0x0000

Bit	Function	Description	R/W	Reset Value
15	--	Reserved	--	0
14	SW_CS1	CS1 software control bit 0: CS1 is “0”, 1: CS1 is “1”	R/W	0
13:5	--	Reserved	--	All “0”
4	CSPOL1	Chip1 Select Polarization (CS1) 0: CS1 active low, 1: CS1 active high	R/W	0

3:1	--	Reserved	--	00
0	CS1	Hardware or Software control Chip1 Select 0: SW control CS1 selected 1: HW control CS1 selected	R/W	0

Example: (Please refer to the SNP70032 Application Note)

```
void main(void)
{
    unsigned int uiTemp1;

    _setSR(SFR_WDT, (SET_BIT1 | SET_BIT0));          // Set WDT 2 Sec

    // Set GPIO P0.11 as "Output High" if users connect Serial Flash WP#/SIO2 pin to it
    _IObitSET(SFR_P0En, 11);
    _IObitSET(SFR_P0M, 11);
    _IObitSET(SFR_P0, 11);

    // Set GPIO P0.12 as "Output High" if users connect Serial Flash Hold#/SIO3 pin to it
    _IObitSET(SFR_P0En, 12);
    _IObitSET(SFR_P0M, 12);
    _IObitSET(SFR_P0, 12);

    // Initial SPI Controller
    SPI_HW_CS1_Setting(0);              // Set CS1 control by HW and active low, idle high
    SPI_Set_Baud_Rate(0);               // Set Baud Rate 24MHz
    SPI_Set_Interface_Mode(0);          // SPI Mode 0
    SPI_Set_Access_Mode(0,1);           // Write mode, 1-bit access
    SPI_Enable(0);                      // Enable SPI Control as Master Mode

    // Demo Serial Flash Read Status Register Command
    // CS1 control by software
    SPI_Buffer[0] = 0x05;               // Serial Flash Read Status Command
    SPI_Buffer[1] = 0;                  // Send Receive Data clock
    SPI_SW_CS1_Signal(0);               // Software Control CS1 signal low to active mode
    uiTemp1 = SPI_Access_Data(SPI_Buffer, 16);
    SPI_SW_CS1_Signal(1);               // Software Control CS1 signal High to idle mode
                                        // SPI_Buffer[1] = Received Data (Status Value)
    // Serial Flash Read RDID Command
    // CS1 control by hardware Demo
    SPI_HW_CS1_Setting(0);              // Set CS1 control by HW and active low, idle high
    SPI_Buffer[0] = 0x9F;               // Serial Flash RDID Command
    uiTemp1 = SPI_Access_Data(SPI_Buffer, 32); // SPI_Buffer[1] = Received Data (Manufacturer ID)
                                        // SPI_Buffer[2] = Received Data (Device ID1)
                                        // SPI_Buffer[3] = Received Data (Device ID2)
    // CS1 control by software Demo
    SPI_Set_Access_Mode(0,1);           // Write mode, 1-bit access
```

```
    SPI_Buffer[0] = 0x03;                       // Serial Flash Read Command
    SPI_Buffer[1] = 0x00;                       // Serial Flash Address A23~A16
    SPI_Buffer[2] = 0x00;                       // Serial Flash Address A15~A8
    SPI_Buffer[3] = 0x20;                       // Serial Flash Address A7~A0
    SPI_Buffer[4] = 0x00;                       // Send cycle for receive data

    SPI_SW_CS1_Signal(0);                       // Software Control CS1 signal low
    uiTemp1 = SPI_Access_Data(SPI_Buffer, 40);
    SPI_SW_CS1_Signal(1);                       // Software Control CS1 signal High
                                                // SPI_Buffer[4] => Received Data
}
```

3.10. Serial Flash Controller

The Serial Flash Controller Features

- 1/2/4 bits read mode and 1/4 bits write mode
- 6MHz,12MHz , 24MHz , 48MHz four kinds of clock frequency
- Program on SPI Flash
- DMA access

Reg_SF_CTRL (0xF800) : initial value = 0x1488

Bit	Function	Description	R/W	Reset Value
15	SPIF_EN	SPI Flash controller enable bit 0: disable, 1: enable	R/W	0
14	QUAD	SPI Flash Quad mode enable control bit 0: disable, 1: enable	R/W	0
13:12	CLK_DIV	SPI Flash clock select (SysClk = 48MHz) 00: SysClk (48MHz), 01: SysClk/2 (24MHz) 10: SysClk/4 (12MHz), 11: SysClk/8 (6MHz)	R/W	01
11:8	--	Reserved	--	0100
7	READ	Read data from SPI Flash via 1 I/O 0: N/A, 1: Command = 0x03 to read data	R/W	1
6	FAST_READ	Read data from SPI Flash via 1 I/O 0: N/A, 1: Command = 0x0B to read data	R/W	0
5	2READ	Read data from SPI Flash via 2 I/O 0: N/A, 1: Command = 0xBB to read data	R/W	0
4	4READ	Read data from SPI Flash via 4 I/O 0: N/A, 1: Command = 0xEB to read data	R/W	0
3	WP	Word Program to SPI Flash via 1 I/O 0: N/A, 1: Commmand = 0x02 to write data ***Unit is Word***	R/W	0
2	4WP	Word Program to SPI Flash via 4 I/O 0: N/A, 1: Commmand = 0x38 to write data ***Unit is Word***	R/W	0
1	BP	Byte Program to SPI Flash via 1 I/O 0: N/A, 1: Commmand = 0x02 to write data ***Unit is Byte***	R/W	0
0	PP	Page (128 Word) Program to SPI Flash via 1 I/O 0: N/A, 1: Commmand = 0x02 to write data	R/W	0

Note : DMA read mode (bit7 ~ bit4) just only one can be selected.
DMA write mode (bit3 ~ bit0) just only one can be selected.
bit0 (page program), SNC7110 does not support.

Reg_SF_CMD (0xF801) : initial value = 0x0000

Bit	Function	Description	R/W	Reset Value
15	READ_ID	Read SPI Flash factory ID 0: N/A, 1: Command = 0x9F to read ID	R/W/C	0
14	READ_SR	Read SPI Flash Status Register 0: N/A, 1: Command = 0x05 to read SPI SR	R/W/C	0
13	DP	SPI Flash Deep Power Down control 0: N/A, 1: Command = 0xB9 to deep power down	R/W/C	0
12	RDP	SPI Flash Release Deep Power Down control 0: N/A, 1: Command = 0xAB to release deep power down	R/W/C	0
11	WRITE_SR	Write data to SPI Flash Status Register 0: N/A, 1: Command = 0x01 to write SPI SR	R/W/C	0
10	SECTOR_ERASE	SPI Sector Erase control bit 0: N/A, 1: Command = 0x20 to execute sector erase	R/W/C	0
9	BLOCK_ERASE	SPI Block Erase control bit 0: N/A, 1: Command = 0xD8 to execute block erase	R/W/C	0
8	CHIP_ERASE	SPI Chip Erase control bit 0: N/A, 1: Command = 0xC7 to execute chip erase	R/W/C	0
7:2	--	Reserved	--	All “0”
1	SPI_STATUS	SPI Flash controller status 0: ready, 1: busy	R	0
0	--	Reserved	--	0

Example: (Please refer to the SNP70032_Application Note)

```
void main(void)
{
    unsigned int uiTemp1;

    _setSR(SFR_WDT, (SET_BIT1 | SET_BIT0));          // Set WDT 2 Sec

    // Enable Serial Flash Controller
    SF_Enable();

    // Read Serial Flash Factory ID
```

```
    uiTemp1 = SF_Read_Factory_ID();

    // Serial Flash Chip Erase
    SF_Chip_Erase();

    // Waiting for Chip Erase complete, users can add timeout protect by yourself
    while (SF_Erase_Data_Polling());                    // Serial Flash Read Status Command

    for (uiTemp1 = 0; uiTemp1 < 0x100; uiTemp1++)
    {
        Write_Data_Array[uiTemp1] = uiTemp1;
    }

    // Set 1x I/O Write via DMA
    SF_Write_DMA_Mode(1, 0x0000, 0x0000, 0x0100, Write_Data_Array);

    // Waiting for DMA write complete, users can add timeout protect by yourself.
    while (SF_Write_DMA_Data_Polling());

    // Set 2x I/O Read via DMA
    SF_Read_DMA_Mode(2, 0x0000, 0x0000, 0x0100, Read_Data_Array);

    // Set 4x I/O Write via DMA
    SF_Write_Status(0x40);                              // Write Serial Flash “Quad” bit to 1
    SF_Enable_Quad_Mode( );
    SF_Write_DMA_Mode(4, 0x0000, 0x0100, 0x0100, Write_Data_Array);

    // Waiting for DMA write complete, users can add timeout protect by yourself.
    while (SF_Write_DMA_Data_Polling());

    // Set 4x I/O Read via DMA
    SF_Read_DMA_Mode(4, 0x0000, 0x0100, 0x0100, Read_Data_Array);
    SF_Disable_Quad_Mode();                             // Disable SF Controller support Quad Mode
}
```

3.11. PWM output

The PWM is managed by the DSP. PWM provides two I/O pins (P0.3 to P0.6) capable for output of 256 levels PWM pulse. The source clock of this PWM comes from Timer 2. Please be aware that user must set Timer2 as auto load mode after 256 count before activating PWM I/O.

PWM0 (0x0051) : initial value = 0x0000

Bit	Function	Description	R/W	Reset Value
15:9	--	Reserved	--	All “0”
8	PWM0EN	Enable PWM0 output on P0.3	R/W	0
7:0	PWM0C	Control the output pulse duration of PWM 0: 0/256, 1: 1/256, 2: 2/256, …, 255: 255/256	R/W	All “0”

PWM1 (0x0052) : initial value = 0x0000

Bit	Function	Description	R/W	Reset Value
15:9	--	Reserved	--	All “0”
8	PWM1EN	Enable PWM1 output on P0.4	R/W	0
7:0	PWM1C	Control the output pulse duration of PWM 0: 0/256, 1: 1/256, 2: 2/256, …, 255: 255/256	R/W	All “0”

PWM2 (0x0053) : initial value = 0x0000

Bit	Function	Description	R/W	Reset Value
15:9	--	Reserved	--	All “0”
8	PWM2EN	Enable PWM2 output on P0.5	R/W	0
7:0	PWM2C	Control the output pulse duration of PWM 0: 0/256, 1: 1/256, 2: 2/256, …, 255: 255/256	R/W	All “0”

PWM3 (0x0054) : initial value = 0x0000

Bit	Function	Description	R/W	Reset Value
15:9	--	Reserved	--	All “0”
8	PWM3EN	Enable PWM3 output on P0.6	R/W	0
7:0	PWM3C	Control the output pulse duration of PWM 0: 0/256, 1: 1/256, 2: 2/256, …, 255: 255/256	R/W	All “0”

Example:

```
void main (void)
{
	_setSR(SFR_T2, _getSR(SFR_T2) | 0x1F00);		// Set T2 Auto reload and Pre scalar 512(/512)
                                                // And enable T2
	_setSR(SFR_T2CNT, 0x00FF);                  // Set T2 Counter value.
	_setSR(SFR_PWM1, 0x0180);                   // Set PWM_1 Enable and output pulse.
	_setSR(SFR_PWM2, 0x01FF);                   // Set PWM_2 Enable and output pulse.
	while(1)
	{
		_setSR(SFR_WDT, _getSR(SFR_WDT) | 0x4000);		// Clr_WDTIRQ;
	}
}
```

3.12. I2S Interface

SNP70032 provides a built-in I2S interface.

(1) Only supports Master Mode (Transmitter / Receiver)

(2) Main Clock fixed = 256fs

(3) Channel bit clock fixed = 32 bits

I2SCTRL (0x0050) : initial value = 0x0000

Bit	Function	Description	R/W	Reset Value
15	I2SEN	I2S Interface control bit 0: disable (P1.6 ~ P1.9 is GPIO pin) 1: enable (P1.6 ~ P1.9 is I2S pin)	R/W	0
14:4	--	Reserved	--	All “0”
3:2	DFSEL	Set I2S format 00: I2S format 01: Left Justfield format 10: Right Justfield format 11: N/A	R/W	00
1	TR	Set I2S Transfer mode 0: Transmitter, 1: Receiver	R/W	0
0	WLCP	Set I2S serial data word length 0: word length is 16-bit 1: word length is 8-bit	R/W	0

PS : How to control I2S FIFO and set sample rate , Please reference Audio DAC FIFO register (0x004B)

Example: Please refer to SNP70032 Application Note

```
void main (void)
{
	_setSR(SFR_WDT, (SET_BIT1 | SET_BIT0));		// Set WDT 2 Sec
	I2S_TR_Select(1);				// Transmitter
	I2S_Format_Select(2);				// I2S_MODE
	I2S_Word_Length_Select(1);			// 16 bit
	I2S_Sample_Rate_Select(1);			// 16KHz
	I2S_FIFO_Control(1);				// FIFO Half
	I2S_Reset_FIFO(1);				// Setero
	I2S_Enable();					// Enable I2S Interface
	I2S_Trigger_Start();
	while(1)
	{
		_setSR(SFR_WDT, (SET_BIT14));
	}
}

void __interrupt [0x0040] I2SINT(void)  // I2S interrup
```

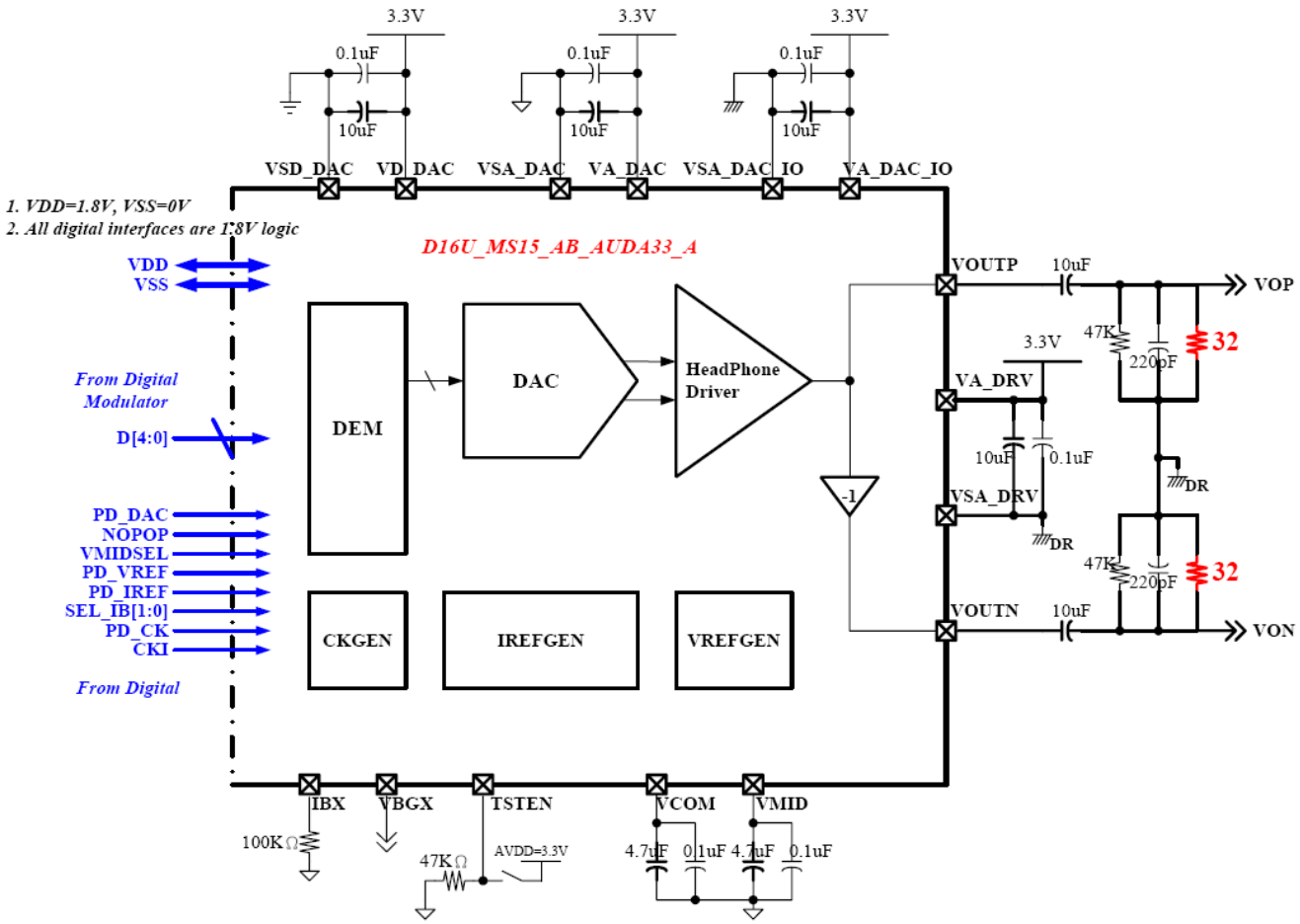
```
{
	unsigned int uiLoop, uiValue;

	_IObitSET(SFR_INTCR, 12);				// Clear I2S INTRQ
	for (uiLoop = 0; uiLoop < 8 ;uiLoop++)
	{
		uiValue = I2S_CH_L[uiLoop];
		_setSR(SFR_DAOL, uiValue);
		uiValue = I2S_CH_R[uiLoop];
		_setSR(SFR_DAOR, uiValue);
	}
}
```

3.13. Audio DAC

3.13.1. Sigma delta Stereo DAC

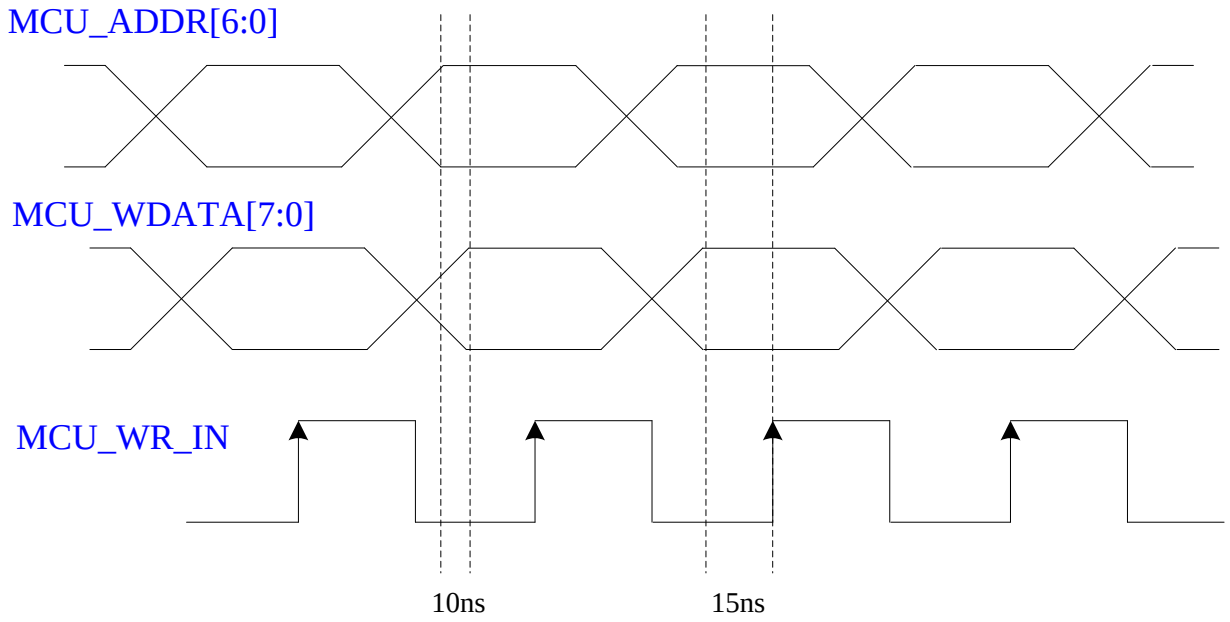
The sigma delta DAC is managed by the DSP. The DAC will convert digital audio signal to analog audio signal.



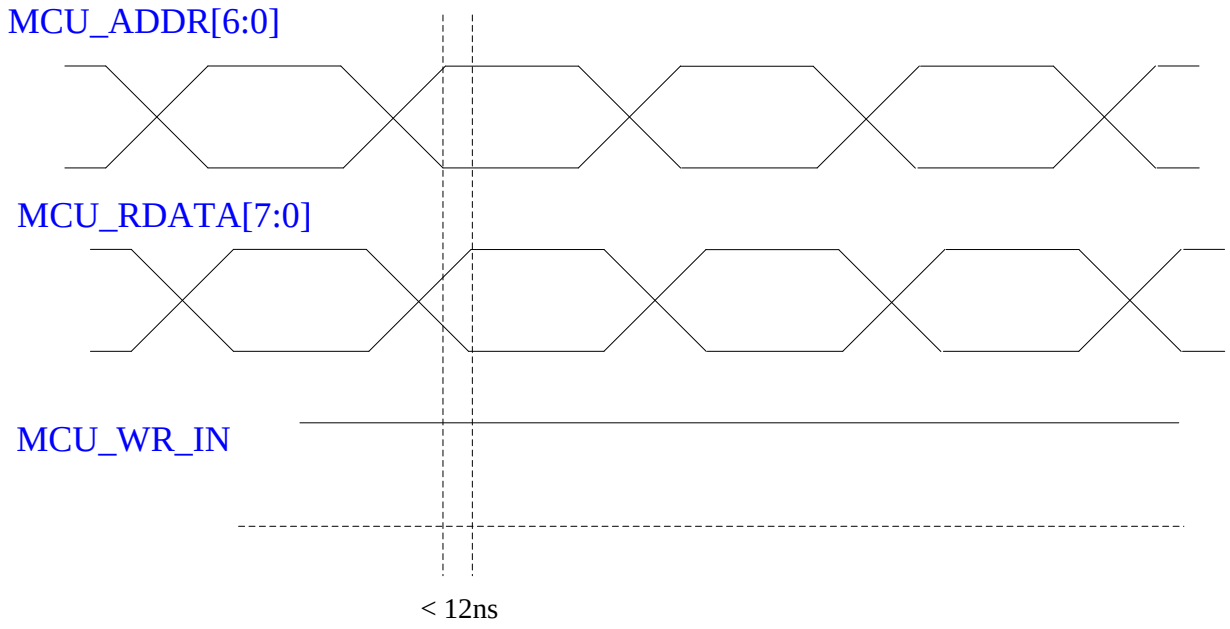
3.13.2. Sigma delta DAC control signal

In SNP70032, DSP core uses a MCU interface to read or write sigma-delta DAC internal register. The following is the timing diagram.

MCU Write Mode



MCU Read Mode



DACSET1 (0x0049) : initial value = 0x0000

Bit	Function	Description	R/W	Reset Value
15	MCU_WR_IN	MCU Interface Read/Write control bit 0: write, 1:read	R/W	0
14:7	--	Reserved	--	All “0”
6:0	MCU_ADDR[6:0]	MCU Interface Address bit	R/W	All “0”

DACSET2 (0x004A) : initial value = 0x0000

Bit	Function	Description	R/W	Reset Value
15	INI_RAM		R	0
14:8	--	Reserved	--	All “0”
7:0	MCU_DATA[7:0]	MCU Interface Data bit	R/W	All “0”

3.13.3. Sigma delta DAC FIFO

SNP70032 equipment two sets 16-level FIFO for DAC channel. H.W will automatically output FIFO data to DAO register when FIFO Timer is time out

DAC_FIFOStatus (0x004B) : Initial Value = 0x0200

Bit	Function	Description	R/W	Reset Value
15	TRG_DAC	Enable Sigma Delta DAC controller 0: disable, 1: enable	R/W	0
14	TRG_I2S	Trigger I2S signal 0: disable, 1: enable	R/W	0
13:11	--	Reserved	--	All “0”
10:9	DAC_INT_MODE	Sigma Delta DAC interrupt triggered is by FIFO status 00: interrupt trigger by FIFO full 01: interrupt trigger by FIFO half 10: interrupt trigger by FIFO empty 11: N/A	R/W	01
8	--	Reserved	--	0
7:4	DAC_SR	Set Sigma Delta DAC Sampling Rate (KHz) 0000: 32, 0001: 16, 0010: 8, 0011: 4 0100: 44.1, 0101: 22.05, 0110: 11.025 0111: 5.5125 1000: 48, 1001: 24, 1010: 12, 1011: 6	R/W	0000

		1100 ~ 1111: N/A		
3	RST_FIFO	Reset FIFO Pointer 1: reset FIFO	R/W	0
2	FIFO_FULL	Info FIFO is full 0: FIFO not full, 1: FIFO full	R	0
1	FIFO_HALF	Info FIFO is half full 0: FIFO not half, 1: FIFO half	R	0
0	FIFO_EPT	Info FIFO is empty 0: FIFO not empty, 1: FIFO empty	R	0

DAOL (0x0055) : initial value = 0x0000

Bit	Function	Description	R/W	Reset Value
15	DAOL[15:0]	Sigma Delta DAC Left Channel Output	R/W	All “0”

DAOR (0x0056) : initial value = 0x0000

Bit	Function	Description	R/W	Reset Value
15	DAOR[15:0]	Sigma Delta DAC Right Channel Output	R/W	All “0”

3.13.4. Sigma delta DAC internal Register Table

Register Addr	BIT	LABEL	Default Value	Description
0x00	7	PD_DAC	1	0: Active DAC IP 1: Power down DAC IP
	6	PD_CLK	1	0: Active CKGEN 1: Power down CKGEN
	5	PD_IREF	1	0: Active IREF 1: Power down IREF
	4:3	SEL_IB[1:0]	01	Internal Bias current select 00: 40uA, 01: 50uA, 10: 50uA, 11: 60uA
	2	PD_VREF	1	0: Active VREF 1: Power down VREF
	1	VMIDSEL	0	Normal mode / Fast start-up mode select 0: Normal mode, 1: Fast start-up mode
	0	NOPOP	1	Headphone output POP noise elimination 0: Internal 5KΩ, 1: Internal 1KΩ
0x01	7:6	RMP[1:0]	00	Attenuation ramp rate

	5	I2S_MODE	1	Select I2S Input data mode or ADO_DST data mode 1 : I2S Input data mode 0 : ADO_DST data mode
	4	BIT_S_IN	0	Select I2S Data format 16 bits or 24 bits 1: 24-bits 0: 16-bits
	3	DEML_IN	0	De-emphasis ON/OFF 1 : De-emphasis on 0 : De-emphasis off
	2	MUTX	1	Mute ON/OFF 1: Mute on 0 : Mute Off
	1	DAC_EN_IN	0	DAC Enable 1: Enable 0: Disable
	0	SOFT_RSTN	1	Software reset digitial circuit. Low reset. One MCLK pulse trigger
0x02	7:0	VOL[7:0]	00000000	Volume control 0dB ~ -96dB, -0.375dB/step
0x03	7	PD_DRV	1	0: Active DAC Driver IP 1: Power down DAC Driver IP
	6:3	Reserved	00000	
	2:1	DEMS[1:0]	00	Select the DAC de-emphasis response curve 0 - Reserved 1 - De-emphasis for 48 kHz 2 - De-emphasis for 44.1 kHz 3 - De-emphasis for 32 kHz
	0	INI_RAM_EN	0	Initial ram enable signal

3.13.5. Fractional PLL

PLL_CTRL (0xF300) : initial value = 0x0000

Bit	Function	Description	R/W	Reset Value
15:3	--	Reserved	--	All “0”
2	DIV	XTIN Divide 0: XTIN/1 1: XTIN/2,	R/W	0
1	SDM_EN	SDM Enable or not 0: Disable	R/W	0

		1: Enable		
0	PLL_PD	Fractional PLL Power Down or not 0: Enable 1: Power down	R/W	0

INT (0xF301) : initial value = 0x0000

Bit	Function	Description	R/W	Reset Value
15:4	--	Reserved	--	All “0”
3:0	INT[3:0]	Integer number for SDM	R/W	0

FRA_L (0xF302) : initial value = 0x0000

Bit	Function	Description	R/W	Reset Value
15:0	FRA [15:0]	Fractional number for SDM	R/W	0

FRA_H (0xF303) : initial value = 0x0000

Bit	Function	Description	R/W	Reset Value
15:4	--	Reserved	--	All “0”
3:0	FRA [19:16]	Fractional number for SDM	R/W	0

The relationship of Fractional PLL and Sigma delta DAC

DAC_ON

- Start Fractional PLL
  - The First, set 0xF301 / 0xF302 / 0xF303
    - If Sampling Rate is 48K/24K/12K/6K or 32K/16K/8K/4K
      - ✓ 0xF301 set as 0x08
      - ✓ 0xF302 set as 0x126E
      - ✓ 0xF303 set as 0x03
    - If Sampling Rate is 44.1K/22.05K/11.025K/5.5125K
      - ✓ 0xF301 set as 0x07
      - ✓ 0xF302 set as 0x6C22
      - ✓ 0xF303 set as 0x08
  - Second, Set 0xF300
    - Bit2 (DIV) : 0 (0: XIN/1, 1:XIN/2)
    - Bit1 (SDM_EN) : 1 (0: disable, 1:enable)
    - Bit0 (PD_PLL) : 0 (0: normal, 1: power down)
- Start DAC
  - SD_DAC_Turn_On (u16 DAC_SR)

DAC_OFF

- Stop DAC
  - SD_DAC_Turn_Off (void)
- Stop Fractional PLL
  - Set 0xF300
    - Bit2 (DIV) : 0 (0: XIN/1, 1:XIN/2)

- Bit1 (SDM_EN) : 0 (0: disable, 1:enable)
- Bit0 (PD_PLL) : 1 (0: normal, 1: power down)

Enter Power Down

- Stop DAC
- Stop Fractional PLL

Example: Please refer to SNP70032 Application Note

```
void main (void)
{
	// 1KHz Sine, 8KHz Sample Rate
	Sine_8k[0] = 0x0000; Sine_8k[1] = 0x40CD; Sine_8k[2] = 0x5BA6; Sine_8k[3] = 0x40CE;
	Sine_8k[4] = 0x0000; Sine_8k[5] = 0xBF34; Sine_8k[6] = 0xA45A; Sine_8k[7] = 0xBF33;

	SD_DAC_Turn_On(SR_8K, Mono);			// Set sampleing rate
	SD_DAC_Volume_Control(10);			// Set DAC Volume
	SD_DAC_Mute_OnOff(MUTE_OFF);			// Set DAC is not muting

	while(1)
	{
		_IObitSET(SFR_WDT, 14);			// Clr_WDTIRQ;
	}
}

void __interrupt [0x0034] DAC_INT(void)
{
	_IObitSET(SFR_INTCR,0);

	_setSR(SFR_DAOL, Sine_8k[0]);	//0
	_setSR(SFR_DAOL, Sine_8k[1]);	//1
	_setSR(SFR_DAOL, Sine_8k[2]);	//2
	_setSR(SFR_DAOL, Sine_8k[3]);	//3
	_setSR(SFR_DAOL, Sine_8k[4]);	//4
	_setSR(SFR_DAOL, Sine_8k[5]);	//5
	_setSR(SFR_DAOL, Sine_8k[6]);	//6
	_setSR(SFR_DAOL, Sine_8k[7]);	//7
}
```

3.14. Huffman Decoder

In SNP70032, We provide a hardware Huffman Decoder for such as MP3 or JPEG application.

HFD_CTRL (0xF200) : initial value = 0x0000

Bit	Function	Description	R/W	Reset Value
15:7	--	Reserved	--	All “0”
6	DEC_FINISH	input data decode finish, firmware can’t write it to 1 and must clear it to 0	R/W	0
5	DEC_TRG	Decoder Enable Trigger 1 -> 0 : Enable	R/W	0
4:0	TAB_SEL[4:0]	Table Select	R/W	0

HFD_INB_CNT (0xF201) : initial value = 0x0000

Bit	Function	Description	R/W	Reset Value
15:0	INB_CNT	Bits size of Huffman Data	R/W	0

HFD_OUTS_REQ_NUM (0xF202) : initial value = 0x0000

Bit	Function	Description	R/W	Reset Value
15:0	OUTS_REQ_NUM	Huffman Output size	R/W	0

HFD_USEB_CNT (0xF203) : initial value = 0x0000

Bit	Function	Description	R/W	Reset Value
15:0	USEB_CNT	When Huffman actually output size, use bits size this momnent	R	0

HFD_OUTS_ACT_NUM (0xF204) : initial value = 0x0000

Bit	Function	Description	R/W	Reset Value
15:0	OUTS_ACT_NUM	Huffman actually output size	R	0

HFD_FLAG (0xF205) : initial value = 0x0000

Bit	Function	Description	R/W	Reset Value
15:4	--	Reserved	--	0
3	DEC_HOLD	Implies Huffman Decoder Hold or not 0: Normal	R	0

		1: Hold		
2	STATED	For Debug used	R	0
1:0	STATE	For Debug used	R	0

3.15. SD Card / NAND Flash

3.15.1. DESCRIPTION

The flash memory (Mass-storage) interface provides an interface between Mass storage and 16bits dsp core. It supports 2 types of memory : NandFlash (Just support read function), SD Card (1/4 bits SD mode, SPI mode).

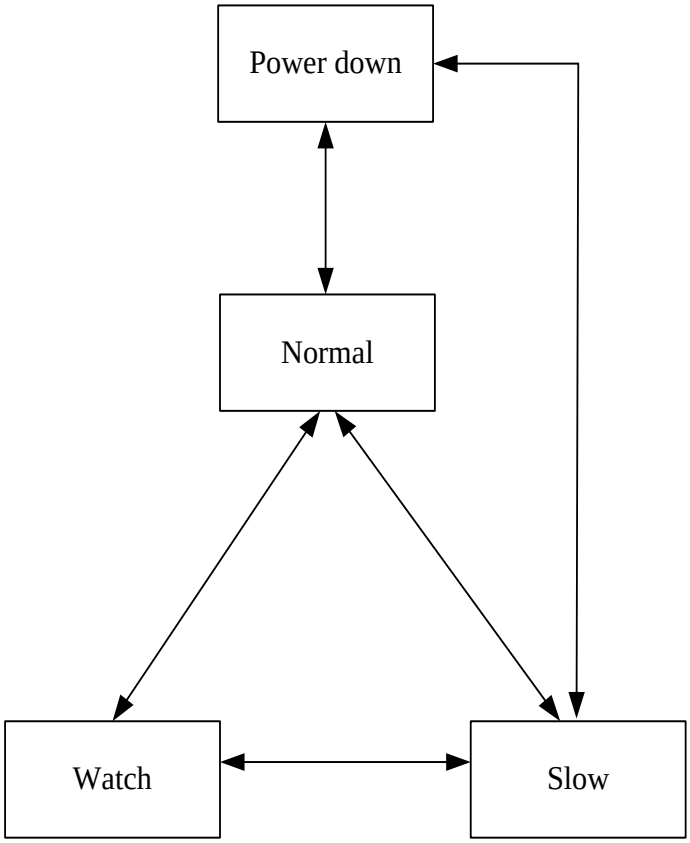
The detailed usage, please refer to SD Card / NAND flash library document

3.16. USB Device

The detailed usage, please refer to USB library document.

4. System Operating Mode

4.1. Operating Mode



The DSP each have the seven different operation modes (Normal mode / Slow mode / Watch mode / Power down mode)

	12M Xtal On	PLL	Logic/RAM/ROM	Freq (Hz)
Normal	V	V	V	48M / 24M / 12M / 6M
Slow	V	X	V	12M / 6M / 3M / 1.5M / 32K
Watch	X	X	X	32K
Power Down	X	X	X	

OPM_CTRL (0x000C) : initial value = 0x0003

Bit	Function	Description	R/W	Reset Value
15	GPIO_SELF_WKP_EN	0: N/A 1: GPIO Self Wakeup enable	R/W	0
14:13	CLK_DIV[1:0]	Set New System Clock 00: Sys_CLK/1, 01: Sys_CLK/2, 10: Sys_CLK/4 11: Sys_CLK/8	R/W	00
12	MCS_12M_DIV	When wakeup then switch to 0: 12MHz 1: 6M / 3M / 1.5MHz	R/W	0
11	MCS_32K	When wakeup then switch to 0: 48 / 12MHz 1: 32KHz	R/W	0
10	MCS_12M	When wakeup then switch to 0: 48MHz 1: 12MHz	R/W	0
9	--	Reserved	--	0
8	CLK12M_WKP_ON	1: Turn no 12MHz clock when wakeup	R/W	0
7	PLL_WKP_ON	1: Turn on PLL when wakeup	R/W	0
6	GPIO_WKP_EN	1: Wakeup source is GPIO	R/W	0
5	RTC_WKP_EN	1: Wakeup source is RTC	R/W	0
4	SELF32_WKP_EN	1: 48MHz switch to 32KHz self wakeup	R/W	0
3	SELF12_WKP_EN	1: 48MHz switch to 12MHz self wakeup	R/W	0
2	--	Reserved	--	0
1	WARM_H_EN	1: Warmup use high clock warmup counter	R/W	0
0	PW_ON_RST_EN	0: Power On next instruction 1: Power On reset system	R/W	0

OPM_CTRL1 (0x0025) : initial value = 0x0000

Bit	Function	Description	R/W	Reset Value
15	STP_SCLK	0: N/A 1: Stop System Clock	R/W	0
14	--	Reserved	--	0
13	STP_12M	0: N/A 1: Stop 12MHz Clock	R/W	0
12	STP_PLL	0: N/A	R/W	0

		1: Stop PLL		
11	PD_REG	0: Internal Regulator On 1: Internal Regulator Off	R/W	0
10:0	--	Reserved	--	All “0”

※ OPM register setting sequence :

1. OPM_CTRL
2. OPM_CTRL1

4.2. Wake up

4.2.1. Wake up DSP

Any High-to-Low or Low-to-High edge signal appear on P0.0~P0.13 or P1.0~P1.15 will force the chip to exit the stop mode and execute the first instruction (located at 0x000000).

P0WKUPEN (0x0030) : initial value = 0x0000

Bit	Function	Description	R/W	Reset Value
15:0	P0.[13:0]_WKUP	Pin wakeup enable 0: disable 1: enable	R/W	0

P1WKUPEN (0x0031) : initial value = 0x0000

Bit	Function	Description	R/W	Reset Value
15:0	P1.[15:0]_WKUP	Pin wakeup enable 0: disable 1: enable	R/W	0

Note:

The wake-up pin must be set to input mode.

Example: Please refer to SNP70032 Application Note

```
void main (void)
{
    unsigned int i;

    _setSR(SFR_P0En, 0xFFFF);
    _setSR(SFR_P0M, 0x0000);
```

```
    _setSR(SFR_P0PH, 0xFFFF);

    _IObitCLR(SFR_P0, 0);
    _IObitSET(SFR_WDT, 14);                          // Reset WDT

    _setSR(SFR_P0WKUPEN, 0xFFFE);                    // Set P0.0 as wakeup trigger source
    Switch_Main_Clk_Freq(HIGH_48M, SLOW_12M);        // System clock from 48MHz switch to slow
                                                     // 12MHz
    Power_Down(HIGH_48M);                            // Enter Power Down, then wakeup at
                                                     // 48MHz
    while(1)
    {
        _IObitSET(SFR_WDT, 14);
    }
}
```

5. Firmware data

DSP machine code

Operation	Mnemonic	BYTE 1	BYTE 2	Commands	Clock	Word
Call		**0**	**Absolute Address**	32K Word in a Program Bank	2	1
Jump		**1 0 0 0**	**Offset**	+/- 2K in one relative jump	2	1
Jump Condition		**1 0 0 1**	**Cnd***	**Offset**	+/- 128 in one relative jump	1/2	1
RW Mem (direct)	DM(imm)=Reg Reg=DM(imm)	**1 0 1 r #**	**Reg***	**Offset[7:0]**	512 Word in a Memory Bank (r=1) Reg <= M ; (r=0) Reg => M ; #, Offset[8]	1	1
Load Immediate	Dreg = immdeiate	**1 1 0 L**	**Reg1***	**Immediate**	L: 00 => Load High, Keep Low 10 => Keep High, Load Low 11 => Clear High, Load Low	1	1
AU(2) To Mem		**1 1 0 0 1**	**A** **Ix**	**0 Xop AU Yop**	Ix => 0 : Ix0, 1 : Ix1 A : 00 => No Change 01 => By Modifier 10 => +1 11 => -1		
LU(1)		**1 1 0 0 1**	**Reg**	**1 Xop LU1 0 YOP**		1	1
LU(2)		**1 1 0 0 1**	**Cst3 Cst2** **F**	**1 Cst1 Cst0 LU2 1 YOP**	F = 0 :r0, 1 : r1	1	1
RW SRAM (indirect)		**1 1 1 0 0**	**Reg**	**0 r A Ixy 0 0**	r = 1 : Reg <= RAM(offset) ; 0 : Reg => RAM(offset) ; A = 00 => No Change 01 => By Modifier 10 => +1 11 => -1 Ix => 0 : Ix0, 1 : Ix1 Ixy => 00 : Ix0, 01 : Ix1 10 : Iy0, 11 : Iy1	1	1
Load ROM (indirect)		**1 1 1 0 0**	**Reg**	**0 1 A Ixy 0 1**	Same as previous row	2	1
Shift index		**1 1 1 0 0**	**Reg**	**0 1 F SF 0 1 0**	F => 0 : r0, 1 : r1	1	1

	I/O(1)		1	1	1	0	0	r	RegL	1	Offset							128 words in IO space r = 1 : RegL <= IO(offset) ; 0 : Reg L=>IO(offset) ;	1	1
	AU(1)		1	1	1	0	1	Reg		Reg		AU		YOP					1	1
	MAC		1	1	1	1	0	MAC		M	Ix	A		DXY		X	Y	M : Multiple – function 0 : simple MAC 1 : multiple-function Ix => 0 : Ix0, 1 : Ix1 A : 00 => No Change 01 => By Modifier 10 => +1 11 => -1 DXY : 00 => X0 01 => X1 10 => Y0 11 => Y1 X => 0 : X0, 1 : X1 Y => 0 : Y0, 1 : Y1	1	1
	Reg Move		1	1	1	1	1	0	0	0	S Reg		D Reg			0	0	S => D	1	1
	Push/Pop		1	1	1	1	1	0	0	0	Reg		0	0	0	1	U	U: 0 : push/1: pop	1	1
	Shift		1	1	1	1	1	0	1	F	Reg		SF		SH			F => 0 : r0, 1 : r1	1	1
	IO(2) Push/Pop I/O		1	1	1	1	1	1	0	0	1	r	Offset					64 words in IO space r: 1 / Pop; Reg <= IO(offset) ; 0 / Push;Reg =>IO(offset) ;	1	1
			1	1	1	1	1	1	0	0	0	Reserved								
	Callff		1	1	1	1	1	1	0	1	Abs_Addr[23:16]							Call Far Far(24 bit address) Push 2 words to Queue	3	2
			Abs_Addr[15:0]																	
	Jumpff		1	1	1	1	1	1	1	0	Abs_Addr[23:16]							Jump Far Far(24 bit address)	2	2
			Abs_Addr[15:0]																	
	Do0		1	1	1	1	1	1	0	0	0	0	CntV					Count Value: 64 times	1	1
	Do1		1	1	1	1	1	1	0	0	0	1	CntV					Count Value: 64 times	1	1
	Loop0		1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	Loop0	1	1
	Loop1		1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	Loop1	1	1
	ret		1	1	1	1	1	1	1	1	0	1	0	0	0	0	0	0	ret	2	1
	reti		1	1	1	1	1	1	1	1	0	1	0	0	0	0	0	1	reti	2	1
	Retff		1	1	1	1	1	1	1	1	0	1	0	0	0	0	1	0	Return Far Far Pop 2 words from Queue	3	1

	ICEC		**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**0**	**1**	ICE Call Function	3	1
	NOP		**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**		1	1
	DisSPSW		**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	Clear SCR.SPSW	1	1
	EnSPSW		**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	Enable SCR.SPSW write	1	1
	EMAC		1	1	1	1	0	MAC	**EM**	**0**	**AmX**	**AmY**	**DmX**	**DmY**	X	Y	EM : Multiple – function 0 : simple MAC 1 : multiple-function AmX : 0 => ImxL (Linear) 1 => ImxC (Circular) AmY : 0 => ImyL (Linear) 1 => ImyC (Circular) DmX : 0 => X0 1 => X1 DmX : 0 => Y0 1 => Y1 X => 0 : X0, 1 : X1 Y => 0 : Y0, 1 : Y1	1	1

AU

0	0	0	X + 1	
0	0	1	X – 1	
0	1	0	X + Y	
0	1	1	X + Y + C	
1	0	0	X – Y	
1	0	1	X – Y + C – 1	
1	1	0	–X + Y	
1	1	1	–X + Y + C – 1	

Cnd

0	0	0	0	Equal	ZF
0	0	0	1	Not Equal	~ZF
0	0	1	0	Greater Than	~(SF^AV)&~ZF
0	0	1	1	Greater Than or Equal	~(SF^AV) | ZF
0	1	0	0	Less Than	(SF^AV)&~ZF
0	1	0	1	Less Than or Equal	(SF^AV) | ZF
0	1	1	0	ALU Overflow	AV
0	1	1	1	Not ALU Overflow	~AV
1	0	0	0	ALU Carry	CF
1	0	0	1	Not ALU Carry	~CF
1	0	1	0	MR0 Negative	MR0[15]

1	0	1	1	Not MR0 Negative	~MR015]
1	1	0	0	MAC Overflow	MV
1	1	0	1	Not MAC Overflow	~MV
1	1	1	0	Index Overflow	IV
1	1	1	1	Special EIR Case	IntRR[7]

{Cst3,Cst2,Cst1,Cst0}

0	0	0	0	0000000000000001
0	0	0	1	0000000000000010
0	0	1	0	0000000000000100
0	0	1	1	0000000000001000
0	1	0	0	0000000000010000
0	1	0	1	0000000000100000
0	1	1	0	0000000001000000
0	1	1	1	0000000010000000
1	0	0	0	0000000100000000
1	0	0	1	0000001000000000
1	0	1	0	0000010000000000
1	0	1	1	0000100000000000
1	1	0	0	0001000000000000
1	1	0	1	0010000000000000
1	1	1	0	0100000000000000
1	1	1	1	1000000000000000

LU1

0	0	X .AND. Y	
0	1	X .OR. Y	
1	0	X .XOR. Y	
1	1	.NOT. X	

LU2

0	0	~Const .AND. Y	CLEAR
0	1	Const .OR. Y	SET
1	0	Const .XOR. Y	TOG
1	1	Const .AND. Y	TEST

MAC

0	0	0	X * Y (IS)	Integer Signed
0	0	1	MR + X * Y (IS)	Integer Signed

0	1	0	MR - X * Y (IS)	Integer Signed
0	1	1	TBD	
1	0	0	X * Y (FS)	Floating Signed
1	0	1	MR + X * Y (FS)	Floating Signed
1	1	0	MR – X * Y (FS)	Floating Signed
1	1	1	TBD	

Reg

Reg			Mem		Command
0	0	0	X0	L	
0	0	1	X1		
0	1	0	R0		
0	1	1	R1		
1	0	0	Y0	H	
1	0	1	Y1		
1	1	0	MR0		
1	1	1	MR1		

Reg1

Reg			Mem		Command
0	0	0	X0	L	
0	0	1	X1		
0	1	0	R0		
0	1	1	R1		
1	0	0	Y0	H	
1	0	1	Y1		
1	1	0	Ix0		
1	1	1	Ix1		

RegL

Reg		Mem		Command
0	0	X0		
0	1	X1		
1	0	R0		
1	1	R1		

SF

0	0	Shift Left Sign Extension
0	1	A/L Shift Left
1	0	A Shift Right
1	1	L Shift Right

SH

0	0	0	1 bit
0	0	1	2 bit
0	1	0	3 bit
0	1	1	4 bit
1	0	0	5 bit
1	0	1	6 bit
1	1	0	7 bit
1	1	1	8 bit

Xop

0	0	X0	
0	1	X1	
1	0	R0	
1	1	R1	

Yop

0	0	Y0	
0	1	Y1	
1	0	R0	
1	1	R1	

DISCLAIMER

The information appearing in SONiX web pages (“this publication”) is believed to be accurate.
However, this publication could contain technical inaccuracies or typographical errors.
The reader should not assume that this publication is error-free or that it will be suitable for any particular purpose.
SONiX makes no warranty, express, statutory implied or by description in this publication or other documents which are referenced by or linked to this publication. In no event shall SONiX be liable for any special, incidental, indirect or consequential damages of any kind, or any damages whatsoever, including, without limitation, those resulting from loss of use, data or profits, whether or not advised of the possibility of damage, and on any theory of liability, arising out of or in connection with the use or performance of this publication or other documents which are referenced by or linked to this publication.
This publication was developed for products offered in Taiwan. SONiX may not offer the products discussed in this document in other countries. Information is subject to change without notice. Please contact SONiX or its local representative for information on offerings available. Integrated circuits sold by SONiX are covered by the warranty and patent indemnification provisions stipulated in the terms of sale only.
The application circuits illustrated in this document are for reference purposes only. SONIX DISCLAIMS ALL WARRANTIES, INCLUDING THE WARRANTY OF MERCHANTABILITY OR FITNESS FOR ANY PURPOSE.
SONIX reserves the right to halt production or alter the specifications and prices, and discontinue marketing the Products listed at any time without notice. Accordingly, the reader is cautioned to verify that the data sheets and other information in this publication are current before placing orders.
Products described herein are intended for use in normal commercial applications.
Applications involving unusual environmental or reliability requirements, e.g. military equipment or medical life support equipment, are specifically not recommended without additional processing by SONIX for such application.