SUNPLUS

SPMC70 Series Reference Manual

V1.0 – Oct. 07, 2003

19, Innovation First Road • Science-Based Industrial Park • Hsin-Chu • Taiwan • R.O.C.
886-3-578-6005 886-3-578-4418 www.sunplus.com.tw

Important Notice

1 Table of Content

PAGE

2 Revision History

Revision	Date	By	Remark
1.0	10/07/2003	Lanin Lin	First edition

3 Introduction

3.1 Introduction

The SPMC 70 series is the embedded 16-bit micro-controller, designed for home application, car control system, and network servo, … etc. This document explains the MCU family architecture and peripheral modules, but not including specification of each device. This menu intends to provide the general details and operation of the SPMC architecture and peripheral modules.

3.2 Device Structure

Each part of a device can be defined into three groups:

1. ISA V1.2 or above CPU Core
2. Peripherals Function
3. Special Features

3.2.1 The CPU Core

The core pertains to the basic features that are the requirements for making the device to operate. These include:

1. Device Oscillator
2. Reset logic
3. CPU (Central Processing Unit) operation
4. ALU (Arithmetic Logical Unit) operation
5. Device memory map organization
6. Interrupt operation
7. Instruction set

3.2.2 Peripherals Function

Peripherals functions are the features that add a differentiation from a microprocessor. These functions facilitate interfacing to the external world (such as general purpose I/O, LCD drivers, A/D inputs, and PWM outputs), and internal tasks such as keeping different time bases (such as timers). The peripherals that are discussed are:

1. General purpose I/O
2. Timer0,Timer1,Timer2
3. Time Base
4. Capture, Compare, and PWM (CCP)
5. Standard Peripheral Interface (SPI)
6. Serial Input & Output (SIO)

7. UART Revision
8. Low Voltage Detection (LVD)
9. 10-bit Analog to Digital (A/D)
10. Liquid Crystal Display (LCD) Drivers (under design)
11. Parallel Communication Interface (PCI)

3.2.3 Special Features

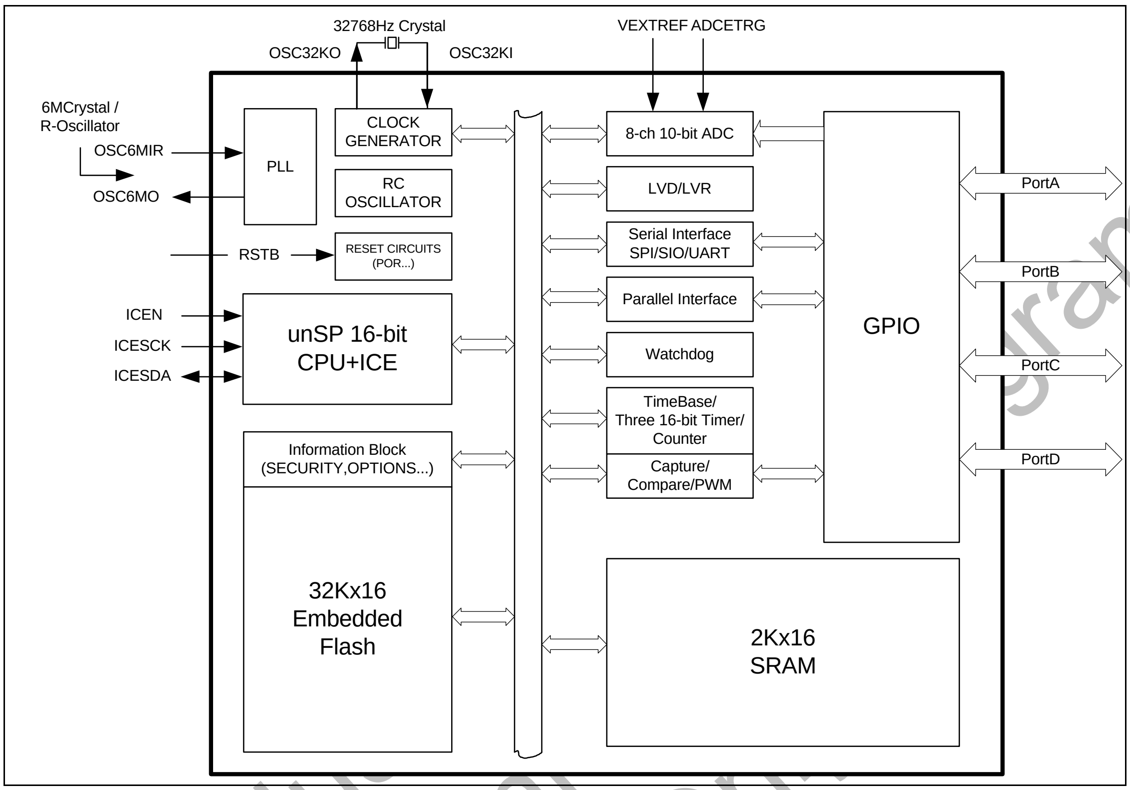
Special features are the unique features to help performing one or more of the following tasks:

1. Lower system cost
2. Increase system reliability
3. Increase design flexibility

The SPMC 70xxx MCUs offer several features to help achieving these goals.

1. On-chip Power-on Reset (POR)
2. Low Voltage Reset (LVR)
3. Watchdog Timer
4. Low power mode (Halt)(Sleep)(Standby)
5. RC device oscillator
6. In-Circuit Serial Programming

3.3 Function Block



3.4 Device Varieties

Once the functional requirements of the device are specified, some other decisions need to be made. These include:

- Operating voltage: 2.7V ~ 3.6V
- Operating temperature range: 0℃ to +70℃
- Operating frequency: 32768Hz, 750KHz, 1.5MHz, 3MHz, 6MHz, 12MHz, 24MHz, and 48MHz
- Memory Varieties
- Packaging

3.5 Device Naming Rule

The naming rule of SPMC 70xxx shows as follows:

SPMC 70ABCDE

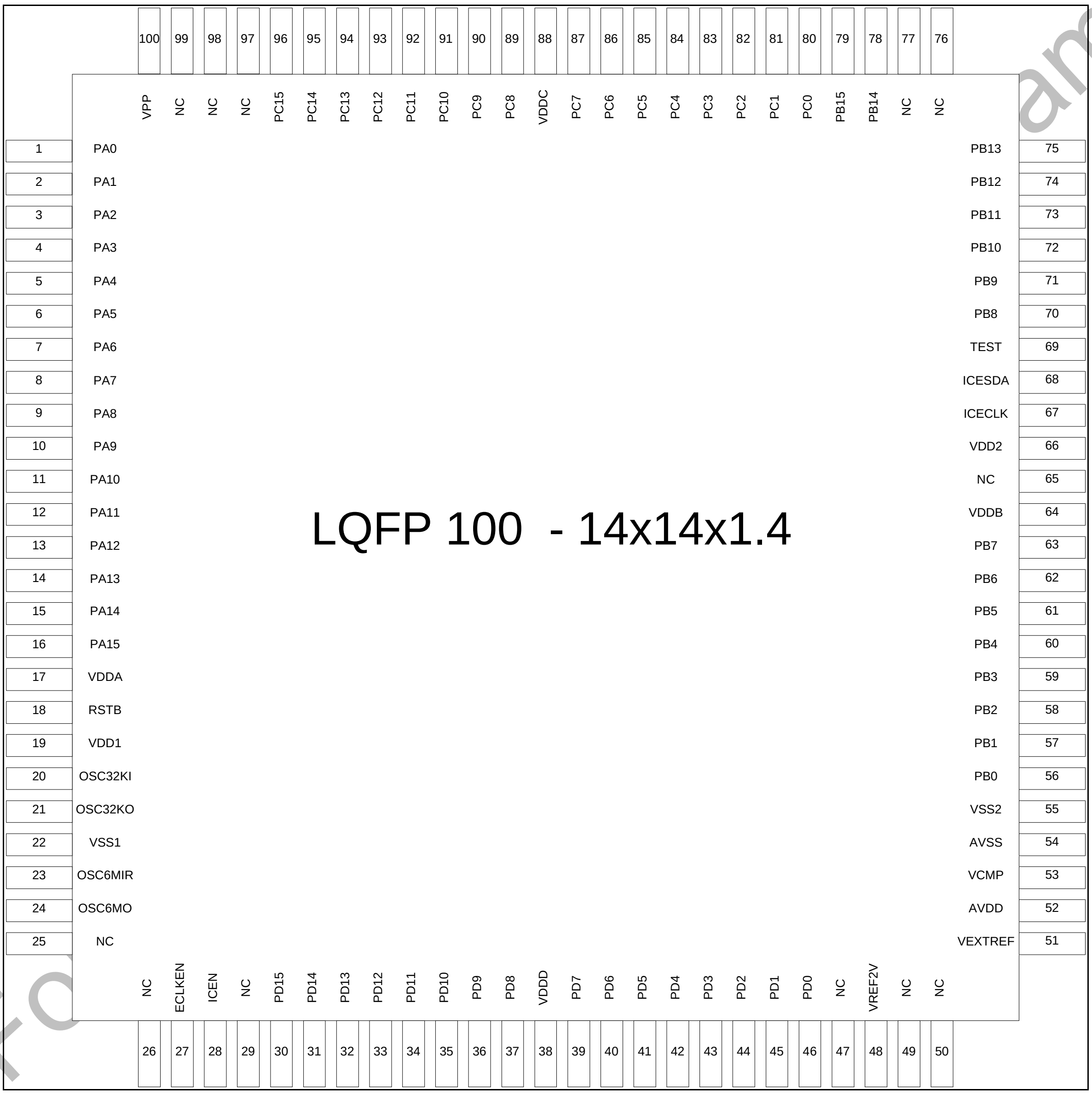
1. A: Function Type	→	0: General	1: A/D	2: LCD		4: Inverter
2. B: ROM Type	→	F: Flash type	P: OTP type	C: Mask ROM type		Z: ROM Less type
2. C: ROM Size	→	0: ROM Less	1: 1K word	2: 2K words	4: 4K words	
		8:8K words	C:12K words	H:16K words	K:24K words	
		M:32K words	P:48K words	S:64K words	W:128K words	

3. D: Sequence Version → 0, 1, 2, 3, ….. 9

4. E: IC version.

3.6 Packaging Rule

A: LQFP 100 – Pitch 0.5 mm → Package Number PL08



3.7 Pin Descriptions

Pin No.	Pin Name	Type	Description
1	PA0/PCID0	I/O	PortA[0] or Parallel Communication Interface data bit 0
2	PA1/PCID1	I/O	PortA[1] or Parallel Communication Interface data bit 1
3	PA2/PCID2	I/O	PortA[2] or Parallel Communication Interface data bit 2
4	PA3/PCID3	I/O	PortA[3] or Parallel Communication Interface data bit 3
5	PA4/PCID4	I/O	PortA[4] or Parallel Communication Interface data bit 4
6	PA5/PCID5	I/O	PortA[5] or Parallel Communication Interface data bit 5
7	PA6/PCID6	I/O	PortA[6] or Parallel Communication Interface data bit 6
8	PA7/PCID7	I/O	PortA[7] or Parallel Communication Interface data bit 7
9	PA8/PCSB	I/O	PortA[8] or Parallel Communication Interface CSB
10	PA9/PCIA0	I/O	PortA[9] or Parallel Communication Interface A0
11	PA10/PWEB/OUTRST	I/O	PortA[10] or Parallel Communication Interface WEB or system reset output
12	PA11/PRDB	I/O	PortA[11] or Parallel Communication Interface RDB
13	PA12/RXD	I/O	PortA[12] or UART receive data input
14	PA13/TXD	I/O	PortA[13] or UART transmission data output
15	PA14/SDA	I/O	PortA[14] or SIO serial address/data
16	PA15/SCL/$\overline{SS}$	I/O	PortA[15], SIO serial clock or SPI $\overline{SS}$ input
17	VDDA	I	3V/5V power for IOA[15:0]
18	RSTB	I(PH)	External reset
19	VDD1		Digital power 1
20	OSC32KI	I	32768Hz crystal input
21	OSC32KO	O	32768Hz crystal output
22	VSS1		Digital ground
23	OSC6MIR	I	External 6MHz crystal input or R-oscillator resistor input Code option for crystal or R-oscillator, the code-option is stored in embedded flash information block.
24	OSC6MO	O	External 6MHz crystal output
25	NC		
26	NC		
27	ECLKEN	I(PL)	External clock enable,
28	ICEN	I(PL)	ICE enable 1= ICE enable 0= Normal mode (ICE disable)
29	NC		
30	PD15	I/O	PortD[15]
31	PD14	I/O	PortD[14]
32	PD13	I/O	PortD[13]
33	PD12	I/O	PortD[12]
34	PD11	I/O	PortD[11]
35	PD10	I/O	PortD[10]
36	PD9	I/O	PortD[9]
37	PD8	I/O	PortD[8]
38	VDDD	I	3V/5V power for IOD[15:0]
39	PD7	I/O	PortD[7]
40	PD6	I/O	PortD[6]
41	PD5	I/O	PortD[5]
42	PD4	I/O	PortD[4]
43	PD3	I/O	PortD[3]

Pin No.	Pin Name	Type	Description
44	PD2	I/O	PortD[2]
45	PD1	I/O	PortD[1]
46	PD0	I/O	PortD[0]
47	NC		
48	VREF2V	O	2V reference output voltage
49	NC		
50	NC		
51	VEXTREF	I	ADC top voltage reference
52	AVDD		Analog power
53	VCMP	I	ADC 1/2 AVDD voltage reference
54	AVSS		Analog ground
55	VSS2		Digital ground
56	PB0/AD0	I/O	PortB[0] or ADC channel 0 analog input
57	PB1/AD1	I/O	PortB[1] or ADC channel 1 analog input
58	PB2/AD2	I/O	PortB[2] or ADC channel 2 analog input
59	PB3/AD3	I/O	PortB[3] or ADC channel 3 analog input
60	PB4/AD4	I/O	PortB[4] or ADC channel 4 analog input
61	PB5/AD5	I/O	PortB[5] or ADC channel 5 analog input
62	PB6/AD6	I/O	PortB[6] or ADC channel 6 analog input
63	PB7/AD7	I/O	PortB[7] or ADC channel 7 analog input
64	VDDB	I	3V/5V power for IOB[7:0]
65	VDDB	I	3V/5V power for IOB[15:8]
66	VDD2	I	Digital power 2
67	ICECLK	I	ICE serial clock input
68	ICESDA	I/O	ICE serial address/data input/output
69	TEST	I(PL)	Test mode control 0= Normal mode 1= Test mode
70	PB8/ADTRIG	I/O	PortB[8] or ADC external trigger
71	PB9/EXT1	I/O	PortB[9] or External input 1
72	PB10/EXT2	I/O	PortB[10] or External input 2
73	PB11/CCP0	I/O	PortB[11] or Timer0 Capture/Compare/PWM input/output
74	PB12/CCP1	I/O	PortB[12] or Timer1 Capture/Compare/PWM input/output
75	PB13/SDO	I/O	PortB[13] or SPI data output
76	NC		
77	NC		
78	PB14/SDI	I/O	PortB[14] or SPI serial data input
79	PB15/SCK	I/O	PortB[15] or SPI serial clock
80	PC0	I/O	PortC[0]
81	PC1	I/O	PortC[1]
82	PC2	I/O	PortC[2]
83	PC3	I/O	PortC[3]
84	PC4	I/O	PortC[4]
85	PC5	I/O	PortC[5]
86	PC6	I/O	PortC[6]
87	PC7	I/O	PortC[7]
88	VDDC	I	3V/5V power for IOC[15:0]
89	PC8	I/O	PortC[8]
90	PC9	I/O	PortC[9]
91	PC10	I/O	PortC[10]
92	PC11	I/O	PortC[11]
93	PC12	I/O	PortC[12]

Pin No.	Pin Name	Type	Description
94	PC13	I/O	PortC[13]
95	PC14	I/O	PortC[14]
96	PC15	I/O	PortC[15]
97	NC		
98	NC		
99	NC		
100	VPP	I	High voltage pin used in test mode and operating in 0 ~ 15V. A high voltage pad with I/O path for VPP pin and an input path for input pin will be provided by TSMC. So this high voltage pad can be used as a normal pad in user mode. But this input path has a Vt drop when pass logic high (VDD) signal.

3.8 Development Support

Sunplus offers a wide range of development tools that allow users to efficiently develop and debug application code. All tools developed by Sunplus operate under the un'SP® Integrated Development Environment (IDE). This IDE supports all the follow functions:

1. Code generation
2. Software debug
3. Device programmer
4. Product evaluation boards

While using IDE to simulate SPMC 70xxx series, the IDE must be set to SPMC701FM0. This body includes ICE and most functions of SPMC70xxx series.

4 Oscillator

4.1 Oscillator

There are two oscillator systems in the SPMC70xxx series, providing two clocks for operation: 6MHz and 32.768KHz. The 6MHz input can be selected as R-oscillator or crystal, and the 32.768KHz only can be set as crystal. The 6MHz input will be as PLL (Phase-Lock-Loop) clock source, and the PLL circuit pumps the 6MHz input clock to 96MHz. The SPMC70xxx also provides the RTC circuit for real-time counter. To support real-time counter, the 32.768KHz crystal is needed for the precise clock source.

In order to prevent the clock glitch, a clock filter is added to limit the pulse width of system clock. The SPMC 70xxx series also has an internal clock source for flash circuit. The system will generate a 1600KHz by internal R-oscillator and produce 200KHz clock by internal divider.

The necessary oscillator circuits are listed as follows:

- 1600KHz R-oscillator (generating 200KHZ clock by divider)
- Phase-Lock Loop (pumps the 6MHz crystal or 6MHz R-oscillator to 96MHz)
- 6MHz crystal pads
- 32.768KHz crystal pads
- Clock filter

The CPU operation clock rate can be programmed as 48MHz, 24MHz, 12MHz, 6MHz, 3MHz, 1.5MHz, 750KHz, and 32.768KHz by clock control register.

4.1.1 200KHz R-oscillator

The 1600KHz R-oscillator provides a 200KHz clock for flashing controller to generate the necessary control signals meeting the timing specifications of flash erasing and programming. It can be disabled in either sleep mode or standby mode for power saving. In addition, 200 KHz clock is also used for power-on-reset Timer (PWRT) and wakeup timer in power saving. The 85Kohm resistor is built-in. The 1600KHz clock source can be chosen from external in test mode.

4.1.2 96MHz oscillator and Phase-Lock Loop (PLL)

There is a code option bit which selects whether the PLL reference clock coming from 6MHz R-oscillator or crystal output. The code option is stored in the information block of embedded flash. In the reset stage, the option word is read and loaded into option registers. In case of test mode, the 96MHz clock source is from PLL output regardless of code option.

4.2 Function Block

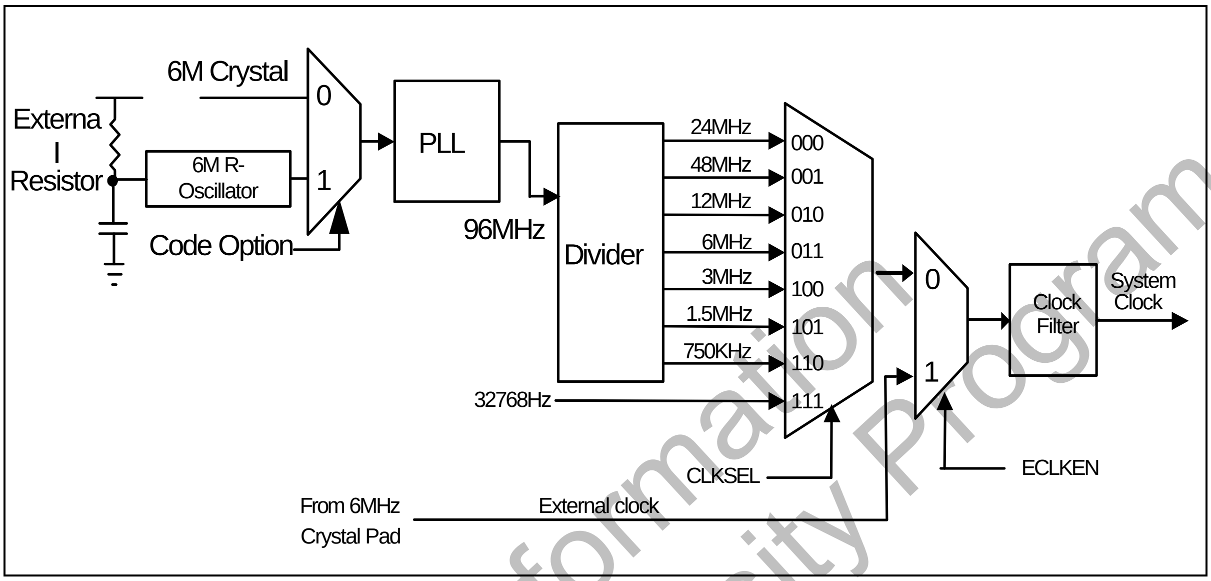


Figure 2-1 Function Block

Note:

1. Code Option is set at P_System_Option bit 0.
2. ECLKEN is set from external IO pin.
3. CLKSEL is set at P_Clk_Ctrl bit 2:0.

4.3 Control Register

4.3.1 P_Clk_Ctrl ($7069): Clock control register

This register is used for setting CPU clock.

b15	b14	b13	b12	b11	b10	b9	b8
-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-

b7	b6	b5	b4	b3	b2	b1	b0
-	-	-	-	-	R/W	R/W	R/W
-	-	-	-	-	0	0	0
-	-	-	-	-	CLKSEL		

Bit 15:3 Reserved

Bit 2:0 CLKSEL: CPU clock selection

000: 24MHz (default)

001: 48MHz

010: 12Mhz

011: 6Mhz

100: 3MHz

101: 1.5MHz

110: 750KHz

111: 32768hz

Note:

1. The procedure of programming is performed with two consecutive write cycles. First, write 0x5a5a into this register. Next, write the setting value into this register within 16 CPU clock cycles.
2. When power drops to or below 3.0V, do not set the system clock to 48 MHz to prevent the unstable state.

4.3.2 P_System_Option ($8000): System Option Register

This address places at information block, not in the general flash address. Users must use ICE function or writer to set this port. For detailed description, please checks section 6.

B15	b14	b13	b12	b11	b10	b9	b8
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	1	0	1	0	1	0	1
Verification Pattern							

b7	b6	b5	b4	b3	b2	b1	b0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	1	0	1	0	1	1	1
Verification Pattern			Security	LVD	LVR	WDG	CLK Source

Bit 15:5 Verification Pattern

ICE or Writer can read/write this area in the information block

Bit 4 Security: Security selection bit

0: protected, the main block in the flash can not be accessed

1: not protect, can be readable or write-able

Bit 3 LVD: enable low voltage detection function

0: enable

1: disable

Bit2 LVR: enable low voltage reset function

0: disable

1: enable

Bit1 WDG: enable watchdog function

0: disable

1: enable

Bit0 CLK Source: Clock Source Selection

0: R oscillator

1: crystal oscillator

4.4 Application Circuit

The application circuit of oscillator in SPMC70xxx series can be separated as 32KHz and 6MHz oscillators. The 32KHz circuit shows in figure 2-2. Also, the 6MHz circuit can be configured as three types: one is configured by connecting with crystal as shown in Figure 2-3; another is configured by connecting with resistance as shown in Figure 2-4; and the other one can be configured as shown in Figure 2-5.

The external clock should work with "ECLKEN" pin. When the ECLKEN pin is pulled high, system will check the external clock source.

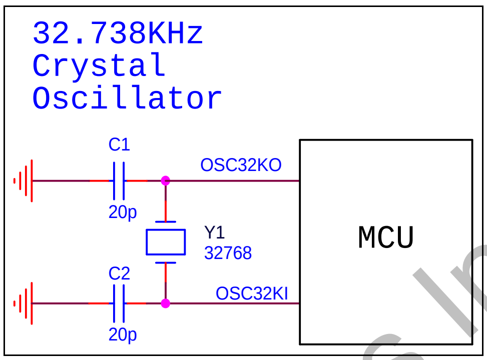


Figure 2-2

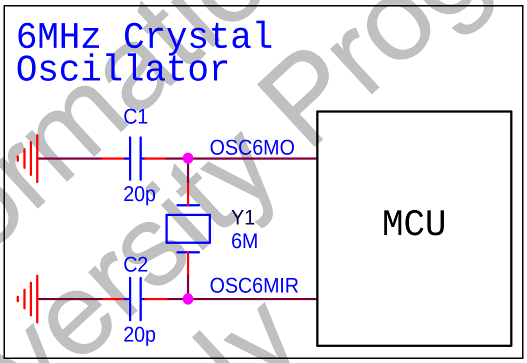


Figure 2-3

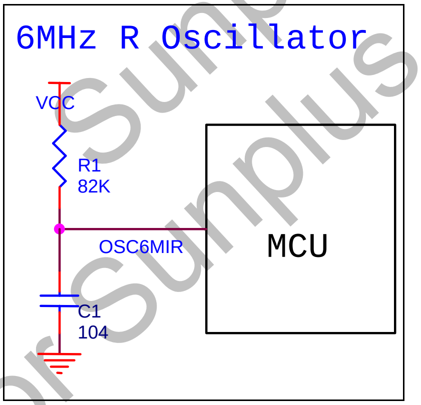


Figure 2-4

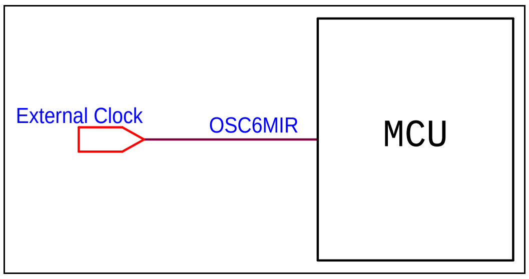


Figure 2-5

4.5 Design Tips

Basically, the system clock is provided by oscillator, or though the PLL and divider to generate the eight CPU clock selections. Programming P_Clk_Ctrl (R/W)($7069H) determines the frequency of CPU clock for system. The default system clock is 24MHz.

【Example-1】: This example shows the procedure of how to set system clock. Users can check P_Clk_Ctrl ($7069) table and change the system clock.

A) Use assemble language

```
R1 = 0x5a5a
[P_Clk_Ctrl] = R1
R1 = C_CLKSEL_12M
[P_Clk_Ctrl] = R1
```

B) Use C language

```
Set_Clk_Ctrl(C_CLKSEL_12M);
```

C) Use C Pointer

```
*P_Clk_Ctrl = 0x5a5a;
*P_Clk_Ctrl = C_CLKSEL_12M;
```

Note:

To enter the consecutive writing cycle, users have to write $5a5a to the registers listed above. Next, write data to the same registers to complete the writing cycle. The procedure of programming is performed with two consecutive writing cycles. First of all, write the confirming command "0x5a5a", and the second write cycle programs essential registers successfully. If the duration of the first write and second write is more than 16 CPU clock cycles, the program sequence fails, but this limit does not exist in ICE mode.

5 Reset

5.1 Introduction

In SPMC70xxx series, the reset logic is used for leading MCU into a known state. The source of reset can be determined by using the device status bits. The reset circuit can be used for increasing system reliability.

The various types of resets:

1. Power on reset (POR)
2. External reset
3. Low voltage reset (LVR)
4. Watchdog timer reset (WDTR)
5. Illegal address reset（IAR）
6. ICE software reset
7. Parallel Port software reset

A simplified block diagram of the on-chip reset circuit is shown in the Figure 3-1.

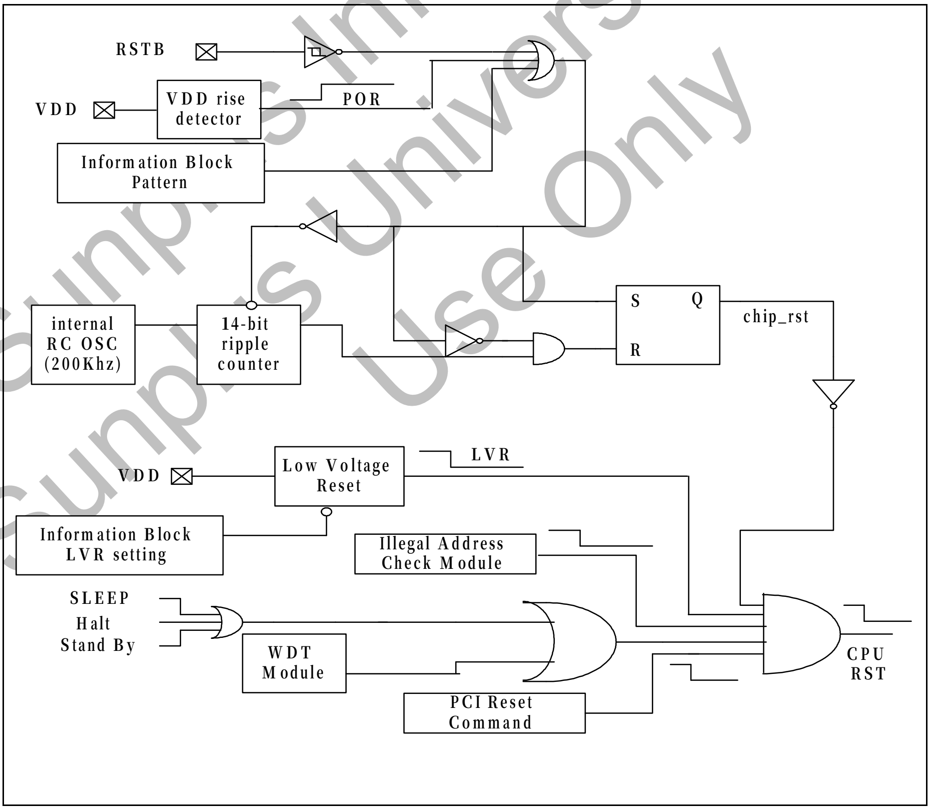


Figure 3-1 Reset Block

5.2 Reset Mode

5.2.1 Power on reset (POR)

A power-on-reset (POR) is generated when VDD rising is detected. When the rising rate is greater than 0.5 V per us and VDD raised to acceptable level, the power on reset circuit starts working. In SPMC70xxx series, the power-up timer will last for 82 ms (T_{PWRT} , 200KHz, Count 2^14) when power on reset. After a reset time, all registers will be initiated at designate value. See Figure 3-2 for timing details.

Note: The power up time delay is not the same value from device to device due to temperature, VDD, or wafer process variation.

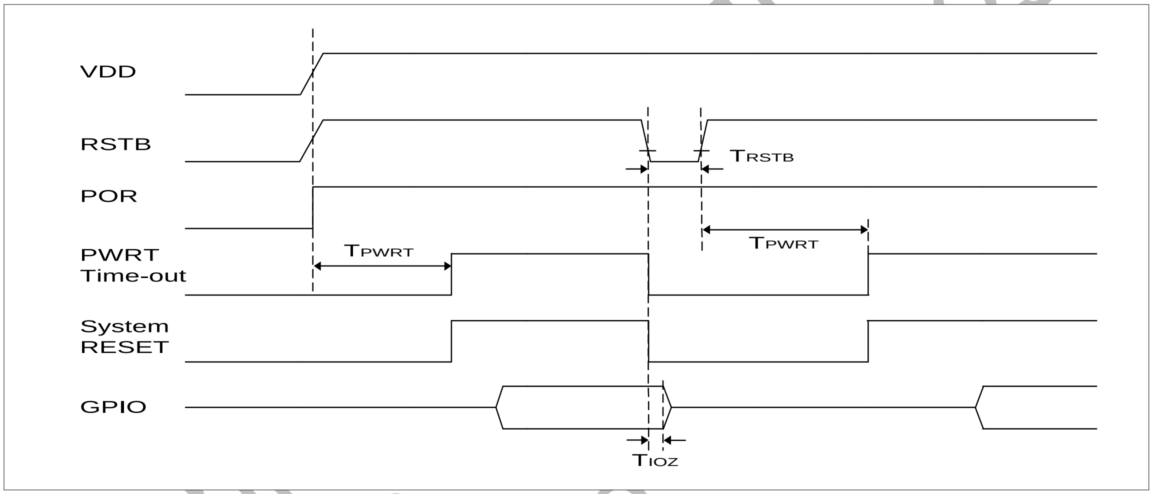


Figure 3-2 External reset, Power-On Reset, Power-Up Timer Timing

5.2.2 External Reset

The SPMC70xxx provides an external pin to force the system returning to the initial status. The RSTB is used to connect an RC circuit, shown in Figure 3-3. This pin is a low active signal. When the RSTB pin falls below 0.3 * VDD, system will be forced entering into reset state. If the recharging voltage on capacitor is greater than acceptable voltage, system reset will last for 82ms (T_{PWRT}) when power on reset and then complete the whole reset function. See Figure 3-2 for timing information.

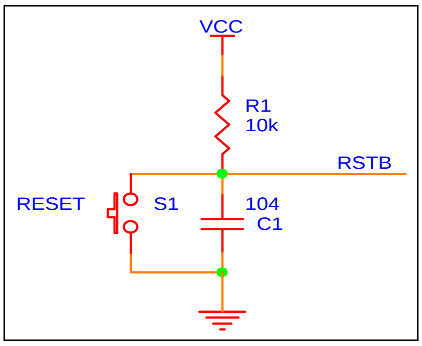


Figure 3-3 External Reset Circuit

5.2.3 Low Voltage Reset (LVR)

The on-chip Low Voltage Reset (LVR) circuitry makes the device entering reset state when the MCU voltage falls below a certain voltage. This function ensures that the MCU does not continue to work at the invalid operating voltage range. To turn on LVR function, user must set the b2 of P_System_Option ($8000). Also, the firmware provides the LVRENB bit of P_LVDLVR_Ctrl($7066) register to turn on/off LVR function. The LVRENB bit is optional to enable/disenable the LVR function. If VDD falls below 2.6V and at least for a period of Tw, the LVR will reset the chip if the LVR is enabled. The chip will remain in the reset condition for the period of Textend after VDD returns above 2.6V. The Figure 3-4 shows the typical low voltage reset timing. For operation method, please see the Chapter 9.

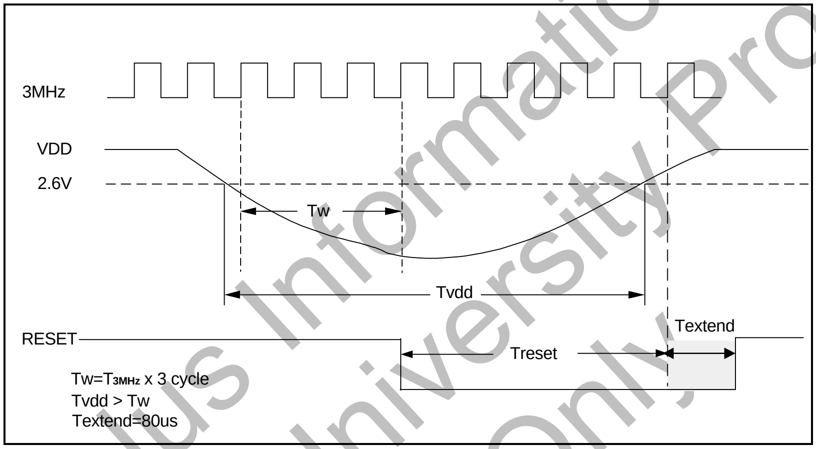


Figure 3-4 Low voltage reset timing

5.2.4 Watch Dog Timer Reset (WDTR)

On-chip watchdog circuitry makes the device entering into reset when the MCU goes into unknown state and without any watchdog clearance. This function ensures the MCU does not continue to work in abnormal condition. The WDTR can be configured by the b1 of **P_System_Option ($8000)** and the b0 of P_WatchDog_Ctrl($7063). We can set the overflow timer by using b[3:1] of P_WatchDog_Ctrl. Also, when “0x5a5aH” is written into P_WatchDog_Clr(W) ($7064), the watchdog timer will be reset and continue to count. If P_WatchDog_Clr is not written between watchdog counting interval, the system will be forced to reset CPU. The watchdog reset is disabled in ICE mode.

5.2.5 Illegal address reset

The SPMC70xxx series offers an illegal address reset for preventing system entering into illegal address. When system goes into illegal address, only CPU will be reset.

5.2.6 ICE software reset

Software reset command is used in ICE mode.

5.2.7 Parallel Port software reset

In PCI transmission, a slave equips a parallel reset function to reset flash and peripheral modules by a master.

5.2.8 Reset Source table

These reset sources can reset various modules in SPMC70xxx, see the following:

Sources	ICE	CPU	Other modules
Power on Reset	Reset	Reset	Reset with PWRT
External Reset	Reset	Reset	Reset with PWRT
Low Voltage Reset		Reset	Reset
Parallel Reset			Reset
ICE soft Reset		Reset	Reset enable in ice mode
Watchdog Reset		Reset except ice mode	
Illegal address reset		Reset	

Note:

Flash controller and other modules are reset by power on reset and external reset, which the reset signal retains until power on timer is counted over. When the pattern in information block is checked failed, the initial reset will be valid until either the power on reset or the external reset occurs. In ICE mode, the flash controller will not be reset by Initial reset. Therefore, users can rewrite the information block in ICE mode.

5.3 Control Register

5.3.1 P_WatchDog_Ctrl (R/W) ($7063)

This register provides the watchdog clear timer and on/off function for firmware setting.

b15	b14	b13	b12	B11	b10	b9	b8
-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-

b7	b6	b5	b4	b3	b2	b1	b0
-	-	-	-	R/W	R/W	R/W	R/W
-	-	-	-	0	0	0	0
-	-	-	-	WDTTMR			WDTENB

Bit15:4	Reserved
Bit3:1	WDTTMR
	111: 2683ms clear
	110: 1341ms clear

101: 670ms clear

100: 335ms clear

011: 168ms clear

010: 83840us clear

001: 41920us clear

000: 20960us clear

Bit 0 WDTENB

0: enable watchdog timer reset depends on WDG (bit 1) of information block

1: disable watchdog timer reset

Note:

The Watchdog control port, P_WatchDog_Ctrl (R/W) ($7063), should be configured by two consecutive writing cycles to avoid false writing. First of all, write 0x5a5a into this register. Then, write the setting value into this register within 16 CPU clock cycles.

5.3.2 P_WatchDog_Clr (W) ($7064): The watchdog clearance port

b15	b14	b13	b12	b11	b10	b9	b8
W	W	W	W	W	W	W	W
0	1	0	1	1	0	1	0
WDTCLR							

b7	b6	b5	b4	b3	b2	b1	b0
W	W	W	W	W	W	W	W
0	1	0	1	1	0	1	0
WDTCLR							

P_WatchDog_Clr(W)($7064) is used to clear watchdog timer, which is activated by write “0x5a5aH” to this port.

5.3.3 P_Reset_Status(R/W) ($7065): This register shows the flag of reset status for firmware checking.

b15	b14	b13	b12	b11	b10	b9	b8
-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-

b7	b6	b5	b4	b3	b2	b1	b0
-	-	R/W	R/W	R/W	R/W	R/W	R/W
-	-	0	0	0	0	0	0
-	-	PCIRST	ICERST	ILLADDR	WDTRST	LVRST	EXTRST

Bit 15:6 Reserved

Bit 5 PCIRST: Reset from PCI

1= The previous reset event is due to parallel reset command

0= No parallel reset occurs

This flag is cleared by writing this bit by “1”

Bit 4 ICERST: ICE software reset flag

1= The previous reset event is due to ICE reset command

0= No ICE software reset occurs

This flag is cleared by writing this bit by “1”

Bit 3 ILLADDR: Illegal address reset flag
1= The previous reset event is due to illegal address occurs
0= No illegal address reset occurs
This flag is cleared by writing this bit by "1"

Bit 2 WDTRST: Watchdog timer reset flag
1= The previous reset event is due to watchdog timer reset
0= No watchdog timer reset occurs
This flag is cleared by writing this bit by 1.

Bit 1 LVRST: Low-voltage reset flag
1= The previous reset event is due to low-voltage reset
0= No low-voltage reset occurs
This flag is cleared by writing this bit by "1"

Bit 0 EXTRST: External reset flag
1= The previous reset event is due to external reset
0= No external reset occurs
This flag is cleared by writing this bit by "1"

Note:

The reset status register can be cleared by two consecutive writing cycles to prevent the status register from false writing. First, write 0x5a5a into this register and then, write the setting value into this register within 16 CPU clock cycles.

5.4 Design Tips

5.4.1 External Reset

The general reset circuit show in bellow:

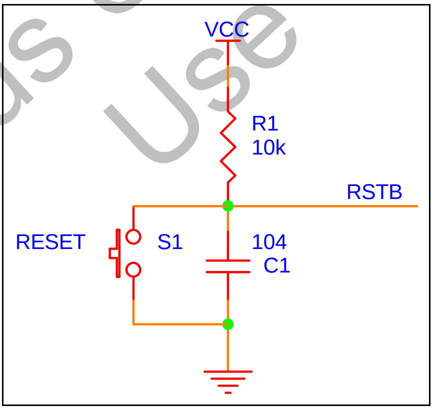


Figure 3-5 External Reset Circuit

In order to design a proper external reset circuit, user must consider the unstable power on VDD and RSTB pins. The following circuit (Figure 3-6) is the one we recommend for the design specification.

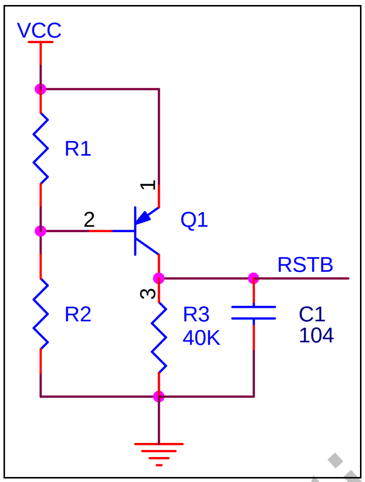


Figure 3-6 External Reset Circuit

Note:

The transistor Q1 turns off when VCC is below a certain level:

VCC * R1/(R1+R2) = 0.7 V

If we select VDD = 2.5V, we can use R1= 140K and R2 = 360K.

Note, this circuit will lose the fix current if user have power consumption issue. Because the R1 and R2 will form a DC path, and the current expense VDD/(R1+R2) = 3.3V(140K+360K) = 6.6 uA

5.4.2 Watchdog Timer Reset Timer (WDTTMR)

The procedure of watchdog reset configuration is shown as the follows.

【**Example3-1**】Set watchdog timer as 168ms and enable WDTR.

A) Use assemble language	B) Use C language
R1 = 0x5a5a	*P_WatchDog_Ctrl = 0x5a5a;
[P_WatchDog_Ctrl] = R1	*P_WatchDog_Ctrl = C_WDTTMR_168ms;
R1 = 0x0006	
[P_WatchDog_Ctrl] = R1	

Note:

The procedure of programming is performed with two consecutive writing cycles. First write the confirming command "0x5a5a" and the second write cycle programs these registers successfully. If the duration of the first write and second write is more than 16 CPU clock cycles, the program sequence fails.

【**Example3-2**】: Clear WDT within the selected period of the watchdog timer.

A) Use assemble language	B) Use C language
R1 = 0x5a5a	Set_WatchDog_Clr();
[P_WatchDog_Clr] = R1	

5.4.3 Reset Status

The reset status will show the type of reset. User can use this register to check the reset status. The system can recover the condition before reset if the software is programmed properly. User can add the checking procedure at the beginning of program as follows.

【Example3-3】:

```
F_Reset_Status_Check:
Check_POR_RST:
        R1 = [P_Reset_Status]       //Check POR
        CMP R1, 0x0000
        JE   Power_On_Reset
Check_EXT_RST:
        R2 = R1 & 0x0001
        JNZ   External_Reset        //Check External Reset
Check_LVR_RST:
        R2 = R1 & 0x0002
        JNZ   Low_Voltage_Reset       //Check Low Voltage Reset
Check_WDT_RST:
        R2 = R1 & 0x0004
        JNZ   Watch_Dog_Reset         //Watchdog Rest
Check_IAR_RST:
        R2 = R1 & 0x0008
        JNZ Illegal_Address_Reset     //Illegal Address Reset
Check_ICE_RST:
        R2 = R1 & 0x0010
        JNZ   ICE_Reset             //ICE Software Reset
Check_PCI_RST:
        R2 = R1 & 0x0020
        JNZ PCIRST                    //PCI Rset
        RETF

Power_On_Reset:
         ……
        JMP End_Reset_Flag_Check
External_Reset:
        R1 = 0x5a5a                 //Confirm write
        [P_Reset_Status] =     R1
        R1 = 0x0001                 //Erase External Reset Flag
        [P_Reset_Status] = R1
        ……
        JMP Check_LVR_RST
Low_Voltage_Reset:
        R1 = 0x5a5a                 //Confirm write
        [P_Reset_Status] = R1
        R1 = 0x0002                //Erase Low Voltage Reset Flag
        [P_Reset_Status] = R1
        ……
        JMP Check_WDT_RST
Watch_Dog_Reset:
        R1 = 0x5a5a                 //Confirm write
        [P_Reset_Status] = R1
        R1 = 0x0004                 //Erase Watch Dog Reset Flag
        [P_Reset_Status] = R1
        ……
        JMP Check_IAR_RST
Illegal_Address_Reset:
        R1 = 0x5a5a                //Confirm write
        [P_Reset_Status] = R1
        R1 = 0x0008                //Erase Illegal Address Reset Flag
        [P_Reset_Status] = R1
        ……
        JMP Check_ICE_RST
ICE_Reset:
        R1 = 0x5a5a                //Confirm write
        [P_Reset_Status] = R1
        R1 = 0x0010                //Erase ICE Software Reset Flag
        [P_Reset_Status] = R1
        ……
        JMP Check_PCI_RST
PCI_Reset:
        R1 = 0x5a5a               //Confirm write
        [P_Reset_Status] = R1
        R1 = 0x0020               //Erase PCI Reset Flag
        [P_Reset_Status] = R1
        ……
End_Reset_Flag_Check:
```

Note:

The procedure of writing **P_Reset_Status** performs with two consecutive write cycles. Write the confirming command "0x5a5a" first, and the second write cycle can program these register successful. If the duration of the first write and second write is more than 16 CPU clock cycles, the program sequence fails.

5.5 Boot-up sequence

When power is turned on, option bits are read by the system. The option bits are stored in the first word of embedded flash information block (address = 0x8000). The first 11 bits are verification pattern, which is verified to assure the option bits are programmed properly. When power is turned on, reset signal is activated and verification pattern will be checked. If the verification pattern passes, the power-on-timer will count 16384 times based on 200KHz clock cycle and then the reset signal is deactivated. In case of verification pattern fails, reset signal is activated and flash cannot be accessed in free run mode. In ICE mode, reset is activated, but flash can still be accessed by IDE. Therefore, users can still program information block properly if verification pattern checking fails. Please check section 6 for **P_System_Option**.

6 MCU

6.1 Introduction

In SPMC 70xxx series, the kernel of MCU is u'nSP version 1.2 or above defined by Sunplus. The functions include:

1:16-bit data bus / 22-bit address bus

- a: 4M words (8M bytes) memory space
- b: 64 banks / 64k words per bank

2: Thirteen 16-bit registers

- a: 5 general registers (R1-R5)
- b: 4 secondary registers (SR1-SR4)
- c: 3 system registers (SP, SR, PC)
- d: inner registers (FR)

3: Ten interrupts

- a: 1 fast interrupt (FIQ)
- b: 8 normal interrupts (IRQ0-IRQ7)
- c: 1 software interrupt (BRK)
- d: Support IRQ nested mode with user-customized priority

4: Six address modes

- a: Immediate (I6/I16)
- b: Direct (A6/A16)
- c: Indirect+ auto indexing address (DS indirect)
- d: Relative (BP+IM6)
- e: Multiple indirect (PUSH/POP)
- f: Register

5: 16x16 multiplication & up to 16-level inner product operation

- a: Three multiplication mode: signed x signed, signed x unsigned, unsigned x unsigned
- b: 4 bits guard bit of inner product operation to avoid overflow
- c: Integer/Fraction mode

6: 1-bit division

- a: DIVS: divide the sign bit DIVQ: divide the quotient
- b: Divide 32-bit numerator and a 16-bit denominator

7: Effective-exponent detect operation (EXP)

8: Bit operation

- a: Bit test / set / clr / inv operation to full memory space or registers

9: Multi-cycles 16-bit shift operation

a: Support 32-bit shift with combining 2 shift instructions

10:Far Indirect JMP by MR register

11:Far Indirect Call by MR register

12:NOP operation

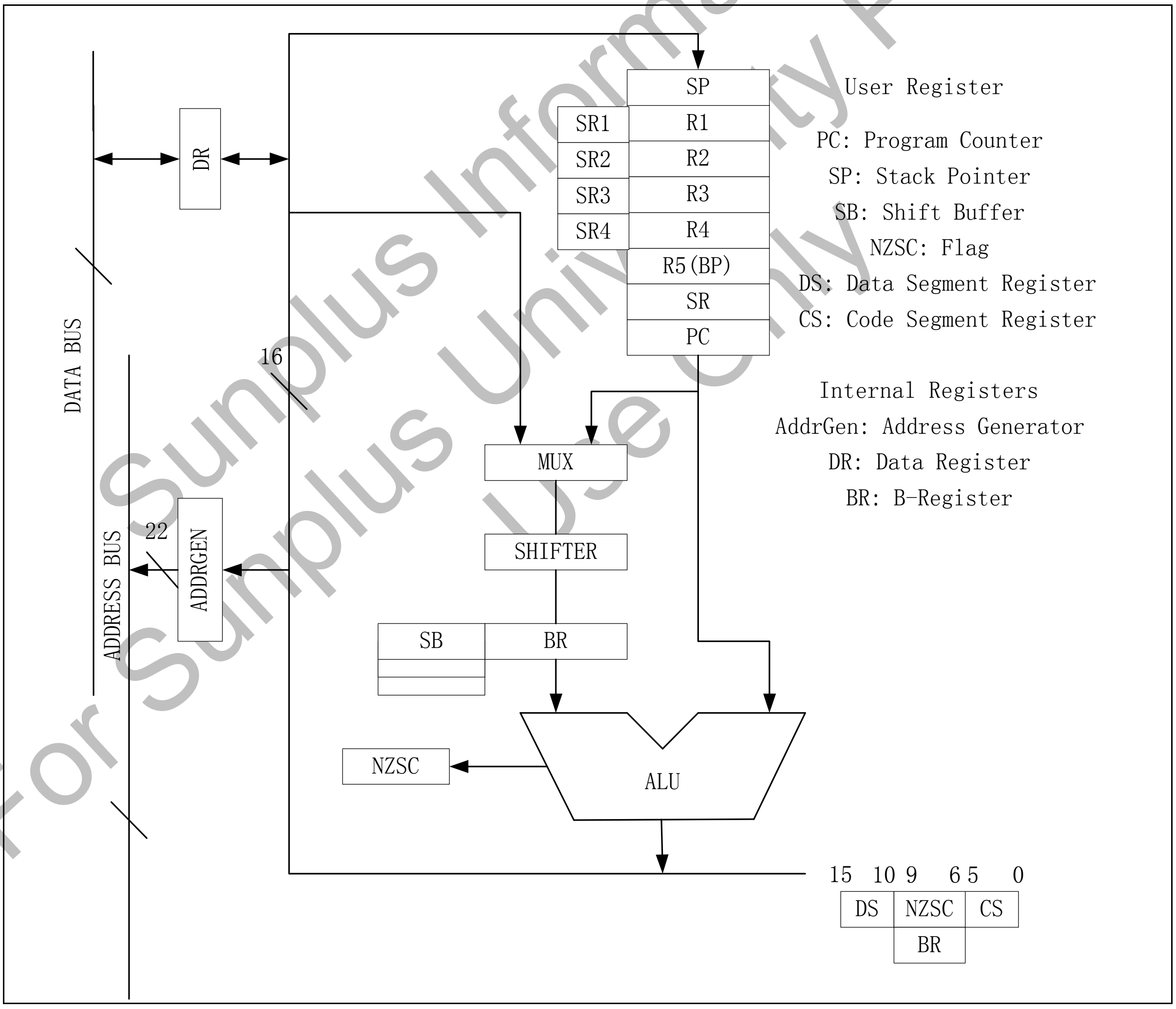
13: DS segment access instructions

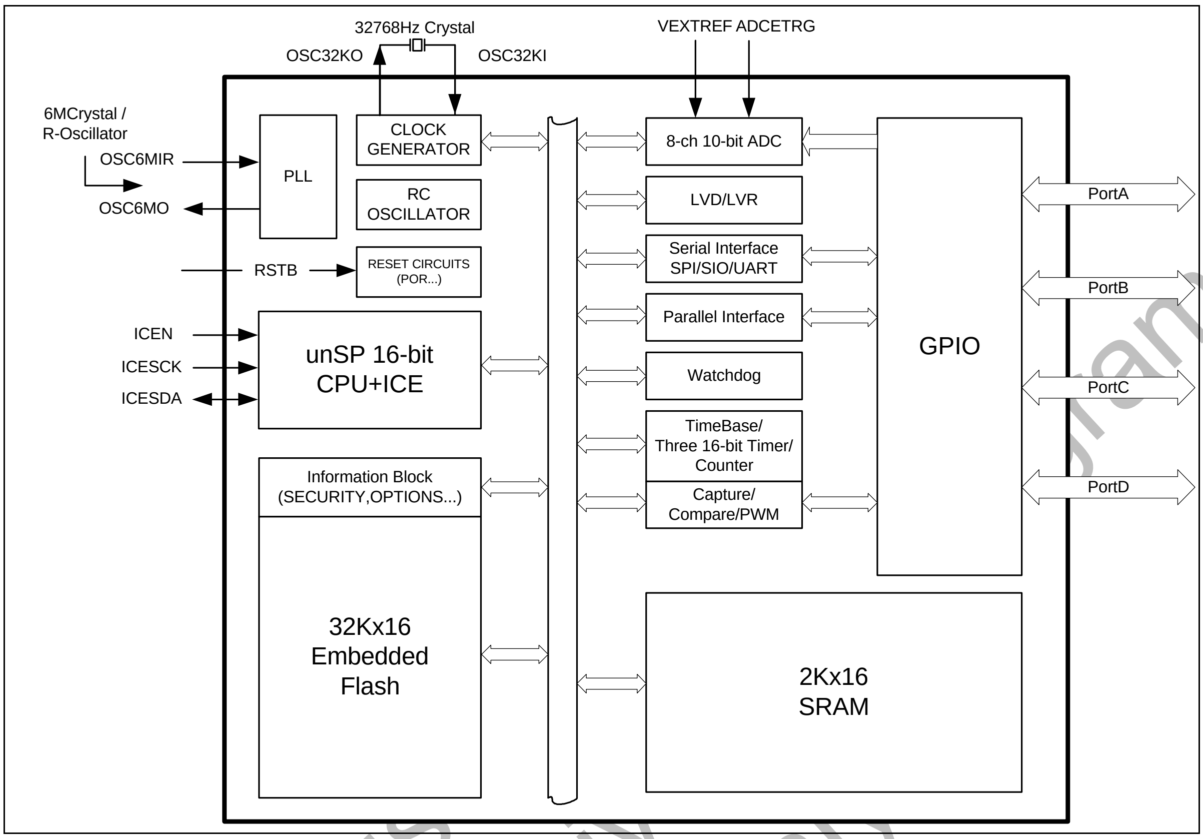
14: CPU inner flags access instructions

15: DMA function

16: Power Down/Sleep Mode

6.2 Function Block





6.3 CPU Instruction and Executions Cycle

The following table 4-1 shows all instruction types, operation styles, and execution cycles. The contents of table means:

1. RW: Memory Read Waiting cycle, RW = 0 if no wait state insertion and RW = N if wait state = N.
2. SW: Memory Write Waiting Cycle, SW= 0, except flash programming
3. RW : store or read waiting cycle, SRW = SW when ALU = store else SRW = RW.

Table 4-1 instruction types

Type	Operation	Cycles
JMPF	Goto label	5+2RW
DSI6	DS=I6	2+RW
JMPR	Goto MR	4+RW
CALL	CALL label	9+2RW+2SW
FIR_MOV	FIR_MOV_ON/OFF	2+RW
Fraction	Fraction ON/OFF	2+RW
INT SET	INT FIQ/IRQ	2+RW
IRQ	IRQ ON/OFF	2+RW
SECBANK	SECBANK ON/OFF	2+RW
FIQ	FIQ ON/OFF	2+RW
IRQ Nest Mode	IRQNEST ON/OFF	2+RW

Type	Operation	Cycles
BREAK	BREAK	10+2RW+2SW
CALLR	CALL MR	8+RW+2SW
DIVS	DIVS MR,R2	2+RW
DIVQ	DIVQ MR,R2	3+RW
EXP	R2= EXP R4	2+RW
NOP	NOP	2+RW
DS Access	DS=Rs/ Rs=Ds	2+RW
FR Access	FR=Rs/ Rs=FR	2+RW
MUL	MR = Rd* Rs,{ss,us,uu}	12+RW / 13+RW (uu)
MULS	MR = [Rd]*[Rs], size,{ss,us,uu}	us,ss : 10*N+6 + (N+1)*2*RW + {N*SW} / uu: 11*N+6 + (N+1)*2*RW + {N*SW}
Register BITOP	BITOP Rd,Rs	4+RW
Register BITOP	BITOP Rd,offset	4+RW
Memory BITOP	BITOP DS: [Rd],offset	7+2RW+SW
Memory BITOP	BITOP DS: [Rd],Rs	7+2RW+SW
Shift	Rd=Rd LSFT Rs	8+RW
RETI	RETI	8+3RW / 10+4RW (IRQ NEST ON)
RETF	RETF	8+3RW
Base+Disp6	Rd = Rd op [BP+IM6]	6+2RW
Imm6	Rd = Rd op IM6	2+RW
Branch	Jxx label	2+RW / 4+RW (taken)
Indirect	Push/Pop Rx,Ry to [Rs]	4+ 2N + (N+1)RW
DS_Indirect	Rd = Rd op DS: [Rs++]	6+RW+SRW / 7+RW+SRW (PC)
Imm16	Rd = Rs op IMM16	4+2RW / 5+2RW (PC)
Direct16	Rd = Rs op A16	7+2RW+SRW / 8+2RW+SRW (PC)
Direct6	Rd = Rd op A6	5+RW+SRW / 6+RW+SRW (PC)
Register	Rd = Rd op Rs SFT sfc	3+RW / 5+RW (PC)

(a) MULS Cycle: 10*N+6 + (N+1)*2*RW + {N*SW} (signedxsigned, unsignedxsigned)

11*N+6 + (N+1)*2*RW + {N*SW} (unsignedxunsigned) where N=1..16, (N*SW) = 0 if FIR_MOVE OFF

(b) DS_Indirect Cycle: 6 + RW + SRW / 7 + RW + SRW (write to PC)

(c) Direct16 Cycle: 7 + 2*RW + SRW / 8 + 2*RW + SRW (write to PC)

(d) Direct6 Cycle: 5+ RW +SRW / 6+ RW +SRW (write to PC)

(e) RW: Memory Read Waiting cycle, RW= 0 ~ N, SW: Memory Write Waiting Cycle, SW= 0 ~N.

SRW: store or read waiting cycle, SRW = SW when ALU = store else SRW = RW.

(f) D: 0 (forward jump) / 1 (backward jump)

W: 0 (not store) / 1 (store)

DS: 0 (not using DS) / 1 (using DS)

7 Memory Organization

7.1 Introduction

The memory of SPMC70xxx can be separated by three blocks: SRAM, I/O port registers, and the flash. The SRAM is used for stack, variable or data storage. The I/O port register is used to control the peripheral modules. The embedded flash is designed for programming code. In addition, the SPMC70xxx MCU has a 16-bit program counters, capable of addressing 64K x 16-bit. The width of address bus is assigned in A[21:0] so that the SPMC70xxx has the ability to address 4M* 16-bit memory. The memory allocation is shown in the Fig 5-1.

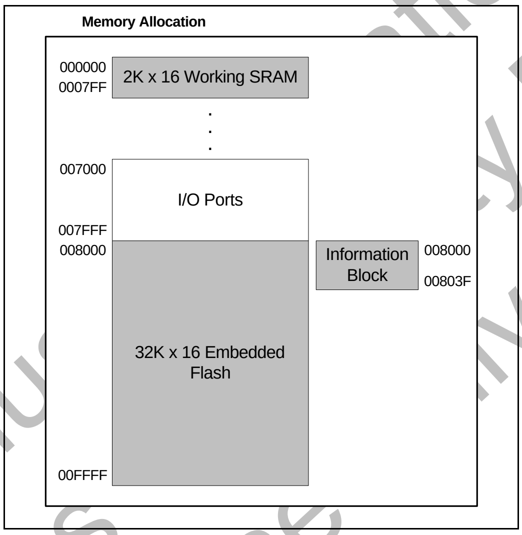


Fig 5-1 memory allocation

Note:

The address of 000800 – 006FFF and 010000 – 3FFFFF are reserved.

7.2 Flash

The SPMC70xxx has two blocks of flash: the information block and normal block. Only one of the two blocks can be addressed at the same time. The information block contains 64 words. The address of information block is mapped from 0x8000 ~ 0x803F. The 0x8000 is a system option register: **P_System_Option. The other address is used for user by storing important information such as version control, date, vender name, project name, etc. The information block can only be modified by ICE or writer. For more information, see Chapter 6.**

The 32K-word embedded flash is partitioned into 16 banks, 2K words each. Except the 00F800 - 00FFFF bank is read-only in free run mode, the other fifteen 2K-word banks can be programmed to be read-only or read-write in free run mode independently. Moreover, each 2K-word bank also can be separated by eight frames so that the 32K embedded flash can be divided to 128 frames. The user can erase each frame separately.

7.3 SRAM

The SRAM in SPMC70xxx series can be used for stack, variable and data storage. Stack is used for storing function call return address and pushing instruction data. The direction of stack goes from bottom to up. This stack is a FILO (first in last out) structure, and the stack address is indicated by stack pointer (SP).

The variable and data storage is configured by the user. Users can use direct access, indirect access or base pointer (BP) to load or save SRAM data. Please note that the stack and variable or storage data must not overlap each other; otherwise, program will run into a unknown status. The SPMC70xxx series addresses maximum 2K-word SRAM. The address range is from 0x0000 to 0x07ff. In addition, the stack pointer (SP) is allocated at the end of maximum address initially.

7.4 I/O port register

The I/O port registers are used by MCU and peripheral modules for performing the desired operation of device. The I/O port registers do not exist in all of the SPMC70xx series. User must check the individual specification of IC body when using I/O port register. The table 5-2 shows all I/O port register address and its alias name.

Table 5-2 I/O port register address

I/O Port Name	Address	I/O Port Name	Address	I/O Port Name	Address	I/O Port Name	Address
P_IOA_Data	$7000	P_IOC_Data	$7010		$7020		$7030
P_IOA_Buffer	$7001	P_IOC_Buffer	$7011		$7021		$7031
P_IOA_Dir	$7002	P_IOC_Dir	$7012		$7022		$7032
P_IOA_Attrib	$7003	P_IOC_Attrib	$7013		$7023		$7033
P_IOA_Latch	$7004	P_IOC_Latch	$7014		$7024		$7034
	$7005		$7015		$7025		$7035
	$7006		$7016		$7026		$7036
	$7007		$7017		$7027		$7037
P_IOB_Data	$7008	P_IOD_Data	$7018		$7028		$7038
P_IOB_Buffer	$7009	P_IOD_Buffer	$7019		$7029		$7039
P_IOB_Dir	$700A	P_IOD_Dir	$701A		$702A		$703A
P_IOB_Attrib	$700B	P_IOD_Attrib	$701B		$702B		$703B
P_IOB_Latch	$700C	P_IOD_Latch	$701C		$702C		$703C
	$700D		$701D		$702D		$703D
	$700E		$701E		$702E		$703E
	#700F		#701F		#702F		#703F

I/O Port Name	Address	I/O Port Name	Address	I/O Port Name	Address	I/O Port Name	Address
P_Timerbase_Setup	$7040		$7050	P_Int_Ctrl	$7060	P_ADC_Ctrl	$7070
P_Timerbase_Ctrl	$7041		$7051	P_Int_Priority	$7061	P_ADC_Data	$7071
	$7042		$7052	P_Int_Status	$7062		$7072
	$7043		$7053	P_WatchDog_Ctrl	$7063		$7073
P_Timer0_Ctrl	$7044		$7054	P_WatchDog_Clr	$7064		$7074
P_Timer0_preload	$7045		$7055	P_Reset_Status	$7065		$7075
P_Timer0_CCPR	$7046		$7056	P_LVDLVR_Ctrl	$7066		$7076
P_Timer0_CCPR2	$7047		$7057	P_LVD_Status	$7067		$7077
P_Timer1_Ctrl	$7048		$7058	P_Flash_RW	$7068		$7078
P_Timer1_preload	$7049		$7059	P_Clk_Ctrl	$7069		$7079
P_Timer1_CCPR	$704A		$705A	P_Wakeup_Source	$706A		$707A
P_Timer1_CCPR2	$704B		$705B	P_Wakeup_Source	$706B		$707B
P_Timer2_Ctrl	$704C		$705C	P_Wait_State	$706C		$707C
P_Timer2_preload	$704D		$705D	P_Stdby_Enter	$706D		$707D
	$704E		$705E	P_Halt_Enter	$706E		$707E
	#704F		#705F	P_Wakeup_Status	#706F		#707F

I/O Port Name	Address	I/O Port Name	Address	I/O Port Name	Address	I/O Port Name	Address

	$7080	P_SPI_Ctrl	$7090	P_UART_Ctrl	$70A0	P_SIO_Ctrl	$70B0
	$7081	P_SPI_Status	$7091	P_UART_Staus	$70A1	P_SIO_Status	$70B1
	$7082	P_SPI_Buf	$7092	P_UART_Reset	$70A2	P_SIO_Buf	$70B2
	$7083		$7093	P_UART_BaudrateL	$70A3	P_SIO_Addr	$70B3
	$7084		$7094	P_UART_BaudrateH	$70A4		$70B4
	$7085		$7095	P_UART_Tx_Buf	$70A5		$70B5
	$7086		$7096	P_UART_Rx_Buf	$70A6		$70B6
	$7087		$7097		$70A7		$70B7
	$7088		$7098		$70A8		$70B8
	$7089		$7099		$70A9		$70B9
	$708A		$709A		$70AA		$70BA
	$708B		$709B		$70AB		$70BB
	$708C		$709C		$70AC		$70BC
	$708D		$709D		$70AD		$70BD
	$708E		$709E		$70AE		$70BE
	#708F		#709F		#70AF		#70BF

I/O Port Name	Address	I/O Port Name	Address	I/O Port Name	Address	I/O Port Name	Address
P_PCI_Ctrl	$70C0		$70D0		$70E0		$7550
P_PCI_Status	$70C1		$70D1		$70E1		$7551
P_PCI_InBuf	$70C2		$70D2		$70E2		$7552
P_PCI_OutBUF	$70C3		$70D3		$70E3		$7553
P_PCI_COMM	$70C4		$70D4		$70E4		$7554
	$70C5		$70D5		$70E5	P_Flash_Ctrl	$7555
	$70C6		$70D6		$70E6		$7556
	$70C7		$70D7		$70E7		$7557
	$70C8		$70D8		$70E8		$7558
	$70C9		$70D9		$70E9		$7559
	$70CA		$70DA		$70EA		$755A
	$70CB		$70DB		$70EB		$755B
	$70CC		$70DC		$70EC		$755C
	$70CD		$70DD		$70ED		$755D
	$70CE		$70DE		$70EE		$755E
	#70CF		#70DF		#70EF		#755F

7.5 Reset and Interrupt Vector

A reset forces the program counter (PC) points to address 0Xfff7. When a device reset occurs, the program execution will branch to 0xfff7, named "Reset Vector Address".

The SPMC70xxx series has 10 interrupts. The address and function name list are given in the following table 5-3.

Table 5-3 interrupts

Address	Name	Function
$FFF5	**BRK**	**Software Interrupt**
$FFF6	**FIQ**	**Fast Interrupt**
$FFF7	**RST**	**Reset Vector**
$FFF8	**IRQ0**	**Normal Interrupt 0**
$FFF9	**IRQ1**	**Normal Interrupt 1**
$FFFA	**IRQ2**	**Normal Interrupt 2**
$FFFB	**IRQ3**	**Normal Interrupt 3**
$FFFC	**IRQ4**	**Normal Interrupt 4**
$FFFD	**IRQ5**	**Normal Interrupt 5**
$FFFE	**IRQ6**	**Normal Interrupt 6**
$FFFF	**IRQ7**	**Normal Interrupt 7**

8 Flash Organization and Control

8.1 Introduction

The SPMC70xxx series has two flash blocks: information block and normal block. Only one of the two blocks can be addressed at the same time. The information block contains 64 words. The address of information block is mapped from 0x8000 ~ 0x803F. The 0x8000 is a system option register **P_System_Option. The other addresses are used for storing important information such version control, date, vender name, project name etc. The information block's structure is in figure 6-1 and they only can be written in ICE mode or by writer.** The 32K words embedded flash is partitioned into 16 banks, 2K words each. Except the 00F800 - 00FFFF bank is read-only in free run mode, the other fifteen 2K-word banks can be programmed to be read-only or read-write for CPU in free run mode independently. In addition, each 2k-word bank also can be separated into eight frames; therefore, the 32K embedded flash can be divided into 128 frames. The user can erase each frame separately. The relation of page and frame of flash is shown in figure 6-2.

The width of address bus is assigned in A[21:0], so the SPMC70xxx has the ability to address 4M* 16-bit memory.

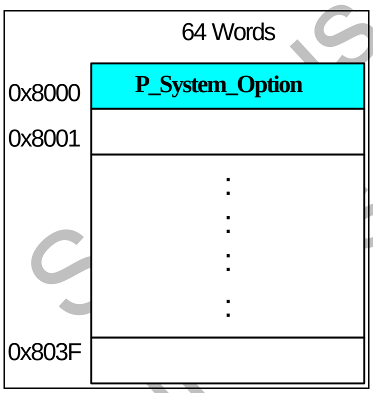


Figure 6-1 Structure of Information block

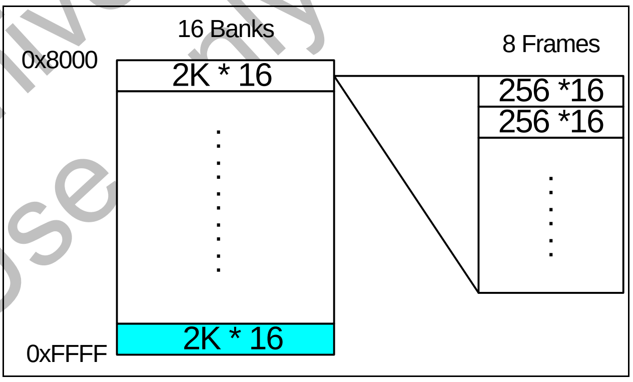


Figure 6-2 Page0 and Frame of Flash

8.2 Flash Operation

There are two registers for flash control: P_Flash_RW ($7068) and P_Flash_Ctrl ($7555). The flash access control port, P_Flash_RW ($7068), can be configured by two consecutive write cycles, keeping away from inadvertent writing. First, write $5a5a to P_Flash_RW, and then write the configuration data to P_Flash_RW within 16 clock cycles.

? **Example 6-1**? : Set bank14 as Read/Write Mode

A) Use assemble language

```
R1 = 0x5A5A;
[P_Flash_RW] = R1;
R1 = 0xBFFF;
[P_Flash_RW] = R1;
```

B) Use C language

```
Set_Flash_RW(C_FlashRW_CMD);
Set_Flash_RW(0xffff^C_BANK14);
```

The flash control register, P_Flash_Ctrl, is a write only register that is for accepting/performing flash command. The Table 6-1 shows the command function and access flow.

Table 6-1 command function and access flow

	Frame Erase	Program Mode	Sequential Program Mode
1 cycle	P_Flash_Ctrl = 0xAAAA		
2 cycle	[P_Flash_Ctrl] = 0x5511	[P_Flash_Ctrl] = 0x5533	[P_Flash_Ctrl] = 0x5544
3 cycle	Set Frame Address	Write Data	Write Data
4 cycle	Wait 20ms End - Auto	Wait 40us End - Auto	Wait 40us - Auto
			Go to 2 cycle
			[P_Flash_Ctrl]= 0xFFFF → Go to End

? **Example 6-2**? : Example for frame erasing:

A) Use assemble language

```
R1 = 0xAAAA;
[P_Flash_Ctrl] = R1;
R1 = 0x5511;
[P_Flash_Ctrl] = R1;
[0xF000] = R1; // Erase the first frame of bank 14
```

B) Use C language

```
unsigned int *P_WordAdr;
*P_Flash_Ctrl = C_FlashCMD;
*P_Flash_Ctrl = C_PageErase;
P_WordAdr = (unsigned int *)0xF000;
*P_WordAdr = 5;     //Erase the first frame of bank 14
```

? **Example 6-3**? : Example for program mode: Write 0x1234 to the address of 0xF000

A) Use assemble language

```
R1 = 0xAAAA;
[P_Flash_Ctrl] = R1;
R1 = 0x5533;
[P_Flash_Ctrl] = R1;
R1 = 0x1234;
[0xF000] = R1;
```

B) Use C language

```
unsigned int *P_WordAdr;
*P_Flash_Ctrl = C_FlashCMD;
*P_Flash_Ctrl = C_Program;
P_WordAdr = (unsigned int *)0xF000;
*(unsigned int *)P_WordAdr = 0x1234;
```

Note:

The characteristic of flash is that the data bit can only be programmed from 1 to 0, but it is not allowed to be from 0 to 1. Therefore, if users intend to program flash, the mass erase or page erase instruction must be executed first, which erase data bit from 0 to 1.

? Example 6-4? Example for sequential program mode: Write data to flash with sequential program mode, address is from 0xF000 to 0xF020

A) Use assemble language

```
    R2 = 0x1;
    R3 = 0x8500;
    R1 = 0xAAAA;
    [P_Flash_Ctrl] = R1;

Loop_Program:
    R1 = 0x5544;
    [P_Flash_Ctrl] = R1;
    [R3++] = R2;          //User input data
    R2 += 1;
    CMP R3,0xF020;
    JBE Loop_Program
    R1 = 0xFFFF
    [P_Flash_Ctrl] = R1;
```

A) Use C language

```
    unsigned int i,uiData=1;
    *P_Flash_Ctrl = C_FlashCMD;
    for(i=0xF000;i<=0xF020;i++)
    {
        *P_Flash_Ctrl = C_Sequential;
        P_WordAdr = (unsigned int *)i;
        *(unsigned int *)P_WordAdr = uiData;
        uiData ++;
    }
    *P_Flash_Ctrl = C_SequentialEnd;
```

8.3 Control Register

8.3.1 P_Flash_RW ($7068): Embedded flash access control

b15	b14	b13	b12	b11	b10	b9	b8
R	R/W	R/W	R/W	R/W	R/W	R/W	R/W
1	0	0	0	0	0	0	0
-	BK14WENB	BK13WENB	BK12WENB	BK11WENB	BK10WENB	BK9WENB	BK8WENB

b7	b6	b5	b4	b3	b2	b1	b0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
BK7WENB	BK6WENB	BK5WENB	BK4WENB	BK3WENB	BK2WENB	BK1WENB	BK0WENB

This port sets up banks as Read only or Read/Write mode.

	Type(default)	Name	Frame Range	Description
B[15]	R (0)	Bank 15	Frame 120~127	F800h-FFFFh is Read-only
B[14]	R/W (0)	Bank 14	Frame 112~119	F000h-F7FFh access control 1= Read-only 0= Read/write
B[13]	R/W (0)	Bank 13	Frame 104~111	E800h-EFFFh access control 1= Read-only 0= Read/write
B[12]	R/W (0)	Bank 12	Frame 96~103	E000h-E7FFh access control 1= Read-only

	Type(default)	Name	Frame Range	Description
				0= Read/write
B[11]	R/W (0)	Bank 11	Frame 88~95	D800h-DFFFh access control 1= Read-only 0= Read/write
B[10]	R/W (0)	Bank 10	Frame 80~87	D000h-D7FFh access control 1= Read-only 0= Read/write
B[9]	R/W (0)	Bank 9	Frame 72~79	C800h-CFFFh access control 1= Read-only 0= Read/write
B[8]	R/W (0)	Bank 8	Frame 64~71	C000h-C7FFh access control 1= Read-only 0= Read/write
B[7]	R/W (0)	Bank 7	Frame 56~63	B800h-BFFFh access control 1= Read-only 0= Read/write
B[6]	R/W (0)	Bank 6	Frame 48~55	B000h-B7FFh access control 1= Read-only 0= Read/write
B[5]	R/W (0)	Bank 5	Frame 40~47	A800h-AFFFh access control 1= Read-only 0= Read/write
B[4]	R/W (0)	Bank 4	Frame 32~39	A000h-A7FFh access control 1= Read-only 0= Read/write
B[3]	R/W (0)	Bank 3	Frame 24~31	9800h-9FFFh access control 1= Read-only 0= Read/write
B[2]	R/W (0)	Bank 2	Frame 16~23	9000h-97FFh access control 1= Read-only 0= Read/write
B[1]	R/W (0)	Bank 1	Frame 8~15	8800h-8FFFh access control 1= Read-only 0= Read/write
B[0]	R/W (0)	Bank 0	Frame 0~7	8000h-87FFh access control 1= Read-only 0= Read/write

Note:

The flash access control port, P_Flash_RW ($7068), can be configured by two consecutive write cycles to keep away from inadvertent writing. First, write $5a5a to P_Flash_RW, and then write the configuration data to P_Flash_RW within 16 clock cycles.

8.3.2 P_Flash_Ctrl ($7555): Embedded flash access control

b15	b14	b13	b12	b11	b10	b9	b8
W	W	W	W	W	W	W	W
0	0	0	0	0	0	0	0
FlashCtrl							

b7	b6	b5	b4	b3	b2	b1	b0
W	W	W	W	W	W	W	W
0	0	0	0	0	0	0	0
FlashCtrl							

This port is used to issue flash command. Please see the table 6-1.

8.3.3 P_Wait_State ($706c): Wait State Number

b15	b14	b13	b12	b11	b10	b9	b8
-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-

b7	b6	b5	b4	b3	b2	b1	b0
-	-	-	-	R/W	R/W	R/W	R/W
-	-	-	-	0	0	0	0
-	-	-	-	WSCYCT			

Bit 15:4 Reserved

Bit 3: WSCYCT: CPU wait-state clock cycle time

Note:

The WSCYCT is applied to be the number of inserted clock cycle for CPU read operation with flash, which has the slower read operation timing. This port is supposed to be used when system clock operates at 48MHz mode. In this mode, the value of WSCYCT is suggested to be set to 0x0001.

8.3.4 P_System_Option ($8000): System Option Register

This address is allocated at information block, not in the general flash address. User must use ICE function or writer to set this port. For detailed description, please check section 6.

b15	b14	b13	b12	b11	b10	b9	b8
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	1	0	1	0	1	0	1
Verification Pattern							

b7	b6	b5	b4	b3	b2	b1	b0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	1	0	1	0	1	1	1
Verification Pattern			Security	LVD	LVR	WDG	CLK Source

Bit 15:5 Verification Pattern

ICE or Writer will check this area and write 01010101010 to this area

Bit 4 Security: security selection bit

0: protected, the normal block in the flash cannot be accessed

1: not protect, can be readable or write-able

Bit 3 LVD: enable low voltage detection function

0: enable

1: disable

Bit2 LVR: enable low voltage reset function

0: disable

1: enable

Bit1 WDG: enable watchdog function

0: disable

1: enable

Bit0 CLK Source: Clock Source Selection

0: R oscillator

1: crystal oscillator

Note:

If the voltage reset and low voltage detection are intended to be enabled or disabled, they must be set by ICE or writer.

8.4 Flash Security protection

In case of security option in information block is activated, SPMC70xxx series are protected from reading data through ICE or Writer function. If the security bit is turned on and under ICE enable mode, the flash main block does not allow to be accessed but information block enable to be read by ICE. In addition, SRAM cannot be accessed (read/write) in ICE enable mode.

The "mass erase" command users can perform exclusively is to erase flash data and clear security bit. The "mass erase" command execute on main block is to erase main block only, but erase main block and information block if the command execute on information block. The above limitations don't exist in free run mode. In normal operation (ICEN = 0), CPU can access the flash data and the working SRAM. The ICE cannot program the flash memory when the ICE mode is activated and security is turned on. This hardware protection prevents users from downloading a program to flash and change to normal mode to write flash data out to GPIOs. The figure 6-3 shows the logical block of flash security.

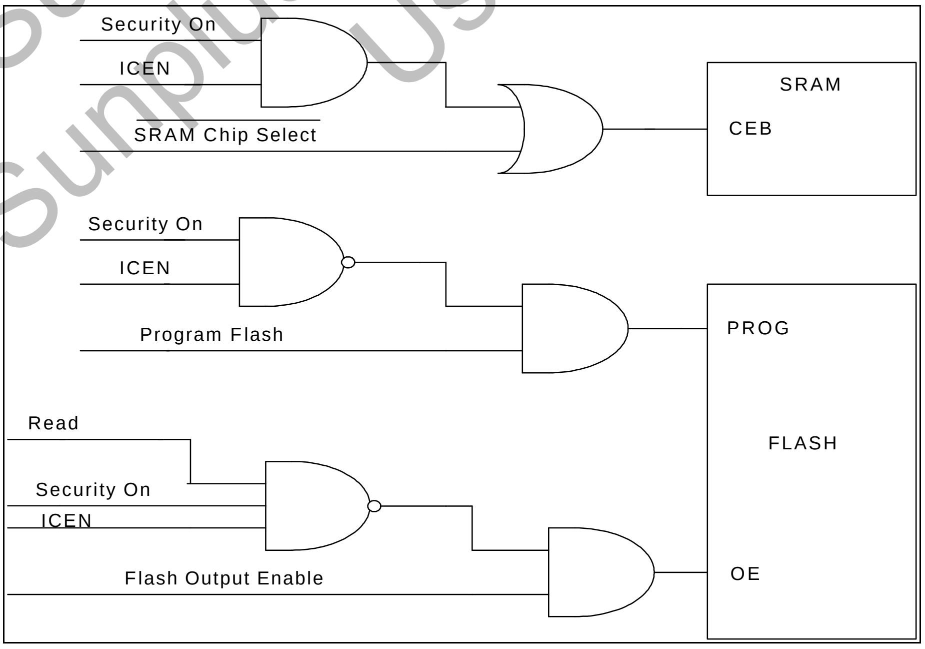


Figure 6-3 Flash security block

9 Interrupt

9.1 Introduction

The SPMC70xxx series MCU has 16 interrupt sources. These 16 interrupt source can be grouped into two types, FIQ (Fast Interrupt Request) and IRQ0~IRQ7 (Interrupt request). The FIQ is the high-priority interrupt and the IRQ is the low-priority one. On the other hand, an IRQ can be interrupted by a FIQ, but not by another IRQ. A FIQ cannot be interrupted by any other interrupt sources. The current interrupts are listed in table 7.1. In the table, it shows the interrupt source, interrupt name, IRQ number, and FIQ selection. The source of FIQ is formed by ADC, External1 and External2 interrupts, UART, SPI, SIO, PCI, Timer0~2, LVD, and Key Interrupts. The IRQ and FIQ selections are set by "P_INT_Priority" register.

Table 7-1

Interrupt Source	Interrupt Name	IRQ NO.	FIQ Selection
ADC	ADCIF	IRQ0	Yes
EXT2	EXT2IF	IRQ1	Yes
EXT1	EXT1IF	IRQ1	Yes
PCI	PCIIF	IRQ2	Yes
UART	UARTIF	IRQ3	Yes
SPI	SPIIF	IRQ3	Yes
SIO	SIOIF	IRQ3	Yes
Timer2	TMR2IF	IRQ4	Yes
Timer1	TMR1IF	IRQ4	Yes
Timer0	TMR0IF	IRQ4	Yes
LVD	LVDIF	IRQ5	Yes
Key change / wakeup	KEYCIF	IRQ5	Yes
TimeBase2	TMB2IF	IRQ6	No
TimeBase1	TMB1IF	IRQ6	No
4HZ	4HZIF	IRQ7	No
2HZ	2HZIF	IRQ7	No

The definitions and usages related to the interrupt configuration are listed as follows.

9.2 Control Register

9.2.1 P_INT_Ctrl $7060: Interrupt control register

This port can be set to enable interrupt. Write "1" to any bit to enable the interrupt.

b15	b14	b13	b12	b11	b10	b9	b8
-	-	-	-	-	R/W	R/W	R/W
-	-	-	-	-	0	0	0
-	-	-	-	-	EXT2INEN	EXT1INEN	EXT2IEN

b7	b6	b5	b4	b3	b2	b1	b0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
EXT1IEN	EXT2IEDG	EXT1IEDG	KEYIEN	TMB2IEN	TMB1IEN	4HZIEN	2HZIEN

Bit 15:11 Reserved

Bit 10 EXT2INEN: EXT2 input enable

1= Enable

0= Disable

Bit 9 EXT1INEN: EXT1 input enable

1= Enable

0= Disable

Bit 8 EXT2IEN: EXT2 interrupt enable

1= Enable

0= Disable

Bit 7 EXT1IEN: EXT1 interrupt enable

1= Enable

0= Disable

Bit 6 EXT2IEDG: EXT2 interrupt trigger mode

1= EXT2 interrupt trigger on rising edge

0= EXT2 interrupt trigger on falling edge

Bit 5 EXT1IEDG: EXT1 interrupt trigger

1= EXT1 interrupt trigger on rising edge

0= EXT1 interrupt trigger on falling edge

Bit 4 KEYIEN: Key-change Interrupt Enable

1= Enable

0= Disable

Bit 3 TMB2IEN: Time base 2 interrupt enable

1= Enable

0= Disable

Bit 2 TMB1IEN: Time base 1 interrupt enable

1= Enable

0= Disable

Bit 1 4HZIEN: 4Hz interrupt enable

1= Enable

0= Disable

Bit 0 2HZIEN: 2Hz interrupt enable

1= Enable

0= Disable

9.2.2 P_INT_Priority ($7061): IRQ and FIQ selection

This port can set interrupt source as IRQ or FIQ. The default interrupt source is IRQ.

b15	b14	b13	b12	b11	b10	b9	b8
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
ADCIP	EXT2IP	EXT1IP	UARTIP	SPIIP	SIOIP	PCIIP	TMR2IP

b7	b6	b5	b4	b3	b2	b1	b0
R/W	R/W	R/W	R/W	-	-	-	-
0	0	0	0	-	-	-	-
TMR1IP	TMR0IP	LVDIP	KEYIP	-	-	-	-

Bit 15	ADCIP: ADC interrupt priority
	1= FIQ
	0= IRQ0
Bit 14	EXT2IP: EXT2 interrupt priority
	1= FIQ
	0= IRQ1
Bit 13	EXT1IP: EXT1 interrupt priority
	1= FIQ
	0= IRQ1
Bit 12	UARTIP: UART interrupt priority
	1= FIQ
	0= IRQ3
Bit 11	SPIIP: SPI interrupt priority
	1= FIQ
	0= IRQ3
Bit 10	SIOIP: SIO interrupt priority
	1= FIQ
	0= IRQ3
Bit 9	PCIIP: PCI interrupt priority
	1= FIQ
	0= IRQ2
Bit 8	TMR2IP: Timer2 interrupt priority
	1= FIQ
	0= IRQ4
Bit 7	TMR1IP: Timer1 interrupt priority
	1= FIQ
	0= IRQ4
Bit 6	TMR0IP: Timer0 interrupt priority
	1= FIQ
	0= IRQ4

Bit 5 LVDIP: Low-voltage detect interrupt priority

1= FIQ

0= IRQ5

Bit 4 KEYIP: Key-change interrupt priority

1= FIQ

0= IRQ5

Bit 3:0 Reserved

Note:

This register is used for configuring the interrupt as IRQ or FIQ separately. This register can only set ADC, External1 and External2 interrupts, UART, SPI, SIO, PCI, Timer0~2, LVD, and Key Interrupt. The timebase1 and timebase2 interrupts can only be set to IRQ6, 2Hz, but 4Hz interrupt can only be set to IRQ7.

9.2.3 P_INT_Status ($7062): Interrupt status

The port passes the interrupt flags of TMR0, TMR1, TMR2, UART, parallel port, SPI, SIO and records the interrupt flags from RTC, analog-to-digital converter, EXT1, EXT2, LVD. The port is used for software polling when all interrupts are disabled.

b15	**b14**	**b13**	**b12**	**b11**	**b10**	**b9**	**b8**
R	**R/W**	**R/W**	**R**	**R**	**R**	**R**	**R**
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**
ADCINT	**EXT2INT**	**EXT1INT**	**UARTINT**	**SPIINT**	**SIOINT**	**PCIINT**	**TMR2INT**

b7	**b6**	**b5**	**b4**	**b3**	**b2**	**b1**	**b0**
R	**R**	**R**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**
TMR1INT	**TMR0INT**	**LVDINT**	**KEYINT**	**TMB2INT**	**TMB1INT**	**4HZINT**	**2HZINT**

Bit 15 ADCINT: ADC interrupt flag

1= ADC interrupt occurs

0= No ADC interrupt

This bit passes the ADCIF bit, which is stored in ADC control module.

Bit 14 EXT2INT: EXT2 interrupt flag

1= EXT2 interrupt occurs

0= No EXT2 interrupt

This bit is set when EXT2 rising/falling edge is detected and is cleared by software by writing this bit "1".

Bit 13 EXT1INT: EXT1 interrupt flag

1= EXT1 interrupt occurs

0= No EXT1 interrupt

This bit is set when EXT1 rising/falling edge is detected and is cleared by software by writing this bit "1".

Bit 12 UARTINT: UART interrupt flag

1= UART interrupt occurs

0= No UART interrupt

This bit is stored in UART module also.

Bit 11 SPIINT: SPI interrupt flag

1= SPI interrupt occurs

0= No SPI interrupt

This bit is stored in SPI module also.

Bit 10 SIOINT: SIO interrupt flag

1= SIO interrupt occurs

0= No SIO interrupt

This bit is stored in SIO module also.

Bit 9 PCIINT: PCI interrupt flag

1= PCI interrupt occurs

0= No PCI interrupt

This bit is stored in PCI module also.

Bit 8 TMR2INT: Timer2 interrupt flag

1= Timer2 interrupt occurs

0= No Timer2 interrupt

This bit is stored in Timer2 module also.

Bit 7 TMR1INT: Timer1 interrupt flag

1= Timer1 interrupt occurs

0= No Timer1 interrupt

This bit is stored in Timer1 module also.

Bit 6 TMR0INT: Timer0 interrupt flag

1= Timer0 interrupt occurs

0= No Timer0 interrupt

This bit is stored in Timer0 module also.

Bit 5 LVDINT: Low-Voltage Detect interrupt flag

1= Low-voltage detect interrupt occurs

0= No low-voltage detect interrupt

This bit is stored in Low-voltage detect control module also.

Bit 4 KEYINT: Key-change Interrupt flag

1= Key-change interrupt occurs

0= No Key-change interrupt

This bit is set when KEYCHANG rising edge is detected and is cleared by software writing this bit by 1.

Bit 3 TMB2INT: Timebase 2 interrupt flag

1= Timebase2 interrupt occurs

0= No Timerbase2 interrupt

This bit is set when TMB2 rising edge is detected and is cleared by software writing this bit by 1.

Bit 2 TMB1INT: Timebase 1 interrupt flag

1= Timebase1 interrupt occurs

0= No Timerbase1 interrupt

This bit is set when TMB1 rising edge is detected and is cleared by software writing this bit by 1.

Bit 1 4HZINT: 4Hz interrupt flag

1= 4Hz interrupt occurs

0= No 4Hz interrupt

This bit is set when 4Hz rising edge is detected and is cleared by software writing this bit by 1.

Bit 0 2HZINT: 2Hz interrupt flag

1= 2Hz interrupt occurs

0= No 2Hz interrupt

This bit is set when 2Hz rising edge is detected and is cleared by software writing this bit by 1.

Note:

1. Analog-to-Digital Converter interrupt is recorded when a conversion of ADC is completed, which is recognized through the READY signal from ADC module.
2. External interrupts EXT1, EXT2 need two registers to record its interrupt flags.
3. Write the corresponding status bits of EXT1 and EXT2 by “1” to clear the interrupt flags.
4. All interrupts are level-trigger type. That is, an interrupt flag has to be clear after interrupt service is completed.

9.2.4 P_TimeBase_Setup($7040H)

b15	b14	b13	b12	b11	b10	b9	b8
-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-

b7	b6	b5	b4	b3	b2	b1	b0
-	-	-	R/W	R/W	R/W	R/W	R/W
-	-	-	0	0	0	0	0
-	-	-	TMBENB	TMB2FS		TMB1FS	

Bit 15:5 Reserved

Bit 4 TMBENB: Timebase enable / disable

1= Disable

0= Enable

Bit 3:2 TMB2FS : Timebase2 Frequency Selection

00= 128Hz

01= 256Hz

10= 512Hz

11= 1024Hz

Bit 1:0 TMB1FS: Timebase1 Frequency Selection

00= 8Hz

01= 16Hz

10= 32Hz

11= 64Hz

9.2.5 P_TimeBase_Clr($7041H)

b15	b14	b13	b12	b11	b10	b9	b8
W	W	W	W	W	W	W	W
0	1	0	1	0	1	0	1
TMBCLR							

b7	b6	b5	b4	b3	b2	b1	b0
W	W	W	W	W	W	W	W
0	1	0	1	0	1	0	1
TMBCLR							

An additional control port, P_Timebase_Clr(W)($7041H), is for time reset and precise correction, which is activated by write "0x5555H" to this port.

9.3 Information Saving During Interrupts

During an interrupt, the returned PC value and SR registers (status register) is saved on the stack by CPU core. Typically, users may save key information registers during an interrupt e.g. R1~R5 register .This has to be implemented by software programming. The action of saving information is commonly referred to as "PUSHing," while the action of restoring the information before the interrupt return is commonly referred to as "POPing." These (PUSH, POP) are not instruction mnemonics, but are conceptual actions. Any interrupt should be returned by "RETI" instruction, or the stack will crash when stack pointer overlaps with data area.

? **Example 7-1**? stores and restores the STATUS and R1~R4 registers for devices with common RAM.

```
push   r1,r5  to [sp]
       ……
pop    r1,r5  from [sp]
reti
```

9.4 Interrupts prototype

The following instruction shows the prototype of the interrupt service routine in SPMC70xx. The name of the interrupt entry points is reserved by the compiler. Please do not change the name of entry points.

A) Use assemble language

```
//*****************************************************************
// Function: Fast Interrupt Service routine Area
//      Service for   (1)FIQ
//                    (2)IRQ 0 ~ IRQ 7
//         User's FIQ must hook on here
//   _FIQ:                  // Fast interrupt entrance
//   _IRQ1:                 // interrupt entrance
//   _IRQ2:                 // interrupt entrance
//   _IRQ3:                 // interrupt entrance
//   _IRQ4:                 // interrupt entrance
//   _IRQ5:                 // interrupt entrance
//   _IRQ6:                 // interrupt entrance
//   _IRQ7:                 // interrupt entrance
//*****************************************************************
.include SPMC701FM0.inc             // include IO information
.TEXT
.public _BREAK;
.public _FIQ;
.public  _IRQ0,_IRQ1, _IRQ2,_IRQ3,_IRQ4, _IRQ5_IRQ6, _IRQ7

//===================================================
//Function: Interrupt Service routine Area
//Service for FIQ
//===================================================
_FIQ:
          push r1,r5 to [sp];
          //------------------------------------------------
          //Add BREAK Function

          //-------------------------------------------------
          pop r1,r5 from [sp];
           reti;

//===================================================
// Function: Interrupt Service routine Area
// Service for          IRQ1 - IRQ7
// User's IRQ must hook on here
//===================================================
_BREAK:
         push r1,r5 to [sp];
         //------------------------------------------------
         //Add BREAK Function

         //------------------------------------------------
         pop r1,r5 from [sp];
         reti;
```

```
//=====================================================
_IRQ0:
        push r1,r5 to [sp];
        //-------------------------------------------------
        //Add ADC Function

        //-------------------------------------------------
        pop r1,r5 from [sp];
         reti;

//=====================================================
_IRQ1:
        push r1,r5 to [sp];
        //-------------------------------------------------
        //Add EXT1 and EXT2 Function

        //-------------------------------------------------
        pop r1,r5 from [sp];
         reti;

//=====================================================
_IRQ2:
        push r1,r5 to [sp];
        //-------------------------------------------------
        //Add PCI Function

        //-------------------------------------------------
        pop r1,r5 from [sp];
         reti;

//=====================================================
_IRQ3:
        push r1,r5 to [sp];
        //-------------------------------------------------
        //Add UART,SPI and SIO Function

        //-------------------------------------------------
        pop r1,r5 from [sp];
         reti;

//=====================================================
_IRQ4:
        push r1,r5 to [sp];
        //-------------------------------------------------
        //Add Timer0,Timer1 and Timer2 Function

        //-------------------------------------------------
        pop r1,r5 from [sp];
         reti;

//=====================================================
_IRQ5:
        push r1,r5 to [sp];
        //-------------------------------------------------
        //Add LVD and Key Change and Wakeup Function
        //--------------------------------------------------
        pop r1,r5 from [sp];
         reti;

//=====================================================
_IRQ6:
        push r1,r5 to [sp];
        //--------------------------------------------------
        //Add Timebase1 and Timebase2 Function

        //--------------------------------------------------
        pop r1,r5 from [sp];
       reti;

//=====================================================
_IRQ7:
        push r1,r5 to [sp];
        //--------------------------------------------------
        //Add 2Hz and 4Hz Function

        //--------------------------------------------------
        pop r1,r5 from [sp];
         reti;

//=====================================================
// End of isr.asm
//=====================================================
```

A) Use C language

```
void BREAK(void) __attribute__ ((ISR));
void BREAK(void)
{
}
void FIQ(void) __attribute__ ((ISR));
void FIQ(void)
{
}
void IRQ0(void) __attribute__ ((ISR));
void IRQ0(void)
{
}
void IRQ1(void) __attribute__ ((ISR));
void IRQ1(void)
{
}
void IRQ2(void) __attribute__ ((ISR));
void IRQ2(void)
{
}
void IRQ3(void) __attribute__ ((ISR));
void IRQ3(void)
{
}
void IRQ4(void) __attribute__ ((ISR));
void IRQ4(void)
{
}
void IRQ5(void) __attribute__ ((ISR));
void IRQ5(void)
{
}
void IRQ6(void) __attribute__ ((ISR));
void IRQ6(void)
{

}

void IRQ7(void) __attribute__ ((ISR));
void IRQ7(void)
{

}
```

9.5 Design Tips

When using interrupt source, the steps given below must be followed.

1. Enable Interrupt through the following control registers:

   P_Int_Ctrl ($7060)

   P_LVDLVR_Ctrl ($7066)

   P_PCI_Ctrl ($70C0)

   P_Timer0_Ctrl ($7044), P_Timer1_Ctrl ($7048), P_Timer2_Ctrl ($704C)

   P_SPI_Ctrl ($7090)

   P_SIO_Ctrl ($70B0)

   P_UART_Ctrl ($70A0)

   P_ADC_Ctrl ($7070)

2. Set P_INT_Priority ($7061) to configure IRQ and FIQ selection for interrupt

3. Use the following instructions to enable IRQ or FIQ.

   a. IRQ ON

   b. INT IRQ

   c. INT FIQ

   d. INT FIQ, IRQ

【Example 7-2】: **Test 2Hz interrupt . Every 0.5s light the LED from IOD15 or IOD14.It**

```
//===================== start of main.c ===============================//
        #include        "SPMC701FM0.H"
        #include        "unSPMACRO.H"
        #define         C_PORT_IOD14  0x4000
        #define         C_PORT_IOD15  0x8000

        int   g_iFlag_IRQ7 = 0x0000;      //Defined for IRQ7 flag
```

```
            main()
            {
                  Set_IOD_Dir(C_PORT_IOD15+C_PORT_IOD14);        //Set IOD14,IOD15 as output PORT
                  Set_IOD_Attrib(C_PORT_IOD15+C_PORT_IOD14);
                  Set_IOD_Data(C_PORT_IOD15+C_PORT_IOD14);

                  Set_INT_Ctrl(C_2HZIEN);                 //Enable the 2Hz interrupt

                  IRQ_ON();                               //Open the IRQ

                  while(1)
                  {
                         if (g_iFlag_IRQ7)       //If the g_iFlag_IRQ7 flag is not 0, set IOD15 as high;else set IOD14 as
            high
                         Set_IOD_Data(C_PORT_IOD15);
                         Set_IOD_Data(C_PORT_IOD14);
                  }
            }
//======================== end of main.c =====================================//


//===================== start of IRQ.c ======================================//

         #include    "SPMC701FM0.H"
         #include    "unSPMACRO.H"

         extern    g_iFlag_IRQ7;

         void IRQ7(void) __attribute__ ((ISR))       ;
         void IRQ7(void)
        {
                if ((Get_INT_Status() & C_2HZINT)!=0)
               {
                     g_iFlag_IRQ7 ^= 0x01;
                     Set_INT_Status(C_4HZINT);           //clear 2Hz interrupt flag
               }
         }
//=====================    end of IRQ.c ====================================//
```

10 I/O Ports

10.1 Introduction

General purpose I/O can be considered the simplest peripheral. They allow the SPMC70xxx to monitor and control other devices. To add flexibility and functionality to a device, some parts of I/O pins are multiplexed with an alternative function. These functions can be switched through appropriate registers. The voltage level of I/O can be set from IC VDD to 5V. Users must connect the power to VDDA, VDDB, VDDC, and VDDD. For most general ports, these I/O structure contain five parts: data, buffer, direction, attribution and latch registers. The naming rules of register are listed as follows:

Data Register	→	P_IO<x>_Data
Buffer Register	→	P_IO<x>_Buffer
Direction Register	→	P_IO<x>_Dir
Attribution Register	→	P_IO<x>_Attrib
Latch Register	→	P_IO<x>_Latch

Figure 8-1 shows the IO a sketch map. Figure 8-2 shows a typical I/O port. This does not add the multiple functions onto I/O pin. In addition, the table 8-1 is a summary of I/O setup configuration.

Table 8-1 I/O configuration

Direction	Attribution	Data	Function	Wakeup	Description
0	0	0	Pull Low*	Yes	Input with pull low
0	0	1	Pull High	Yes	Input with pull high
0	1	0	Float	Yes	Input with float
0	1	1	Float	No	Input with float
1	0	0	( ) Inverted	No	Output with data inverted (write “0” to the Data Port and will output “1” to the I/O pad)
1	0	1	Inverted	No	Output with data inverted (write “1” to the Data Port and will output “0” to the I/O pad)
1	1	0	Not Inverted	No	Output with buffer (data not inverted)
1	1	1	Not Inverted	No	Output with buffer (data not inverted)

*Default: Pull Low

The Direction, Attribution and Data represent three ports. Each corresponding bit in these ports should be given a value. The setting rules are as follows:

a. The direction setting determines whether this pin is an input or output.

b. The attribute setting gives a feature to the pin, float / pull for input, not inverted/ inverted for output.

c. The data setting affects the initial content of the pin. For inputs, it also determines the pull high or pull low setting. For example, suppose Port A.0 is used as input with pull low. The bit0 in Port A's Direction, Attribution and Data control ports should be given all 0s. If Port A.1 is intended to being input with float and wakeup function, the bit1 of the Port A's Direction, Attribution and Data registers should be given a value of "010" correspondingly. Note that each port characterizes 16 Direction, Attribution and Data bits. Users should pay extra attention while configuring I/O.

d. The I/O structure is able to change attribute easily. For example, the float (011) can be changed to output high (111) by only modifying the direction bit from "0" to "1".

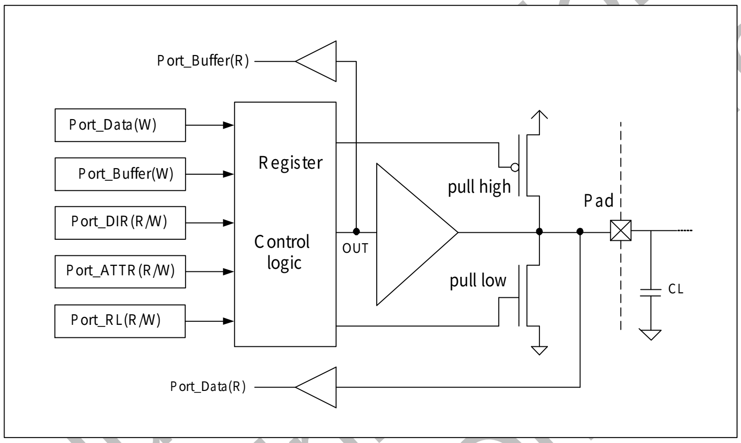


Figure 8-1 IO diagram

The most I/O ports support wakeup function. The wakeup function is available only when the I/O is configured as input pull high, low or input with float. Users can develop a low-power-consumed application by using sleep or wait function and taking the advantage of wake-up function. To reach this function, user must latch the I/O status by reading latch register. In addition, when I/O changed in sleep mode, the system will detect the I/O change and wakeup from sleep mode.

When peripheral functions are multiplexes onto general I/O pins, the functionality of I/O pins may change to accommodate the requirements of the peripheral module. Examples of this are the Analog-to-Digital(A/D) converter and SIO series interface. When the I/O pin is configured as peripheral function, the I/O direction and attribution register will be set as designate status.

For example, the SIO use PortA[15:14] two pins as SCL and SDA pin. When the SIO peripheral function is chosen, the two pins are configured as the required attributes: open drain input and output status.

10.2Port A

Port A contains five registers to be used for controlling the function of the general I/O. The following tables are the description of the registers.

10.2.1 P_IOA_Data ($7000H)

Write data into the data register and read data from the I/O pad. Writing data into $7000H will be the same as writing into $7001H.

B15	B14	B13	B12	B11	B10	B9	B8
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
P_IOA_Data							

B7	B6	B5	B4	B3	B2	B1	B0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
P_IOA_Data							

10.2.2 P_IOA_Buffer ($7001H)

Reading means to read data from data register. Write data into $7001H will be the same as writing into $7000H.

B15	B14	B13	B12	B11	B10	B9	B8
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
P_IOA_Buffer							

B7	B6	B5	B4	B3	B2	B1	B0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
P_IOA_Buffer							

Note:

Although the data is written into the same data register by writing P_IOA_Data(W) and P_IOA_Buffer(W), it is read from different locations by P_IOA_Buffer(R) and P_IOA_Data(R). The reading of P_IOA_Data (R)($7000H) and P_IOA_Buffer (R)($7001H) is through different physical path. The instruction of Reading P_IOA_Data (R)($7000H) reads data from I/O pad. The instruction of reading P_IOA_Buffer (R)($7001H) reads data from I/O buffer. Please refer to the I/O block diagram.

10.2.3 P_IOA_Dir ($7002H)

Read/Write direction-vector from/into the Direction Register.

B15	B14	B13	B12	B11	B10	B9	B8
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
P_IOA_Dir							

B7	B6	B5	B4	B3	B2	B1	B0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

0	0	0	0	0	0	0	0
P_IOA_Dir							

10.2.4 P_IOA_Attrib ($7003H)

Read/Write attribute vector from/into the Attribute Register.

B15	B14	B13	B12	B11	B10	B9	B8
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
P_IOA_Attrib							

B7	B6	B5	B4	B3	B2	B1	B0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
P_IOA_Attrib							

10.2.5 P_IOA_Latch ($7004H)

Read this port to latch data on the I/O PortA for key change wakeup before getting into sleep mode .

B15	B14	B13	B12	B11	B10	B9	B8
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0
P_IOA_Latch							

B7	B6	B5	B4	B3	B2	B1	B0
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0
P_IOA_Latch							

Note:

In addition, IOA[15:0] are key change wake-up sources. To activate the key change wake-up function, the P_IOA_Latch(R)($7004H) must be read to latch the I/O state of PortA and key change wake-up function must be enabled before entering into standby mode. Wake-up is triggered when the I/O state of PortA is different from the state at the time latched.

10.2.6 Special Functions of PortA

PortA shares with the following special functions for normal operation:

1. Parallel Communication Interface	→	Port A [11:0]
2. UART Serial Interface	→	Port A[13:12]
3. SIO interface	→	Port A[15:14]

There is an enable control signal for every special function to configure designate pins as the required attribution. The control signals and description of the special function are shown in table 8-2.

Table 8-2 special function

	SFR Pin	Type	PHB	PL	OEB	Description
IOA15	SCL($\overline{SS}$)	I/O(I)	1	0	SIOEN (SPIEN)	SIOEN=1, SIO serial clock Input in Slave mode and Output in Master mode, open drain. (SPIEN=1, SPI $\overline{SS}$ input)
IOA14	SDA	I/O	1	0	SIOEN	SIOEN=1, SIO serial data Input/output, Open drain.
IOA13	TXD	O	1	0	TXDOEB	UART transmission data output when UARTEN=1, Hi-Z when TXDOEB=1
IOA12	RXD	I	1	0	1	UART receive data input when UARTEN=1
IOA11	PCIRDB	I/O	1	0	PCIM0B	PCI RDB input/output when PCIEN=1 Output in Master mode and Input in Slave mode
IOA10	PCIWEB/ RST_N	I/O	1	0	PCIM0B/ RSTENB	PCI WEB input/output when PCIEN=1 Output in Master mode and Input in Slave mode System reset output when RSTEN=1
IOA9	PCIA0	I/O	1	0	PCIM0B	PCI A0 input/output when PCIEN=1 Output in Master mode and Input in Slave mode
IOA8	PCICSB	I/O	1	0	PCIM0B	PCI CSB input/output when PCIEN=1 Output in Master mode and Input in Slave mode
IOA7	PCID7	I/O	1	0	PDOEB	PCI data bit 7 when PCIEN=1
IOA6	PCID6	I/O	1	0	PDOEB	PCI data bit 6 when PCIEN=1
IOA5	PCID5	I/O	1	0	PDOEB	PCI data bit 5 when PCIEN=1
IOA4	PCID4	I/O	1	0	PDOEB	PCI data bit 4 when PCIEN=1
IOA3	PCID3	I/O	1	0	PDOEB	PCI data bit 3 when PCIEN=1
IOA2	PCID2	I/O	1	0	PDOEB	PCI data bit 2 when PCIEN=1
IOA1	PCID1	I/O	1	0	PDOEB	PCI data bit 1 when PCIEN=1
IOA0	PCID0	I/O	1	0	PDOEB	PCI data bit 0 when PCIEN=1

10.3 Port B

Port B contains five registers to be used for controlling the function of the general I/O. The following tables are the description of the registers.

10.3.1 P_IOB_Data ($7008H)

Write data into data register and read from I/O pad. Writing data into $7008H will be the same as writing into $7009H.

B15	B14	B13	B12	B11	B10	B9	B8
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
P_IOB_Data							

B7	B6	B5	B4	B3	B2	B1	B0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
P_IOB_Data							

10.3.2 P_IOB_Buffer ($7009H)

Write data into the data register and read data from the I/O buffer. Writing data into $7009H will be the same as writing into $7008H.

B15	B14	B13	B12	B11	B10	B9	B8
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
P_IOB_Buffer							

B7	B6	B5	B4	B3	B2	B1	B0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
P_IOB_Buffer							

Note: The reading of P_IOB_Data (R)($7008H) and P_IOB_Buffer (R)($7009H) is through different physical path. The data is from I/O pad by reading P_IOB_Data (R)($7008H). The data is form I/O buffer by reading P_IOB_Buffer (R)($7009H).

10.3.3 P_IOB_Dir ($700AH)

Read/Write direction vectors from/into the direction register.

B15	B14	B13	B12	B11	B10	B9	B8
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
P_IOB_Dir							

B7	B6	B5	B4	B3	B2	B1	B0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
P_IOB_Dir							

10.3.4 P_IOB_Attrib ($700BH)

Read/Write attribute vector from/into the Attribute Register.

B15	B14	B13	B12	B11	B10	B9	B8
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
P_IOB_Attrib							

B7	B6	B5	B4	B3	B2	B1	B0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
P_IOB_Attrib							

10.3.5 P_IOB_Latch ($700CH)

Read this port to latch data on the I/O Port B for key change wakeup before getting into sleep mode.

B15	B14	B13	B12	B11	B10	B9	B8
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0
P_IOB_Latch							

B7	B6	B5	B4	B3	B2	B1	B0
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0
P_IOB_Latch							

10.3.6 Special Function of PortB

Port B shares with the following special functions:

1. SPI pins	→	Port B [15:13]
2. Capture/Compare/PWM pins	→	Port B [12:11]
3. External input (EXT1, EXT2)	→	Port B [10:9]
4. ADC interface	→	Port B [8:0]

The control signals and description of the special function pins are shown in table 8-3.

Table 8-3

	SFR Pin	Type	PHB	PL	OEB	Description
IOB15	SCK	I/O	1	0	SCKOEB	SPIEN=1, Input in Slave mode and Output in Master mode
IOB14	SDI	I	1	0	1	SPIEN=1
IOB13	SDO	O	1	0	0	SPIEN=1,output in master and slave mode
IOB12	CCP1	I/O	1	0	CCP1OEB	Timer1 CCP mode enable Input in Capture mode Output in Compare mode Output in PWM mode
IOB11	CCP0	I/O	1	0	CCP0OEB	Timer0 CCP mode enable Input in Capture mode Output in Compare mode Output in PWM mode
IOB10	EXT2	I	User	User	1	
IOB9	EXT1	I	User	User	1	
IOB8	ADCETRG	I	User	User	1	Analog-to-digital converter external trigger to start a sample conversion
IOB7	ADCCH7	I	User	User	User	Analog input of ADC channel 7
IOB6	ADCCH6	I	User	User	User	Analog input of ADC channel 6
IOB5	ADCCH5	I	User	User	User	Analog input of ADC channel 5
IOB4	ADCCH4	I	User	User	User	Analog input of ADC channel 4
IOB3	ADCCH3	I	User	User	User	Analog input of ADC channel 3
IOB2	ADCCH2	I	User	User	User	Analog input of ADC channel 2

	SFR Pin	Type	PHB	PL	OEB	Description
IOB1	ADCCH1	I	User	User	User	Analog input of ADC channel 1
IOB0	ADCCH0	I	User	User	User	Analog input of ADC channel 0

10.4Port C

Port C contains five registers to be used for controlling the function of the general I/O. The following tables are the description of the registers.

10.4.1 IOC_Data ($7010H)

Write data into data register and read from I/O pad. Writing data into $7010H will be the same as writing into $7011H.

B15	B14	B13	B12	B11	B10	B9	B8
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
P_IOC_Data							

B7	B6	B5	B4	B3	B2	B1	B0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
P_IOC_Data							

10.4.2 P_IOC_Buffer ($7011H)

Write data into the data register and read data from the I/O buffer. Writing data into $7011H will be the same as writing into $7010H.

B15	B14	B13	B12	B11	B10	B9	B8
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
P_IOC_Buffer							

B7	B6	B5	B4	B3	B2	B1	B0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
P_IOC_Buffer							

Note: The reading of P_IOC_Data (R)($7010H) and P_IOC_Buffer (R)($7011H) is through different physical path. The data is from I/O pad by reading P_IOC_Data (R)($7010H). The data is form I/O buffer by reading P_IOB_Buffer (R)($7011H).

10.4.3 P_IOC_Dir ($7012H)

Read/Write direction vectors from/into the direction register.

B15	B14	B13	B12	B11	B10	B9	B8
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
P_IOC_Dir							

B7	B6	B5	B4	B3	B2	B1	B0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
P_IOC_Dir							

10.4.4 P_IOC_Attrib ($7013H)

Read/Write attribute vector from/into the Attribute Register.

B15	B14	B13	B12	B11	B10	B9	B8
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
P_IOC_Attrib							

B7	B6	B5	B4	B3	B2	B1	B0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
P_IOC_Attrib							

10.4.5 P_IOC_Latch ($7014H)

Read this port to latch data on the I/O Port C for key change wakeup before getting into sleep mode.

B15	B14	B13	B12	B11	B10	B9	B8
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0
P_IOC_Latch							

B7	B6	B5	B4	B3	B2	B1	B0
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0
P_IOC_Latch							

10.5Port D

Port D contains five registers to be used for controlling the function of the general I/O. The following tables are the description of the registers.

10.5.1 P_IOD_Data ($7018H)

Write data into data register and read from I/O pad. Write data into $7018H will be the same as writing into $7019H.

B15	B14	B13	B12	B11	B10	B9	B8
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
P_IOD_Data							

B7	B6	B5	B4	B3	B2	B1	B0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
P_IOD_Data							

10.5.2 P_IOD_Buffer ($7019H)

Write data into the data register and read data from the I/O buffer. Writing data into $7019H will be the same as writing into $7018H.

B15	B14	B13	B12	B11	B10	B9	B8
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
P_IOD_Buffer							

B7	B6	B5	B4	B3	B2	B1	B0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
P_IOD_Buffer							

Note: The reading of P_IOD_Data (R)($7018H) and P_IOD_Buffer (R)($7019H) is through different physical path. The data is from I/O pad by reading P_IOD_Data (R)($7018H). The data is form I/O buffer by reading P_IOD_Buffer (R)($7019H).

10.5.3 P_IOD_Dir ($701AH)

Read/Write direction vectors from/into the direction register.

B15	B14	B13	B12	B11	B10	B9	B8
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
P_IOD_Dir							

B7	B6	B5	B4	B3	B2	B1	B0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
P_IOD_Dir							

10.5.4 P_IOD_Attrib ($701BH)

Read/Write attribute vector from/into the Attribute Register.

B15	B14	B13	B12	B11	B10	B9	B8
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
P_IOD_Attrib							

B7	B6	B5	B4	B3	B2	B1	B0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
P_IOD_Attrib							

10.5.5 P_IOD_Latch ($701CH)

Read this port to latch data on the I/O Port D for key change wakeup before getting into sleep mode.

B15	B14	B13	B12	B11	B10	B9	B8
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0
P_IOD_Latch							

B7	B6	B5	B4	B3	B2	B1	B0
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0
P_IOD_Latch							

10.6I/O Programming Considerations

When using GPIO, design considerations need to be taken into account to ensure that the operation is as intended.? **Example 8-1**? Set IOA[3:0] as Pull Low, IOA[7:4] as Pull High, IOA[11:8] as Output Low, and IOA[15:12] as Output High.

A) Use assemble language

```
R1 = 0xFF00;
[P_IOA_Dir] = R1;
R1 = 0xFF00;
[P_IOA_Attrib] = R1;
R1 = 0xF0F0;
[P_IOA_Data] = R1;
```

B) Use C language

```
Set_IOD_Dir(C_PORT_IOD0);
Set_IOD_Attrib(C_PORT_IOD0);
Set_IOD_Data(C_PORT_IOD0);
```

? **Example 8-2**? There are two ways to read data from IOA. Reading from P_IOA_Data is to read the signal existing on the I/O pin; however, reading from P_IOA_Buffer will acquire the data on the buffer registers.

A) Use assemble language

```
R1 = [P_IOA_Data] ;     // Read data from IOA pad
R1 = [P_IOA_Buffer];    // Read data from IOA buffer
```

B) Use C language

```
unsigned int iData;
iData = Get_IOA_Data();     // Read data from IOA pad
iData = Get_IOA_Buffer();   // Read data from IOA buffer
```

? **Example 8-3**? There are two ways to write data to IOA. Writing P_IOA_Data is the same as writing P_IOA_Buffer.

A) Use assemble language

```
R1 = 0x5555;
[P_IOA_Data] = R1;
[P_IOA_Buffer] = R1;
```

B) Use C language

```
Set_IOA_Data(0x5555);
Set_IOA_Buffer(0x5555);
```

10.7I/O Initialization

When reset conditions occur, which are POR, LVR, External Reset, Parallel Reset and ICE Reset, the IO Port will be set in initial state, shown in table 8-4. However, when the watchdog reset and illegal reset occur, the IO port will keep the original status.

Table 8-4 Reset status

Port Name	Register Name	Address	Initial Value
Port A	P_IOA_Data	$7000H	0000,0000,0000,0000B
	P_IOA_Buffer	$7001H	0000,0000,0000,0000B
	P_IOA_Dir	$7002H	0000,0000,0000,0000B
	P_IOA_Attrib	$7003H	0000,0000,0000,0000B
	P_IOA_Latch	$7004H	0000,0000,0000,0000B
Port B	P_IOB_Data	$7008H	0000,0000,0000,0000B
	P_IOB_Buffer	$7009H	0000,0000,0000,0000B
	P_IOB_Dir	$700AH	0000,0000,0000,0000B
	P_IOB_Attrib	$700BH	0000,0000,0000,0000B
	P_IOB_Latch	$700CH	0000,0000,0000,0000B
Port C	P_IOC_Data	$7010H	0000,0000,0000,0000B
	P_IOC_Buffer	$7011H	0000,0000,0000,0000B
	P_IOC_Dir	$7012H	0000,0000,0000,0000B
	P_IOC_Attrib	$7013H	0000,0000,0000,0000B
	P_IOC_Latch	$7014H	0000,0000,0000,0000B
Port D	P_IOD_Data	$7018H	0000,0000,0000,0000B
	P_IOD_Buffer	$7019H	0000,0000,0000,0000B
	P_IOD_Dir	$701AH	0000,0000,0000,0000B
	P_IOD_Attrib	$701BH	0000,0000,0000,0000B
	P_IOD_Latch	$701CH	0000,0000,0000,0000B

11 Low Voltage Reset (LVR) and Low Voltage Detection (LVD)

11.1 Introduction

The LVR/LVD module of SPMC70xxx provides a reset and power level detection signals to reset system or indicate the power level status. Figure 9-1 shows the LVR and LVD function block. The low voltage detection level is programmable by controlling the low voltage detection level selection (LVDS) bits. Two registers can control the LVR and LVD function. P_System_Option B3, B2 must be turned on, and set P_LVDLVR_Ctrl B5, B3 to turned on. Then, LVD and LVR will be activated. However, the LVR function is disabled in test mode. The LVDINT is used for checking if the system gets into low voltage level. The status can be checked by interrupt function or polling the appropriate register bit.

The LVR and LVD circuits also provide a 2V stability voltage, which is the ADC reference voltage. Programming the register bit, VREFEN, can turn on this function.

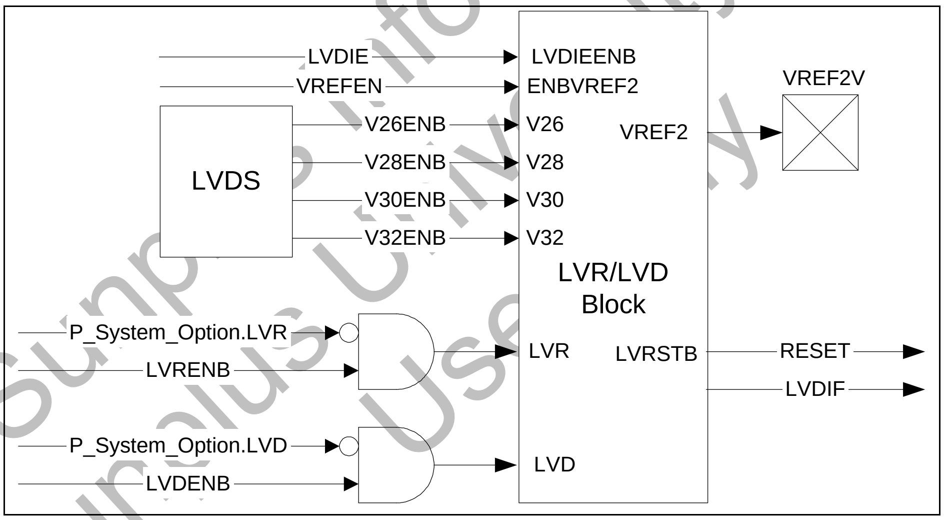


Figure 9-1 LVR and LVD function block

11.2 Control Register

Three registers can control the LVD and LVR function. The P_System_Option is set by ICE or writer function; the others registers, P_LVDLVR_Ctrl and P_LVDLVR_Status, can be programmed by users. Please see the 9.4 design tips.

11.2.1 P_LVDLVR_Ctrl ($7066): LVD/LVR control register

B15	B14	B13	B12	B11	B10	B9	B8
-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-

B7	B6	B5	B4	B3	B2	B1	B0
-	-	R/W	R/W	R/W	R/W	R/W	R/W
-	-	0	0	0	0	0	0
-	-	LVDENB	VREFENB	LVRENB	LVDIEN	LVDLS	

Bit 15:6 Reserved

Bit 5 LVDENB: Software LVD power control

1= Disable LVD

0= Enable/Disable LVD depend on P_System_Option

Bit 4 VREFENB: Reference voltage enable, enable to build reference voltage

0= Enable

1= Disable

Bit 3 LVRENB: Software LVR control

1= Disable

0= Enable/Disable LVR depend on P_System_Option

Bit 2 LVDIEN: Low-voltage detection interrupt enable

1= Enable

0= Disable

Bit 1:0 LVDLS: Low-voltage detection level selection

00= 2.6V

01= 2.8V

10= 3.0V

11= 3.2V

11.2.2 P_LVD_Status ($7067): LVD status register

B15	B14	B13	B12	B11	B10	B9	B8
-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-

B7	B6	B5	B4	B3	B2	B1	B0
-	-	-	-	-	-	-	R/W
-	-	-	-	-	-	-	0
-	-	-	-	-	-	-	LVDIF

Bit 15:1 Reserved

Bit 0 LVDIF: Low-voltage detection interrupt flag

1= LVD interrupt occurs

0= No LVD interrupt

This bit is set when VDD drops below the programmed LVD voltage level and is cleared by software writing this bit '1'.

11.2.3 P_System_Option ($8000): System Option Register

This address places at information block, not in the general flash address. User must use ICE function or writer to set this port. For detailed description, please see section 6.

B15	B14	B13	B12	B11	B10	B9	B8
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	1	0	1	0	1	0	1
Verification Pattern							

B7	B6	B5	B4	B3	B2	B1	B0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	1	0	1	0	1	1	1
Verification Pattern			Security	LVD	LVR	WDG	CLK Source

Bit 15:5 Verification Pattern

ICE or Writer will check this area and write 01010101010 to this area

Bit 4 Security: select flash enable read or write

0: protect

1: not protect, can be readable or write-able

Bit 3 LVD: enable low voltage detection function

0: enable

1: disable

Bit2 LVR: enable low voltage reset function

0: disable

1: enable

Bit1 WDG: enable watchdog function

0: disable

1: enable

Bit0 CLK Source: Clock Source Selection

0: R oscillator

1: crystal oscillator

Note: The low voltage reset and low voltage detection can be enabled or disabled by ICE or writer function.

11.3LVD Interrupt

SPMC70xxx provides the low voltage detection interrupt to let system can deal with the LVD signal immediately. This interrupt have two paths, FIQ and IRQ5 (see section 7). User can check LVDIF of P_LVD_Status or LVDINT of P_INT_Status to monitor LVD signal. If user does not use this function, user must check this flag to acquire the status of these flag. When the VDD level is lower than the designate level, LVDIF and LVDINT (Low voltage detect interrupt flag) is set and the system enters into the interrupt service routing. It is suggested that checking LVDIF or LVDINT flag to assure the low voltage status and setting the alarm function such as warning message, warning sign or sound. Writing 1 to the B0 of P_LVDLVR_Status can clear the LVDINT and LVDIF flags. If users turn on the LVR (low voltage reset) function, the low-voltage-reset signal is issued when the VDD level is lower than 2.6V for three consecutive 3MHz clock cycle. In case of voltage rising back to above 2.6, the reset duration will increase with an 80us extension time. As LVR occur and come into reset status, system will be reset and clear the LVD interrupt flag.

11.4Design Tips

11.4.1 LVR

There are 2 steps to set LVR function:

1. Set P_System_Option LVR bit: Set by ICE or writer function, used only to turn LVR on or off.
2. Set P_LVDLVR_Ctrl LVRENB bit: 0: enable LVR 1:Disable LVR

```
R1 = [P_LVDLVR_Ctrl];
R1 &= (0x0008 ^ 0xFFFF);
[P_LVDLVR_CTRL] = R1;
```

If user has set LVR function, system will deal the reset command when the power drops below 2.6V for four 3MHz clock cycles, see figure 9-2. The P_Reset_Status LVRST bit will be set for examination.

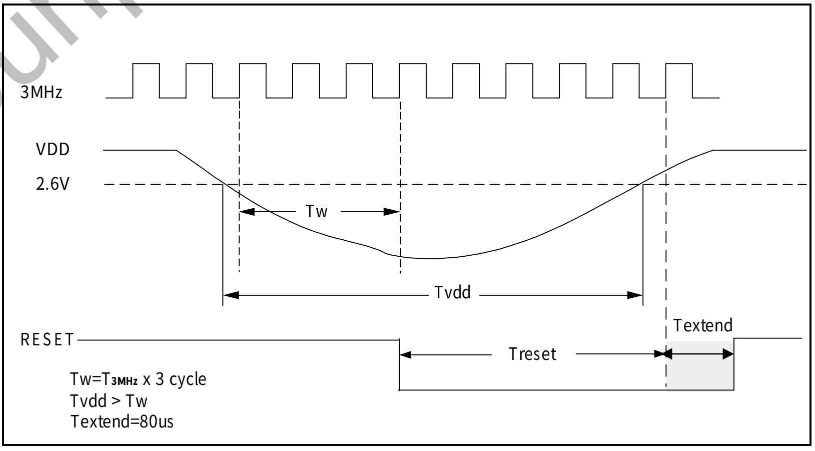


Figure 9-2 LVR function

11.4.2 LVD

There are 5 steps to set LVD function.

1. Set P_System_Option LVD bit: Set by ICE or writer function, used only to turn LVD on or off.
2. Set P_LVDLVR_Ctrl LVDENB bit: 0: enable LVD 1:Disable LVD
3. Set P_LVDLVR_Ctrl LVDLS bit: 00: 2.6V 01: 2.8V 10: 3.0V 11: 3.2V
4. Set P_LVDLVR_Ctrl LVDIEN bit to enable LVD interrupt

```
R1 = [P_LVDLVR_Ctrl]
R1 &= ( 0x0008 ^ 0xFFFF)        //enable LVD function
R1 |= 0x0001                    //Set 2.8V as low voltage detect level
R1 |= 0x0004                    //enable LVD interrupt
[P_LVDLVR_CTRL] = R1
```

5. Check LVD flag in interrupt or polling in the main function.

```
        R1 = [P_LVDLVR_Status]
        JZ   L_NoLVDDectect
        ……
  L_NoLVDDectect:
        ……
```

Note:

1. If LVR is set and LVD level is set as 2.6V, the system only operates the reset command and LVDIF is erased.
2. If system clock works at 48 MHz, the power must be set above 3.0V to prevent the unstable situation.

【Example 9-1】 Set the Low-voltage level as 3.0V . If the voltage level is lower than 3.0V, the system will produce a interrupt then invert the IOD0 LED. In addition, IOD1 LED glints to indicate the program rightly.

```
.INCLUDE SPMC701FM0.INC

.CODE
.PUBLIC _main
_main:
     R1 = 0x0003;
     [P_IOD_Dir] =R1;
     [P_IOD_Attrib] = R1;
      R1 = 0;
     [P_IOD_Data] = R1;
```

```
        R1 = 0x000E;        //Enable LVD,Enable interrupt,LVD Level 3.0V
        [P_LVDLVR_Ctrl] = R1;
        IRQ ON;

L_MainLoop:
       R1 = [P_IOD_Buffer]
        R1 ^= 0x2;
        [P_IOD_Buffer] = R1;
        call F_Delay;
       JMP L_MainLoop;

F_Delay:
        R2 = 0x5;
?L_Delay1:
        R1=0xFFFF;
?L_Delay2:
        R1-=1;
        JNZ ?L_Delay2
        R2 -= 1;
        JNZ ?L_Delay1
        RETF


.TEXT
.PUBLIC _IRQ5
_IRQ5:
        PUSH R1,R5 TO [SP]
        R1 = [P_IOD_Buffer]
        R1 ^= 0x1;
        [P_IOD_Buffer] = R1;

        R1= 0x01;                    //clear interrupt flag
        [P_LVD_Status] = R1;

        POP R1,R5 FROM [SP]
        RETI
```

12 Power Saving modes and wakeup

12.1 Introduction

The SPMC70xxx provides three modes for power saving, which are listed as follows:

1. halt mode: Only CPU off
2. sleep mode: Only 32768Hz on
3. standby mode: all clock sources are off

The halt mode only turns off the CPU and it halts to execute instruction and wait wakeup command. The sleep mode turns off all function and system clocks, except 32.768KHz and real time timer for wakeup. The standby mode will turn off all function and system clocks to consume the least current. All these three modes will be awaken by key-press if the key wakeup function is enabled. The table 10-1 shows the relationship of mode vs. operation. The table 10-2 shows the wakeup source of each mode.

Table 10-1 the relationship of mode vs. operation

Clock Source	Halt	Sleep	Standby
OSC32ENB(32768Hz crystal)	ON	ON	OFF
PLLENB (PLL)	ON	OFF	OFF
RON(R-osc and 6M crystal)	ON	OFF	OFF
CPUPD (CPU power down)	OFF	OFF	OFF
LVDLVRENB(LVD/LVR)	Keep Previous State	Keep Previous State	OFF
ADCENB (ADC)	Keep Previous State	OFF	OFF
Working SRAM	ON	OFF	OFF
Flash ROM	ON	OFF	OFF
1.6MHz R-oscillator	ON	OFF	OFF

Table 10-2 wakeup source

Wake-up source	HALT mode	SLEEP mode	STDBY mode
KCAWE	v	v	v
KCBWE	v	v	v
KCCWE	v	v	v
KCDWE	v	v	v
UARTIWE	v		
SPIIWE	v		
SIOIWE	v		
PCIIWE	v	v	
TMR2WE	v	v	
TMR1WE	v	v	
TMR0WE	v	v	
TMB2WE	v	v	
TMB1WE	v	v	
4HZWE	v	v	
2HZWE	v	v	

12.2 Control Register

12.2.1 P_Wakeup_Source ($706a)

B15	B14	B13	B12	B11	B10	B9	B8
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
KCAWEN	KCBWEN	KCCWEN	KCDWEN	UARTIWEN	SPIIWEN	SIOIWEN	PCIIWEN

B7	B6	B5	B4	B3	B2	B1	B0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
TMR2WEN	TMR1WEN	TMR0WEN	LVDWEN	TMB2WEN	TMB1WEN	4HZWEN	2HZWEN

Bit 15 KCAWEN: Port A Key change wakeup enable

1= Enable

0= Disable

Bit 14 KCBWEN: Port B Key change wakeup enable

1= Enable

0= Disable

Bit 13 KCCWEN: Port C Key change wakeup enable

1= Enable

0= Disable

Bit 12 KCDWEN: Port D Key change wakeup enable

1= Enable

0= Disable

Bit 11 UARTIWEN: UART interrupt wakeup enable

1= Enable

0= Disable

Bit 10 SPIIWEN: SPI interrupt wakeup enable

1= Enable

0= Disable

Bit 9 SIOIWEN: SIO interrupt wakeup enable

1= Enable

0= Disable

Bit 8 PCIIWEN: PCI interrupt wakeup enable

1= Enable

0= Disable

Bit 7 TMR2WEN: Timer2 interrupt wakeup enable

1= Enable

0= Disable

Bit6 TMR1WEN: Timer1 interrupt wakeup enable

1= Enable

0= Disable

Bit 5 TMR0WEN: Timer0 interrupt wakeup enable

1= Enable

0= Disable

Timer0 overflows from FFFFh to 0000h.

Bit 4 LVDWEN: Low-Voltage Detect wakeup enable

1= Enable

0= Disable

Bit 3 TMB2WEN: Timebase 2 wakeup enable

1= Enable

0= Disable

Bit 2 TMB1WEN: Timebase 1 wakeup enable

1= Enable

0= Disable

Bit 1 4HZWEN: 4Hz wakeup enable

1= Enable

0= Disable

Bit 0 2HZWEN: 2Hz wakeup enable

1= Enable

0= Disable

Note:

If no wakeup source is enabled, the tree power saving command is not acceptable by hardware protection. According to three power saving modes, at least one of wakeup sources needs to be activated.

12.2.2 P_Halt_Enter ($706E)

B15	B14	B13	B12	B11	B10	B9	B8
W	W	W	W	W	W	W	W
0	1	0	1	0	1	0	1
HaltCMD							

B7	B6	B5	B4	B3	B2	B1	B0
W	W	W	W	W	W	W	W
0	1	0	1	0	1	0	1
HaltCMD							

1. Write $5555 to P_Halt_Enter ($706E) to enter halt mode.
2. CPUPD is change to "1" at the 3rd CPUCLK rising edge after $5555 is written into P_Halt_Enter.
3. If there is no appropriate wake-up source enabled, SPMC70xxx series body can't enter into halt mode.
4. In ICE mode, SPMC70xxx series body cannot enter into halt mode.

12.2.3 P_Sleep_Enter ($706B)

B15	B14	B13	B12	B11	B10	B9	B8
W	W	W	W	W	W	W	W
0	1	0	1	0	1	0	1
SleepCMD							

B7	B6	B5	B4	B3	B2	B1	B0
W	W	W	W	W	W	W	W
1	0	1	0	1	1	0	0
SleepCMD							

1. Write $55aa to P_Sleep_Enter ($706b) to enter sleep mode(only 32768Hz on).
2. CPUPD is change to 1 at the 3rd CPUCLK rising edge after $55aa is written into P_Sleep_Enter.
3. At the 4th CPUCLK rising edge, working SRAM, flash ROM and ADC are turned off.
4. At the 5th rising edge of the system clock, PLL, 6M crystal or R-oscillator and 1.6MHz R-oscillator is turned off.
5. If there is no appropriate wake-up source enabled, SPMC70xxx series body can't enter into sleep mode.
6. If PCI interrupt (except PCI wakeup command), Timer0, Timer1, Timer2 interrupt or low voltage detect interrupt is chosen to be wakeup source, the system clock must be changed as 32768Hz before entering into sleep mode.
7. In ICE mode, SPMC70xxx series body can't enter into sleep mode.

12.2.4 P_Stdby_Enter ($706D)

B15	B14	B13	B12	B11	B10	B9	B8
W	W	W	W	W	W	W	W
0	1	0	1	1	0	1	0
StdbyCMD							

B7	B6	B5	B4	B3	B2	B1	B0
W	W	W	W	W	W	W	W
0	1	0	1	1	0	1	0
StdbyCMD							

1. Write $5a5a to P_Stdby_Enter ($706d) to enter standby mode (all function off).
2. CPUPD is change to 1 at the 3rd CPUCLK rising edge after $5a5a is written into P_Stdby_Enter.
3. At the 4th CPUCLK rising edge, working SRAM, flash ROM , ADC and LVDLVR are turned off.
4. At the 5th CPUCLK rising edge, PLL, 6M crystal or R-oscillator, 1.6MHz R-oscillator and 32768Hz crystal is turned off.
5. If key change wake-up source is not enabled, SPMC70xxx series body cannot enter into standby mode.
6. In ICE mode, SPMC70xxx series body cannot enter into standby mode.

12.2.5 P_Wakeup_Status ($706F)

B15	B14	B13	B12	B11	B10	B9	B8

-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-

B7	B6	B5	B4	B3	B2	B1	B0
-	-	-	-	R/W	R/W	R/W	R/W
-	-	-	-	0	0	0	0
-	-	-	-	HALTSF	SLEEPSF	STDBYSF	WAKEUPSF

Bit 15:4	Reserved
Bit 3	HALTSF: Halt mode status flag 1= Halt mode occurs 0= No halt mode occurs
Bit 2	SLEEPSF: Sleep mode status flag 1= Sleep mode occurs 0= No sleep mode occurs
Bit 1	STDBYSF: Standby mode status flag 1= Standby mode occurs 0= No standby mode occurs
Bit 0	WAKEUPSF: Wakeup flag status flag 1= the flag is set when wakeup occurs The flag must be cleared by write “0x55aa” to this port.

12.3 Design Tips

When MCU enters into “Halt mode” or “Sleep mode” or “Standby mode”, the MCU will stop running and waiting for the wakeup source to be activated. Note that MCU halts and stops executing the instruction only. The clock and other peripheral functions will keep the original statues. SPMC70xxx provides a register to point out the wakeup statues. User can read P_Wakeup_Status ($706f) to acknowledge the wakeup status.

? **Example10-1**? The following program shows if program executes the I/O toggle instruction. The program will execute halt mode instruction correctly. If system does not go into halt mode, the program will execute the IO toggle function.

```
    .include   SPMC701FM0.inc
    .CODE
    .PUBLIC _main
    _main:
        R4 = 0x0000;             //Set IOA as key input(input with pull low) for wake-up source
         [P_IOA_Dir] = R4;
         [P_IOA_Attrib] = R4;
         [P_IOA_Buffer] = R4;
         R4 = 0xFFFF;                //Set indicate IO to check MCU isn't in Halt mode
         [P_IOD_Dir] = R4;
         [P_IOD_Attrib] = R4;
         R4     = 0x5500;
         [P_IOD_Buffer] = R4;

         R1 = C_KEYIEN;
         [P_INT_Ctrl] = R1;
         IRQ ON;

    L_LoopHaltMode:
         R1 = [P_IOA_Latch]

         R1 = C_KCAWEN;                      //Set wakeup source
         [P_Wakeup_Source] = R1;             // if not, MCU can't entry halt mode
         R1 = C_HaltCMD;                     //Halt mode instruction
         [P_Halt_Enter] = R1;

        //R1 = C_SleepCMD;                   //Sleep mode instruction
        //[P_Sleep_Enter] = R1;

         //R1 = C_StdbyCMD;                  //Standby mode instruction
         //[P_Stdby_Enter] = R1;

         R4 ^=   0x5555;
         [P_IOD_Buffer] = R4;                //Display LED to indicate wakup
         jmp    L_LoopHaltMode


  .TEXT
    .PUBLIC _IRQ5
    _IRQ5:
         PUSH R1,R5 TO [SP]
         R1 = [P_Wakeup_Status];           // check Wakeup Status
         R1 = 0x55AA;
         [P_Wakeup_Status] = R1;
```

```
R1 = [P_Wakeup_Status];          // check Wakeup Status

R1 = C_KEYINT;
[P_INT_Status] = R1;
POP R1,R5 FROM [SP]
RETI
```

13 Parallel Communication Interface -PCI

13.1 Introduction

The SPMC70xxx provides a parallel communication protocol to communicate with other MCUs. This parallel communication interface (PCI) provides half-duplex asynchronous parallel communication between a master device and a slave device. The PCI module supports both master mode and slave mode. The basic features are listed as follows:

1. Supports master/slave modes
2. Total of 11/12 pins needed for parallel interface (programmable RDB, WRB to be combined into RWB).
3. If RSTEN is enabled, the parallel interface can be configured as 11-pin mode only.
4. Dedicated hardware to generate interface signals for master mode.
5. Supports software reset
6. Supports software wake-up
7. Reflects slave sleeping status, data ready, data acknowledgment.

The relationship between master and slave are shown in the figure 11-1.

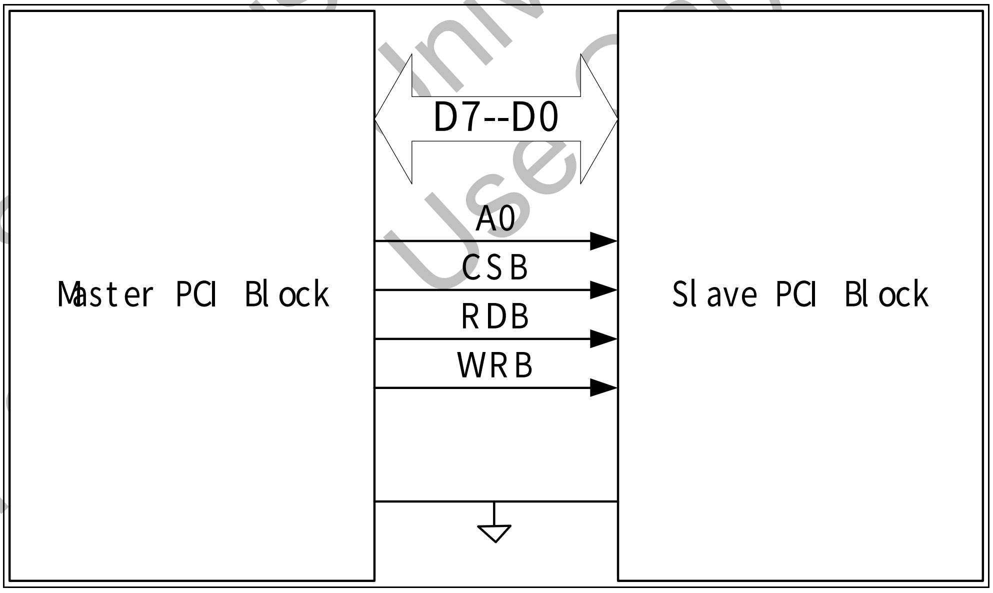


Figure 11-1 The relationship between master and slave

To be a master, the hardware generates the required interface signals to slave. The communication cases between a Master and a Slave include:

1. Master reads Slave's status to Master's input buffer
2. Master reads Slave's output buffer to Master's input buffer
3. Master sends command to Slave

4. Master sends data to Slave

The above procedures are triggered by firmware reading or writing ports. P_PCI_RxBuf and P_PCI_TxBuf with P_PCI_COMM port are defined for Master to generate the interface signals for reading Slave's status or output buffer and sending command or data to Slave.

In order to identify the received data byte in input buffer is from Slave's output buffer or status register, an PCIDAT bit in status register is defined.

A transmission transfer is issued only when a Master's output buffer is written into the byte (command or data depending on A0).

Parallel Interface Signals is described as follows:

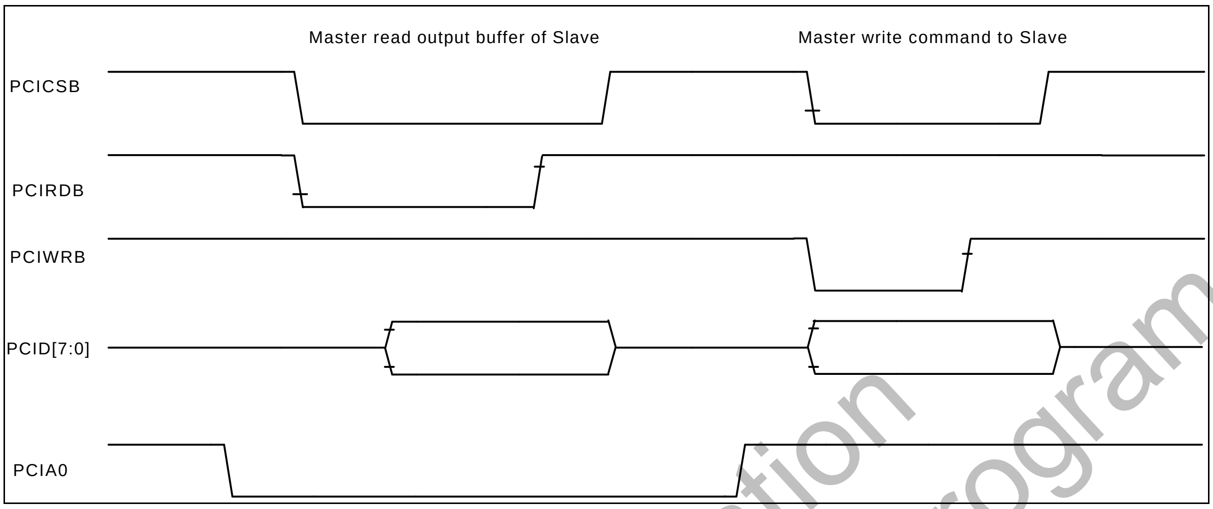
1. PCID7 – PCID0: parallel data bus (I/O).
2. PCICSB: chip select (Input for Slave and Output for Master), low active.
3. PCIRDB/PCIRWB: read control signal (Input for Slave and Output for Master), low Active. This pin can be configured as read/write control signal.
4. PCIWRB: write control signal (Input for Slave and Output for aster), low active. In case of RSTOEN = 1, this pin is configured as system reset output.
5. PCIA0: indicate that command/data which master write to slave, or status/data which master read from slave (Input for Slave and Output for Master).

Parallel port's pins are shared with the GPIO Port A shows in tale 11-1

Table 11-1 Parallel port's pins

Port A I/O	Parallel Port Pins
Port A11	PCIRDB
Port A10	PCIWRB
Port A9	PCIA0
Port A8	PCICSB
Port A[7:0]	PCID[7:0]

The PCI interface will generate the figure 11-2 timing automatic.


Master read output buffer of Slave
Master write command to Slave
PCICSB
PCIRDB
PCIWRB
PCID[7:0]
PCIA0

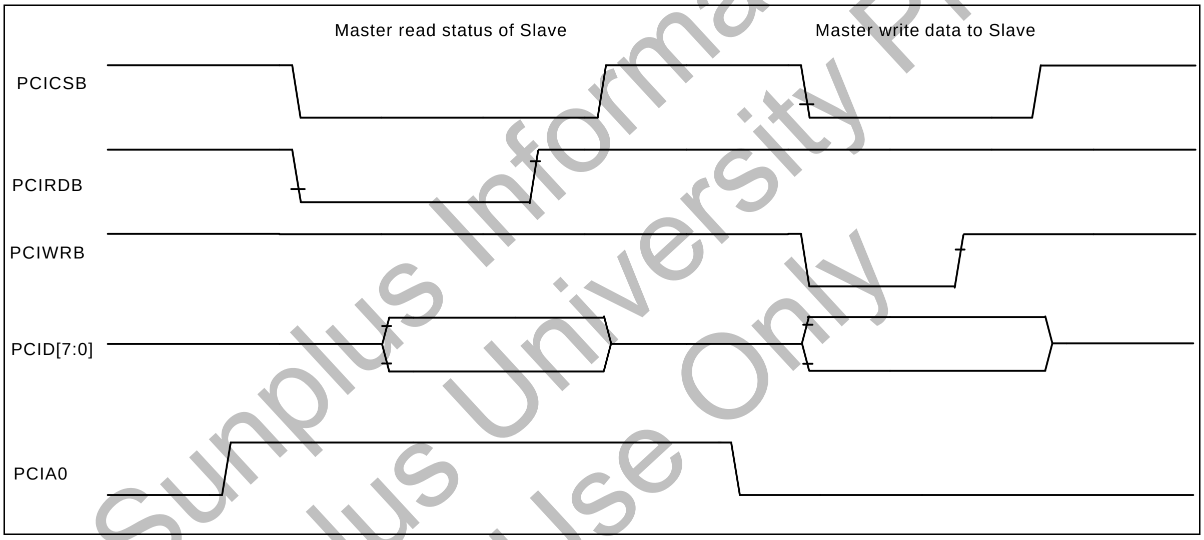


Figure 11-2

13.2 Control Register

13.2.1 P_PCI_Ctrl ($70C0): PCI Control Register

B15	B14	B13	B12	B11	B10	B9	B8
-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-

B7	B6	B5	B4	B3	B2	B1	B0
R/W	R/W	R/W	R/W	R/W	-	-	-
0	0	0	0	0	-	-	-
PCIEN	PCIPS	PCIMS	PCIIEN	RSTOEN	-	-	-

Bit 15:8	Reserved
Bit 7	PCIEN: PCI Enable
	1= Enable
	0= Disable
Bit6	PCIPS: PCI pin mode selection

1= 12-pin mode, PCIRDB for read control and PCIWRB for write control

0= 11-pin mode, PCIRDB for read/write control, PCIWRB for system reset output if RSTEN = 1

Bit 5 PCIMS: PCI mode selection

1= Master mode

0= Slave mode

Bit 4 PCIIEN: PCI Interrupt Enable

1= Enable

0= Disable

Bit 3 RSTOEN: System reset output enable, this configuration can be cleared automatically when power on reset occur

1= Enable

0= Disable

Bit 2:0 Reserved

13.2.2 P_PCI_Status ($70C1): PCI Status register

B15	B14	B13	B12	B11	B10	B9	B8
-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-

B7	B6	B5	B4	B3	B2	B1	B0
R	R	R	R/W	R	R	-	-
0	0	0	0	0	0	-	-
PCIRBF	PCITBF	PCIROR	PCIIF	PCIDAT	PWRSF	-	-

Bit 15~8 Reserved

Bit 7 PCIRBF: PCI receive buffer full, byte receiving complete

1= Input buffer full

0= Input buffer empty

This bit is set by hardware and cleared by CPU reading input buffer (P_PCI_RxBuf)

Bit 6 PCITBF: PCI transmission buffer full, data is being transmitted

1= Output buffer full

0= Output buffer empty

The bit is set by firmware writing P_PCI_TxBuf register and cleared by hardware when transmission is completed.

Bit 5 PCIROR: PCI receive over run error, a new byte received and PCIRBF=1

1= Overrun error occurs

0= No overrun errors

Bit 4 PCIIF: PCI interrupt flag

1= The flag is set when data is received in both modes or transmitted in slave mode

The flag must be cleared by firmware by writing this bit "1".

Bit 3 PCIDAT: Received data type, the previous received byte

For Master:

1:From Slave's status register

0:From Slave's output buffer

For Slave:

1:Command

0: Data

Bit 2 PWRSF: Power sleep flag

1= In Sleep mode

0= In normal operation mode

Bit 1:0 Reserved

13.2.3 P_PCI_RxBuf ($70C2): PCI receiving buffer

B15	B14	B13	B12	B11	B10	B9	B8
-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-

B7	B6	B5	B4	B3	B2	B1	B0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
PCIRXBUF							

For read:

Bit 15~8 reserved

Bit 7~0 PCIRXBUF: PCI input buffer register

For write:

Write any data into P_PCI_RxBuf register to do the master read status from slave.

13.2.4 P_PCI_TxBuf ($70C3)：PCI transmission buffer

B15	B14	B13	B12	B11	B10	B9	B8
-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-

B7	B6	B5	B4	B3	B2	B1	B0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
PCITXBUF							

For write:

Bit 15~8 reserved

Bit 7~0 PCITXBUF: PCI output buffer register

For read:

Read from P_PCI_TxBuf register to do the master read data from slave.

13.2.5 P_PCI_COMM ($70C4): PCI send command to slave port

Write any data into P_PCI_COMM register to do the master send command to slave.

B15	B14	B13	B12	B11	B10	B9	B8
W	W	W	W	W	W	W	W
0	0	0	0	0	0	0	0
-	-	-	-	-	-	-	-

B7	B6	B5	B4	B3	B2	B1	B0
W	W	W	W	W	W	W	W
0	0	0	0	0	0	0	0
PCICMD							

13.3 Operation

13.3.1 Command Description in slave mode

When the input buffer is full and the data type is command, slave will decode the content in P_PCI_RxBuf, and then execute command. In case of reset being decoded, the reset sate will be cleared by hardware after 8th CPU clock cycle. In case of wakeup being decoded, the wakeup state will be cleared through reading P_PCI_RxBuf by firmware.

PPIBUF[7:0]	Command Description
01010101	Reset by master
10101010	Wakeup by master
Others Command	Defined by firmware of user

13.3.2 Description of commands in master

Direction	Function	Command	Target register
Master ← Slave	Read Data	Read	P_PCI_TxBuf
Master → Slave	Send Command	Write	P_PCI_COMM
Master → Slave	Send Data	Write	P_PCI_TxBuf
Master ← Slave	Read Status	Write	P_PCI_TxBuf

13.3.3 Download (Master to Slave)/ Upload (Slave to Master) Procedures

A: Master reads slave's status:

1. Master's CPU (program) writes any data to **P_PCI_RxBuf**.
2. The above writing triggers the master command decoder to send the corresponding control signals to parallel port and read the slave's status into master's **P_PCI_RxBuf**.
3. Master state machine sets the input buffer status as full, that is, "1".
4. Master's CPU (program) polls(read) the **P_PCI_Status**'s input buffer(**PCIRBF**) status bit to be "1".

5. Master's CPU (program) reads the **P_PCI_RxBuf** data.
6. The above reading clears the input buffer status bit as "0".

B: Master reads slave's data:

1. Master reads slave's status and make sure that there is data in slave output buffer by checking the slave's output data buffer status.
2. Master CPU (program) reads P_PCI_TxBuf.
3. Triggers the master command decoder to generate the corresponding control signals to parallel port and read the slave's P_PCI_TxBuf data into master's P_PCI_RxBuf.
4. Master state machine sets the input buffer status as "1".
5. Master CPU reads P_PCI_Status to check the input buffer status bit.
6. Master's CPU (program) reads the P_PCI_RxBuf data.
7. The above reading clears the input buffer status bit.

C: Master sends data to slave:

1. Master's CPU writes P_PCI_TxBuf.
2. Triggers the master command decoder to generate the corresponding control signals to parallel port.
3. The data from CPU data bus send to slave's P_PCI_RxBuf.
4. Slave CPU reads P_PCI_Status to check the input buffer status bit.
5. Master's CPU (program) reads the P_PCI_RxBuf data.

D: Master sends command to slave:

1. Master's CPU writes P_PCI_COMM.
2. Triggers the master command decoder to generate the corresponding control signals to parallel port.
3. The command code from CPU data bus send to slave's P_PCI_RxBuf.
4. Slave CPU execute the command.

13.4 PCI Interrupt

The SPMC70xxx series provides the PCI interrupt to save CPU on polling PCIRBF or PCITBF. The interrupt vector of PCI can be two ways entrance. One is IRQ2 and the other is FIQ. The default PCI interrupt is IRQ2, and user can change it as FIQ by **P_INT_Priority Bit 9.** The PCI interrupt enabled by P_PCI_Ctrl b4 (PCIIEN). In master mode, when the P_PCI_RxBuf received data from slave, the master's P_PCI_Status PCIIF bit will be enabled and enter the interrupt. In slave mode, when the P_PCI_RxBuf received data from master or P_PCI_TxBuf send out to master, the slave's P_PCI_Status PCIIF bit will be enable and enter the interrupt. If user does not use the PCI interrupt to indicate the date in or out, user must poll the PCIRBF and PCITBF two bits in program.

13.5 Design Tips

【Example11-1】PCI Master Mode Test.This example is that the Master send 10 data to slave,and receive the data from slave (The slave sends the data back), user can chenk T_Buffer buffer.

A) Use assemble language

```
        .IRAM
        T_Buffer: .DW 10 DUP(0)

        .code
        .public _main
        _main:
          R1 = 0x00E0                 // Master,12 bit Mode
         [P_PCI_Ctrl] = R1
          R1 = 0x0055                 // Marster send Reset Command
          [P_PCI_COMM] = R1;

          R3 = 0x0;
      L_TransLoop:
          R1 = R3;
          [P_PCI_TxBuf] = R1          // master send data to salve
          R1 = [P_PCI_TxBuf];
      ?L_CheckSendDataEnd:
         R1 = [P_PCI_Status];
         R1 &= 0x0010;
         JZ   ?L_CheckSendDataEnd
          R1 = [P_PCI_Status];
         [P_PCI_Status] = R1
         R1 = [P_PCI_RxBuf];

        R2 = R3 + T_Buffer;
        [R2] = R1;                    // for check the recrice data

        R3 += 1;
        CMP R3,10;
        JBE   L_ TransLoop
    L_MainLoop:
        JMP L_MainLoop
```

B) Use C language

```
main()
{
        unsigned int uiData = 0;
        unsigned int uiRecData[10] = {0};
        Set_PCI_Ctrl(C_PCIEN+C_PCIPS_12+C_PCIMS_M);         //Master,12 bit Mode
        *P_PCI_COMM = C_PCIReset;                           //Marster send Reset Command

              while(uiData < 10)
              {
                     Set_PCI_TxBuf(uiData);
                     Get_PCI_TxBuf();
                     while(!(Get_PCI_Status() & C_PCIRBF)) ;
                     Set_PCI_Status(Get_PCI_Status());
                     uiRecData[uiData] = Get_PCI_RxBuf();
                     uiData ++;
                 }
                 while(1);
}
```

【Example11-2】PCI Slave Mode Test.This example is that the Slave receive data and then send the data back to marster.

A) Use assemble language

```
        R1 = 0x00C0                        //set as 12 bite salve
        [P_PCI_Ctrl] = R1

L_MainLoop:
L_CheckRxBuffer:
        R1 = [P_PCI_Status];
        R1 &= 0x0080;                 //check receive buffer if full
        JZ   L_CheckRxBuffer;
        R1 = [P_PCI_Status];
        [P_PCI_Status] = R1;
```

```
    R1 = [P_PCI_RxBuf];
    [P_PCI_TxBuf] = R1;
    JMP    L_MainLoop
```

B) Use C language

```
#include       "SPMC701FM0.H"
#include       "unSPMACRO.H"

main()
{
      Set_PCI_Ctrl(C_PCIEN+C_PCIPS_12+C_PCIMS_S);
       while(1)
      {
            while(!(Get_PCI_Status() & C_PCIRBF)) ;    //check receive buffer if full
            Set_PCI_Status(Get_PCI_Status());

            Set_PCI_TxBuf(Get_PCI_RxBuf());
       }
}
```

14 Timer 0

14.1 Introduction

SPMC70xxx series provides 16-bit timer/counter Timer0 and cooperating with another three 16-bit register, P_Timer0_Preload, P_Timer0_CCPR and P_Timer0_CCPR2, these timer and counters can support Capture, Compare and PWM functions. The CCP mode please refers to section 15.

The Timer0 module timer and counter has the following features:

1. Timer mode (internal clock) or counter mode (external clock)
2. Flexible clock source by logical AND two programmable clock sources
3. Interrupt on overflow from 0xFFFF to 0x0000
4. Edge select for external clock
5. Synchronized with system clock or not
6. When the timer is turned on, the content of P_Timer0_Preload is loaded into timer counter. This can be used to initialize the counter.
7. The source of Timer0 has two choice, One is source A, and the other is source B. Two sources combine by "AND" gate the following Figure 12-1 show the Timer0 block structure.

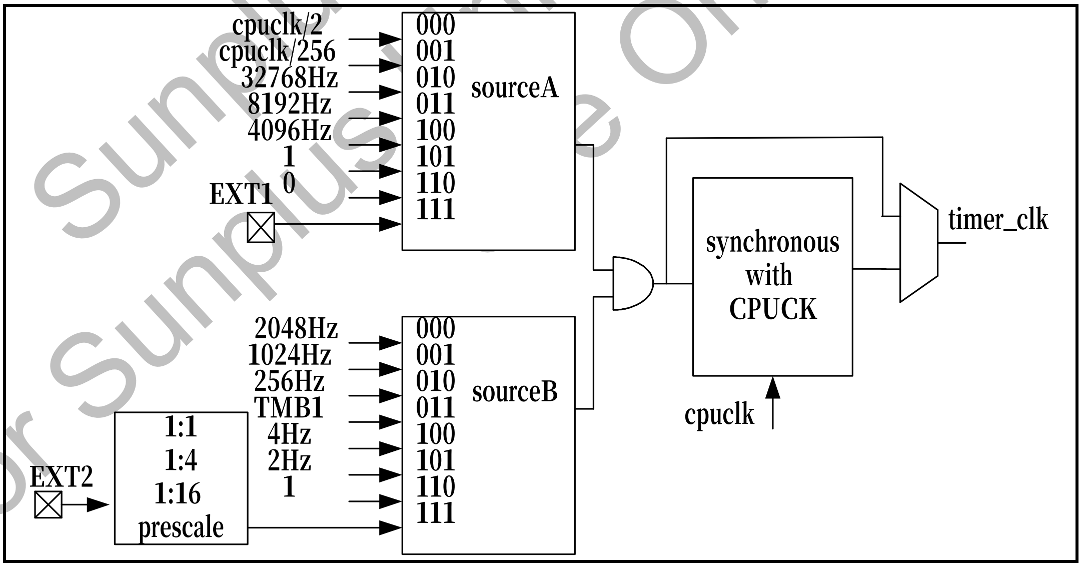


Figure 12-1 Timer 0 Block Structure

14.2 Control Register

14.2.1 P_Timer0_Ctrl ($7044): Timer0 control/status register

B15	B14	B13	B12	B11	B10	B9	B8
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
TMR0EN	TMR0IF	TMR0IEN	EXT2PS		CCP0MS		

B7	B6	B5	B4	B3	B2	B1	B0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
CCP0MS	SYNCLK	CLKBFS			CLKAFS		

Bit 15 TMR0EN: Timer0 Enable

1= Enable Timer0 Counter or CCP mode

0= Disable Timer0 Counter or CCP mode

Bit 14 TMR0IF: Timer0 Interrupt Flag

1= Timer0 Interrupt asserted

0= No Timer0 interrupt

As timer mode, this bit is set when Timer0 rollover from 0xFFFF to 0x0000,and is cleared by software writing this bit '1'. In addition, as CCP mode, this bit is set when CCP mode condition match.

Bit 13 TMR0IEN: Timer0 Interrupt Enable

1= Enable

0= Disable

This bit is used for enabling Timer0 interrupt. It's different from Bit15 TMR0EN. If only enable TMR0IEN else enable TMR0EN, the Timer0 interrupt can't be enabled.

Bit 12:11 EXT2PS: Counter mode prescale selection for EXT2

00= every falling

01= every rising

10= every 4 rising

11= every 16 rising

Bit 10:7 CCP0MS: Timer0 CCP0 Mode Selection

0000= Capture/Compare/PWM off

0100= Capture mode, every falling

0101= Capture mode, every rising

0110= Capture mode, every 4 rising

0111= Capture mode, every 16 rising

1000= Compare mode, set output (CCP0IF set)

1001= Compare mode, clear output (CCP0IF set)

101X= Compare mode, CCP0 unaffected (CCP0IF set)

1100= PWM mode, not return to one output (NRO)

1101= PWM mode, not return to zero output (NRZ)

1110= PWM mode, return to one output (RTO)

1111= PWM mode, return to zero output (RTZ)

Bit 6 SYNCLK: The clock source of TMR0 is synchronized by CPU clock

1= Synchronized

0= Not synchronized

Bit 5:3 CLKBFS: Clock Source B frequency selection

000= 2048Hz

001= 1024Hz

010= 256Hz

011= TMB1

100= 4Hz

101= 2Hz

110= 1

111= EXT2

Bit 2:0 CLKAFS: Clock Source A frequency selection

000= CPUCLK/2

001= CPUCLK/256

010= 32768Hz

011= 8192Hz

100= 4096Hz

101= 1

110= 0

111= EXT1

14.2.2 P_Timer0_Preload ($7045): Timer0 Preload Register

B15	B14	B13	B12	B11	B10	B9	B8
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
TMR0PR							

B7	B6	B5	B4	B3	B2	B1	B0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
TMR0PR							

14.2.3 P_Timer0_CCPR ($7046): Timer0 Capture/Compare/PWM Register

B15	B14	B13	B12	B11	B10	B9	B8

R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
TMR0CCPR							

B7	B6	B5	B4	B3	B2	B1	B0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
TMR0CCPR							

14.2.4 P_Timer0_CCPR2 ($7047): Timer0 Capture/Compare/PWM Register

B15	B14	B13	B12	B11	B10	B9	B8
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
TMR0CCPR2							

B7	B6	B5	B4	B3	B2	B1	B0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
TMR0CCPR2							

14.3 Operation

The 16-bit timer/counter itself is an up counter and increment either on the rising or falling (external clock only) edge of clock source. The initial value of the up counter is stored in the pre-load register (P_Timer0_Preload) . When the counter is enabled or overflow occurs, the initial value is loaded into counter on the next increment clock edge (synchronous load). So, if the initial value is FFFC, the counter will count by the sequence 0xFFFC, 0xFFFD, 0xFFFE, 0xFFFF, 0xFFFC, 0xFFFD, 0xFFFE, 0xFFFF, 0xFFFC, 0xFFFD, … , and so on. The TMR0IF interrupt flag is set when Timer 0 overflows.

Note: If the preload data is 0xFFFF, the interrupt occurs every timer clock.

The clock sources of Timer 0 can be generated by the internal clock sources (timer mode) or from external pin (counter mode).

1. In timer mode, the clock source of Timer 0 is programmable by selecting the internal clock sources such as CPUCLK/2, CPUCLK/256, 32768Hz, 8192Hz, 4096Hz, 2048Hz, 1024Hz, 256Hz, 4Hz, 2Hz, 1Hz, or some combinations by source A and B. The clock can be synchronized by CPUCLK before entering Timer 0 by set the corresponding control bit (P_Timer0_Ctrl.**SYNCLK)**.
2. In counter mode, the clock source of Timer0 is generated by external clock source pin (EXT1 and EXT2). Additionally, EXT2 can be divided by 1, 4, 16 prescale divider and capable of selecting clock edge. The EXT1 cannot select prescale divider and only trigger as rising edge. The clock source is synchronized with CPUCLK. In order to make sure the synchronization procedure is completed, the clock rate has to be half of CPUCLK (CPUCLK/2) or less.

14.4 Timer0 Interrupt

The timer 0 interrupt is generated when the Timer0 register overflow from 0xFFFF to 0x0000. This overflow sets bit TMR0IF when the TMR0EN bit setting. The interrupt flag can be masked by clearing bit (TMR0IEN). In addition, when the TMR0IF set and go in interrupt and TMR0IF must be cleared at software by the timer 0 interrupt before re-enabling this interrupt. The Timer 0 interrupt can awaken the processor from sleep since the timer is shut-off during sleep. The timer 0 interrupt can be two ways entrances. IRQ4 is a default entrance, and FIQ set by P_INT_Priority Bit 6 (TMR0IP).

14.5 Design Tips

1. How to set timer0 frequency?

The timer 0 frequency is depended on the timer 0 source clock and preload data. The clock source has three sources. One is Source A and Source B, another is EXT1 and the other is EXT2. After the clock source is selected. Preload data should be assigned.

For instance: If we intend to select timer0 as 16K:

[P_Timer0_Preload] = 0xFFFF - (Source_Clock) / 16K + 1

? **Example 12-1**? If CPU clock is selected 24M, select timer0 as 16K

A) Use assemble language

```
R1=0x0030;     //Select Timer1 source clock
CPU/2
[P_Timer0_Ctrl] = R1;
R1 = 0xFFFF – 24M/16K +1;
[P_Timer0_Preload] = R1;
```

B) Use C language

```
*P_Timer1_Ctrl = 0x0030;
*P_Timer1_Preload = 0xFFFF – 24M/16K +1;
```

2. How to enable Timer0 interrupt

【**Example12-2**】Enable Timer1 interrupt

A) Use assemble language

```
R1 = [P_Timer1_Ctrl];
//enable TMR1EN ,TMR1IEN and clear
TMR1IF
R1 |= 0xE000;
[P_Timer1_Ctrl] = R1;
IRQ ON;
```

B) Use C language

```
unsigned int TimeData;
TimeData = Get_Timer1_Ctrl();
TimeData |=  0xE000;
Set_Timer1_Ctrl (TimeData);
IRQ_ON();
```

【**Example12-3**】Test Timer0 interrupt ,Select Timer0 clock source as 4096Hz . Timer0 overflow Every 1 second, when Timer0 overflow, the PortD status changes.

```
//=========================    start of main.c    =================================//
    #include        "SPMC701FM0.H"
    #include        "unSPMACRO.H"
    unsigned int uiFlag = 0;
    main()
    {
        *P_IOD_Dir = 0xffff ;       //Set PortD as output
         *P_IOD_Attrib = 0xffff ;
        *P_IOD_Data = 0x0000;

        //Enable Timer0, Enable Timer0 interrupt,
        Set_Timer0_Ctrl(C_TMR0EN+C_TMR0IEN+C_CLKBFS_1+C_CLKAFS_4096Hz+C_TMR0IF) ;
        Set_Timer0_Preload(0xffff-4096) ;    //Set the preload value
        INT_IRQ() ;
        while(1)
        {
            if (g_iTimer0_Flag)
            Set_IOD_Data(0xffff);
            else
            Set_IOD_Data(0x0000);
        }
    }
//=========================    end of main.c    =================================//

//=========================    start of IRQ.c    =================================//
    #include        "SPMC701FM0.H"
    #include        "unSPMACRO.H"
    extern  uiFlag;

    void IRQ4(void) __attribute__ ((ISR));
    void IRQ4(void)
    {
        if (Get_INT_Status() & C_TMR0INT)
        {
            Set_Timer0_Ctrl(Get_Timer0_Ctrl());
            uiFlag ^= 0xFFFF;
        }
    }
    //=========================    end of IRQ.c =================================//
```

15 Timer 1

15.1 Introduction

SPMC70xxx series provides 16-bit timer/counter Timer1 and cooperating with another three 16-bit registers, P_Timer1_Preload, P_Timer1_CCPR and P_Timer1_CCPR2, these timer and counters can support Capture, Compare and PWM functions. The CCP mode, please refer to Chapter 15.

The Timer1 module timer and counter has the following features:

1. Timer mode (internal clock) or counter mode (external clock).
2. Flexible clock source by source A
3. Interrupt on overflow from 0xFFFF to 0x0000;
4. Edge select for external clock
5. Synchronized with system clock or not
6. When the timer is turned on, the content of P_Timer1_Preload is loaded into timer counter. This can be used to initialize the counter.

The source of Timer1 only has one source A. The Figure 13-1 shows the time 1 block structure.

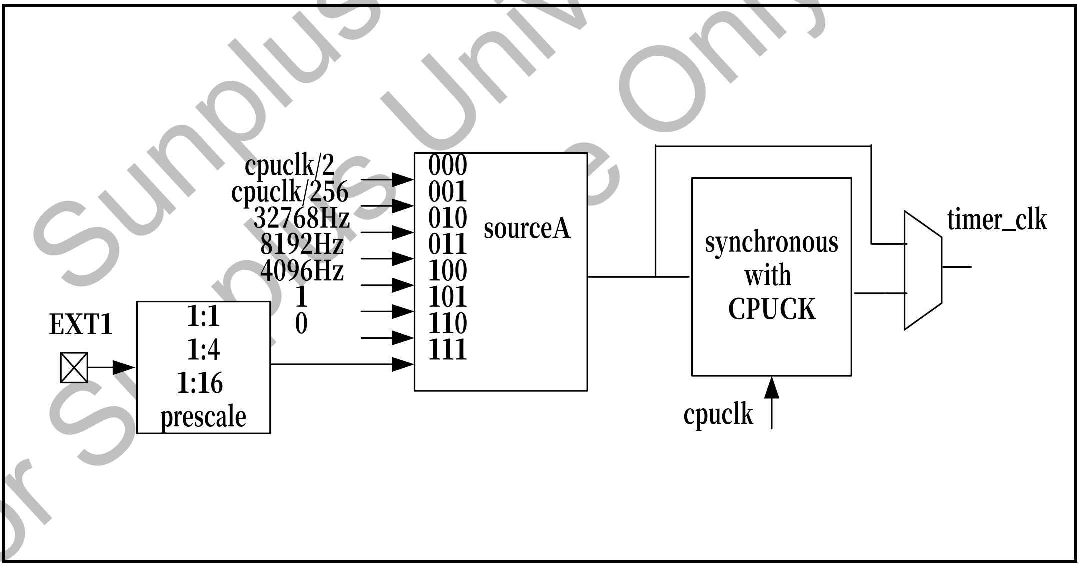


Figure 13-1: Timer 1 Block Structure

15.2 Control Register

15.2.1 P_Timer1_Ctrl ($7048): Timer1 control/status register

B15	B14	B13	B12	B11	B10	B9	B8
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
TMR1EN	TMR1IF	TMR1IEN	EXT1PS		CCP1MS		

B7	B6	B5	B4	B3	B2	B1	B0
R/W	R/W	-	-	-	R/W	R/W	R/W
0	0	-	-	-	0	0	0
CCP1MS	SYNCLK	-	-	-	CLKAFS		

Bit 15 TMR1EN: Timer1 Enable
1= Enable Timer1 Counter
0= Disable Timer1 Counter

Bit 14 TMR1IF: Timer1 Interrupt Flag
1= Timer1 Interrupt asserted
0= No Timer1 interrupt
This bit is set when timer1 rollover from 0xFFFF to 0x0000; and is cleared by software writing this bit '1'

Bit 13 TMR1IEN: Timer1 Interrupt Enable
1= Enable
0= Disable
This bit is used for enabling timer 1 interrupt. It is different from Bit15 TMR1EN. If only enable TMR1IEN else enable TMR1EN, the timer1 interrupt can't be enabled.

Bit 12:11 EXT1PS: Counter mode prescale selection for EXT1
00= every falling
01= every rising
10= every 4 risings
11= every 16 risings

Bit 10:7 CCP1MS: Timer1 CCP1 Mode Selection
0000= Capture/Compare/PWM off
0100= Capture mode, every falling
0101= Capture mode, every rising
0110= Capture mode, every 4 risings
0111= Capture mode, every 16 risings
1000= Compare mode, set output (CCP1IF set)
1001= Compare mode, clear output (CCP1IF set)
101X= Compare mode, CCP1 unaffected (CCP1IF set)
1100= PWM mode, not return to one output (NRO)
1101= PWM mode, not return to zero output (NRZ)
1110= PWM mode, return to one output (RTO)
1111= PWM mode, return to zero output (RTZ)

Bit 6 SYNCLK: The clock source of TMR1 is synchronized by CPU clock
1= Synchronized
0= Not synchronized

Bit 5:3 reserved

Bit 2:0 CLKAFS: Clock source A frequency selection
000= CPUCLK/2
001= CPUCLK/256
010= 32768Hz
011= 8192Hz
100= 4096Hz
101= 1

110= 0
111= EXT1

15.2.2 P_Timer1_Preload ($7049): Timer1 Preload Register

B15	B14	B13	B12	B11	B10	B9	B8
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
TMR1PR							

B7	B6	B5	B4	B3	B2	B1	B0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
TMR1PR							

15.2.3 P_Timer1_CCPR ($704A): Timer1 Capture/Compare/PWM Register

B15	B14	B13	B12	B11	B10	B9	B8
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
TMR1CCPR							

B7	B6	B5	B4	B3	B2	B1	B0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
TMR1CCPR							

15.2.4 P_Timer1_CCPR2 ($704B): Timer1 Capture/Compare/PWM Register

B15	**B14**	**B13**	**B12**	**B11**	**B10**	**B9**	**B8**
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**
TMR1CCPR2							

B7	**B6**	**B5**	**B4**	**B3**	**B2**	**B1**	**B0**
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**
TMR0CCPR2							

15.3 Operation

The 16-bit timer/counter itself is an up counter and increment either on the rising or falling (external clock only) edge of clock source. The initial value of the up counter is stored in the pre-load register(P_Timer1_Preload). When the counter is enabled or overflow occurs, the initial value is loaded into counter on the next increment clock edge (synchronous load). Therefore, if the initial value is 0xFFFC, the counter will count by the sequence of 0xFFFC, 0xFFFD, 0xFFFE, 0xFFFF, 0xFFFC, 0xFFFD, 0xFFFE, 0xFFFF, 0xFFFC, 0xFFFD, etc. The TMR1IF interrupt flag is set when Timer1 overflows.

Note: If the preload data is FFFF, the interrupt occurs every timer clock.

The clock sources of Timer 1 can be generated by the internal clock sources (timer mode) or from external pin (counter mode).

1. In timer mode, the clock source of Timer 1 is programmable by selecting the internal clock sources such as CPUCLK/2, CPUCLK/256, 32768Hz, 8192Hz, and 4096Hz. The clock can be synchronized by CPUCLK before entering Timer 1 by set the corresponding control bit (P_Timer1_Ctrl. **SYNCLK)**.
2. In counter mode, the clock source of Timer 1 is generated by external clock source pin (PB9). Additionally, EXT1 can be divided by 1, 4, 16 prescale divider and capable of selecting clock edge. The clock source is synchronized with CPUCLK. In order to make sure the synchronization procedure is completed, the clock rate has to be half of CPUCLK (CPUCLK/2) or less.

15.4 Timer1 Interrupt

The timer 1 interrupt is generated when the Timer1 register overflow from 0xFFFF to 0x0000. This overflow sets bit TMR1IF when the TMR1EN bit setting. The interrupt flag can be masked by clearing bit (TMR1IEN). In addition, when the TMR1IF set and go in interrupt and TMR1IF must be cleared at software by the timer 1 interrupt before re-enabling this interrupt. The timer 1 interrupt can awaken the processor from sleep since the timer is shut-off during sleep.

The timer 1 interrupt can be two ways entry. IRQ4 is default entry, and FIQ set by P_INT_Priority Bit 7 (TMR1IP).

15.5 Design Tips

1. How to set timer 1 frequency?

The timer 1 frequency depends on the timer 1 source clock and pre-load data. The source clock has 2 kinds of sources. One is Source A, and the other is EXT1. After the clock source is selected. Pre-load data should be assigned. For instance: to select timer1 as 8K:

[P_Timer1_Preload] = 0xFFFF- (Source_Clock) / 8K +1

【**Example13-1**】 If CPU clock is selected 24M, select timer1 as 8K

A) Use assemble language

```
 R1=0x0030;      //Select Timer1 source clock
CPU/2
 [P_Timer1_Ctrl] = R1;
 R1 = 0xFFFF – 24M/8K +1;
 [P_Timer1_Preload] = R1;
```

B) Use C language

```
*P_Timer1_Ctrl = 0x0030;
*P_Timer1_Preload = 0xFFFF – 24M/8K +1;
```

2. How to enable Timer1 interrupt

【**Example13-2**】Enable Timer1 interrupt

A) Use assemble language

```
R1 = [P_Timer1_Ctrl];
//enable TMR1EN ,TMR1IEN and clear
TMR1IF
R1 |= 0xE000;
[P_Timer1_Ctrl] = R1;
IRQ  ON;
```

B) Use C language

```
unsigned int TimeData;
TimeData = Get_Timer1_Ctrl();
TimeData |=   0xE000;
Set_Timer1_Ctrl (TimeData);
IRQ_ON();
```

16 Timer 2

16.1 Introduction

SPMC70xxx series provides 16-bit timer/counter Timer2 and cooperating with one 16-bit P_Timer2_Preload register.

The Timer2 module timer and counter has the following features:

1. Timer mode (internal clock) or counter mode (external clock)
2. Flexible clock source by source A
3. Interrupt on overflow from 0xFFFF to 0x0000
4. Edge select for external clock
5. Synchronized with system clock or not
6. When the timer is turn on, the content of P_Timer2_Preload is loaded into timer counter. This can be used to initialize the counter.

The source of Timer2 only has one source A. The following Figure 14-1 shows the time2 block structure.

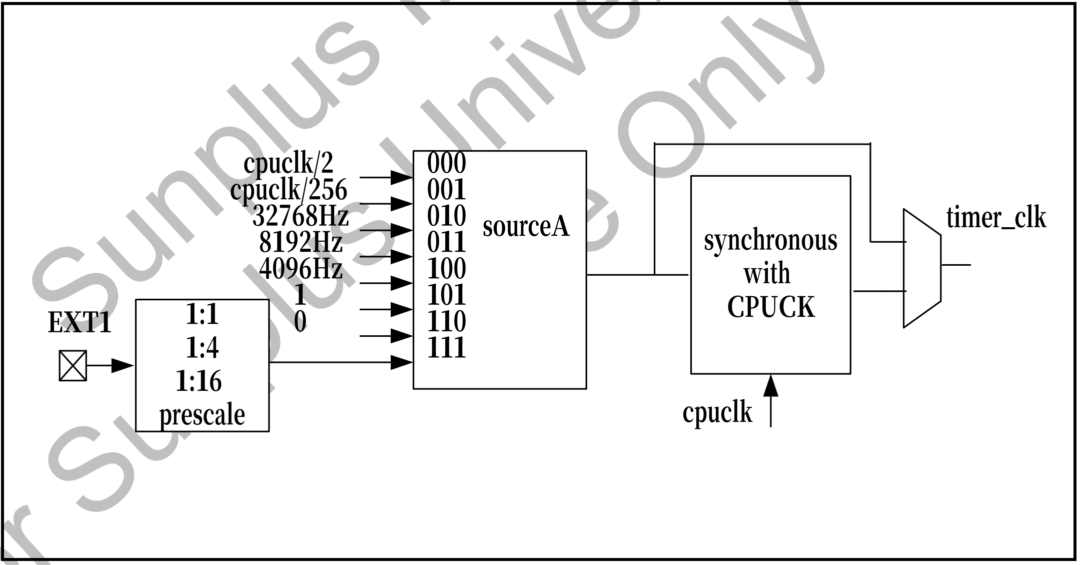


Figure 14-1 Timer 2 Block Structure

16.2 Control Register

16.2.1 P_Timer2_Ctrl ($704C) : Timer2 control/status register

B15	B14	B13	B12	B11	B10	B9	B8
R/W	R/W	R/W	R/W	R/W	-	-	-
0	0	0	0	0	-	-	-
TMR2EN	TMR2IF	TMR2IEN	EXT1PS		-	-	-

B7	B6	B5	B4	B3	B2	B1	B0
-	R/W	-	-	-	R/W	R/W	R/W
-	0	-	-	-	0	0	0
-	SYNCLK	-	-	-	CLKAFS		

Bit 15 TMR2EN: Timer2 Enable

1= Enable Timer2 Counter

0= Disable Timer2 Counter

Bit 14 TMR2IF: Timer2 Interrupt Flag

1= Timer2 Interrupt asserted

0= No Timer2 interrupt

This bit is set when timer2 rollover from 0xFFFF to 0x0000 and is cleared by software writing this bit '1'

Bit 13 TMR2IEN: Timer2 Interrupt Enable

1= Enable

0= Disable

This bit is used for enabling timer 2 interrupt. It's different from Bit15 TMR1EN. If only enable TMR21E else enable TMR2EN, the timer2 interrupt cannot be enabled.

Bit 12:11 EXT1PS: Counter mode prescale selection for EXT1

00= every falling

01= every rising

10= every 4 rising

11= every 16 rising

Bit 10:7 Reserved

Bit 6 SYNCLK: The clock source of TMR2 is synchronized by CPU clock

1= Synchronized

0= Not synchronized

Bit 5:3 reserved

Bit 2:0 CLKAFS: Clock source A frequency selection

000= CPUCLK/2

001= CPUCLK/256

010= 32768Hz

011= 8192Hz

100= 4096Hz

101= 1

110= 0

111= EXT1

16.2.2 P_Timer2_Preload ($704D): Timer2 Pre-load Register

B15	B14	B13	B12	B11	B10	B9	B8
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
TMR2PR							

B7	B6	B5	B4	B3	B2	B1	B0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
TMR2PR							

16.3 Operation

The 16-bit timer/counter itself is an up counter and increment either on the rising or falling (external clock only) edge of clock source. The initial value of the up counter is stored in the pre-load register(P_Timer2_Preload). When the counter is enabled or overflow occurs, the initial value is loaded into counter on the next increment clock edge (synchronous load). Therefore, if the initial value is FFFC, the counter will count by the sequence 0xFFFC, 0xFFFD, 0xFFFE, 0xFFFF, 0xFFFC, 0xFFFD, 0xFFFE, 0xFFFF, 0xFFFC, 0xFFFD … etc. The TMR2IF interrupt flag is set when Timer 1overflows.

Note: If the preload data is 0xFFFF, the interrupt occurs every timer clock.

The clock sources of Timer2 can be generated by the internal clock sources (timer mode) or from external pin (counter mode).

1. In timer mode, the clock source of Timer 2 is programmable by selecting the internal clock sources such as CPUCLK/2, CPUCLK/256, 32768Hz, 8192Hz, and 4096Hz. The clock can be synchronized by CPUCLK before entering Timer 1 by set the corresponding control bit (P_Timer2_Ctrl.**SYNCLK)**.
2. In counter mode, the clock source of Timer 2 is generated by external clock source pin (EXT1). Additionally, EXT1 can be divided by 1, 4, 16 prescale divider and capable of selecting clock edge. The clock source is synchronized with CPUCLK. In order to make sure the synchronization procedure is completed, the clock rate has to be half of CPUCLK (CPUCLK/2) or less.

16.4 Timer2 Interrupt

The timer 2 interrupt is generated when the Timer2 register overflows from 0xFFFF to 0x0000. This overflow sets bit TMR2IF when the TMR2EN bit setting. The interrupt flag can be masked by clearing bit (TMR2IEN). When the TMR2IF sets and goes in interrupt and TMR2IF must be cleared in software by the timer 2 interrupt before re-enabling this interrupt. The timer 2 interrupt can awake the processor from sleep since the timer is shut-off during sleep.

The timer 2 interrupt can be two-way entries. IRQ4 is default entry, and FIQ set by P_INT_Priority Bit 8 (TMR2IP).

16.5 Designed Tips

How to set timer 2 frequency?

The timer2's frequency depends on the timer2's source clock and pre-load data. The source clock has 2 types of sources: Source A and EXT1.After the clock source is selected. Pre-load data should be assigned.

For instance:To select timer2 as 12K

[P_Timer2_Preload] = 0xFFFF- (Source_Clock) / 12K +1

【Example14-1】 If CPU clock is select 24M, select timer2 as 12K

A) Use assemble language

```
 R1=0x0030;      //Select Timer1 source clock
CPU/2
 [P_Timer2_Ctrl] = R1;
 R1 = 0xFFFF – 24M/128K +1;
 [P_Timer2_Preload] = R1;
```

B) Use C language

```
Set_Timer2_Ctrl (0x0030);
Set_Timer2_Preload (0xFFFF – 24M/12K +1);
```

2: How to enable Timer2 interrupt?

【Example14-2】 Enable Timer1 interrupt

A) Use assemble language

```
R1 = [P_Timer2_Ctrl];
//enable TMR2EN ,TMR2IEN and clear TMR2IF
R1 |= 0xE000;
[P_Timer2_Ctrl] = R1;
IRQ  ON;
```

B) Use C language

```
unsigned int TimeData;
TimeData = Get_Timer2_Ctrl();
TimeData |=   0xE000;
Set_Timer2_Ctrl (TimeData);
IRQ_ON();
```

17 CCP Mode

17.1 Introduction

SPMC70xxx series provides two CCP (Capture/Compare/PWM) modes. Each CCP module contains a 16-bit register, which can operate as a16-bit capture register, as a 16-bit compare register or a 15-bit PWM register. The output or input pin of CCP modules shared with PB11 and PB12 and named as CCP0 and CCP1. The relation of timer, IO and register are shown in table 15-1.

Table 15-1 The relation of timer, IO and register

Name	Share pin	Use timer	Other function	Register
CCP0	PB11	Timer0	Capture0 PWM0	P_Timer0_Preload P_Timer0_CCPR P_Timer0_CCPR2
CCP1	PB12	Timer1	Capture1 PWM1	P_Timer1_Preload P_Timer1_CCPR P_Timer1_CCPR2

17.2 Control Register

17.2.1 P_Timer0_Ctrl ($7044) : Timer0 control/status register

B15	B14	B13	B12	B11	B10	B9	B8
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
TMR0EN	TMR0IF	TMR0IEN	EXT2PS		CCP0MS		

B7	B6	B5	B4	B3	B2	B1	B0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
CCP0MS	SYNCLK	CLKBFS			CLKAFS		

Bit 15 TMR0EN: Timer0 Enable

1= Enable Timer0 Counter or CCP mode

0= Disable Timer0 Counter or CCP mode

Bit 14 TMR0IF: Timer0 Interrupt Flag

1= Timer0 Interrupt asserted

0= No Timer0 interrupt

As timer mode, this bit is set when Timer0 rollover from 0xFFFF to 0x0000,and is cleared by software writing this bit '1'. In addition, as CCP mode, this bit is set when CCP mode condition match.

Bit 13 TMR0IEN: Timer0 Interrupt Enable

1= Enable

0= Disable

This bit is used for enabling Timer0 interrupt. It's different from Bit15 TMR0EN. If only enable TMR0IEN else enable TMR0EN, the Timer0 interrupt can't be enabled.

Bit 12:11 EXT2PS: Counter mode prescale selection for EXT2

00= every falling

01= every rising

10= every 4 rising

11= every 16 rising

Bit 10:7 CCP0MS: Timer0 CCP0 Mode Selection

0000= Capture/Compare/PWM off

0100= Capture mode, every falling

0101= Capture mode, every rising

0110= Capture mode, every 4 rising

0111= Capture mode, every 16 rising

1000= Compare mode, set output (CCP0IF set)

1001= Compare mode, clear output (CCP0IF set)

101X= Compare mode, CCP0 unaffected (CCP0IF set)

1100= PWM mode, not return to one output (NRO)

1101= PWM mode, not return to zero output (NRZ)

1110= PWM mode, return to one output (RTO)

1111= PWM mode, return to zero output (RTZ)

Bit 6 SYNCLK: The clock source of TMR0 is synchronized by CPU clock

1= Synchronized

0= Not synchronized

Bit 5:3 CLKBFS: Clock Source B frequency selection

000= 2048Hz

001= 1024Hz

010= 256Hz

011= TMB1

100= 4Hz

101= 2Hz

110= 1

111= EXT2

Bit 2:0 CLKAFS: Clock Source A frequency selection

000= CPUCLK/2

001= CPUCLK/256

010= 32768Hz

011= 8192Hz

100= 4096Hz

101= 1

110= 0

111= EXT1

17.2.2 P_Timer0_Preload ($7045): Timer0 Pre-load Register

B15	B14	B13	B12	B11	B10	B9	B8
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
TMR0PR							

B7	B6	B5	B4	B3	B2	B1	B0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
TMR0PR							

17.2.3 P_Timer0_CCPR ($7046): Timer0 Capture/Compare/PWM Register

B15	B14	B13	B12	B11	B10	B9	B8
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
TMR0CCPR							

B7	B6	B5	B4	B3	B2	B1	B0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
TMR0CCPR							

17.2.4 P_Timer0_CCPR2 ($7047): Timer0 Capture/Compare/PWM Register

B15	B14	B13	B12	B11	B10	B9	B8
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
TMR0CCPR2							

B7	B6	B5	B4	B3	B2	B1	B0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
TMR0CCPR2							

17.2.5 P_Timer1_Ctrl ($7048): Timer1 control/status register

B15	B14	B13	B12	B11	B10	B9	B8
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
TMR1EN	TMR1IF	TMR1IEN	EXT1PS		CCP1MS		

B7	B6	B5	B4	B3	B2	B1	B0
R/W	R/W	-	-	-	R/W	R/W	R/W
0	0	-	-	-	0	0	0
CCP1MS	SYNCLK	-	-	-	CLKAFS		

Bit 15 TMR1EN: Timer1 Enable

1= Enable Timer1 Counter

0= Disable Timer1 Counter

Bit 14 TMR1IF: Timer1 Interrupt Flag

1= Timer1 Interrupt asserted

0= No Timer1 interrupt

This bit is set when timer1 rollover from 0xFFFF to 0x0000; and is cleared by software writing this bit ‘1’

Bit 13 TMR1IEN: Timer1 Interrupt Enable

1= Enable

0= Disable

This bit is used for enabling timer 1 interrupt. It’s different from Bit15 TMR1EN. If only enable TMR1IEN else enable TMR1EN, the timer1 interrupt can’t be enabled.

Bit 12:11 EXT1PS: Counter mode prescale selection for EXT1

00= every falling

01= every rising

10= every 4 rising

11= every 16 rising

Bit 10:7 CCP1MS: Timer1 CCP1 Mode Selection

0000= Capture/Compare/PWM off

0100= Capture mode, every falling

0101= Capture mode, every rising

0110= Capture mode, every 4 rising

0111= Capture mode, every 16 rising

1000= Compare mode, set output (CCP1IF set)

1001= Compare mode, clear output (CCP1IF set)

101X= Compare mode, CCP1 unaffected (CCP1IF set)

1100= PWM mode, not return to one output (NRO)

1101= PWM mode, not return to zero output (NRZ)

1110= PWM mode, return to one output (RTO)

1111= PWM mode, return to zero output (RTZ)

Bit 6 SYNCLK: The clock source of TMR1 is synchronized by CPU clock

1= Synchronized

0= Not synchronized

Bit 5:3 reserved

Bit 2:0 CLKAFS: Clock source A frequency selection

000= CPUCLK/2

001= CPUCLK/256

010= 32768Hz

011= 8192Hz

100= 4096Hz

101= 1

110= 0

111= EXT1

17.2.6 P_Timer1_Preload ($7049): Timer1 Pre-load Register

B15	B14	B13	B12	B11	B10	B9	B8
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
TMR1PR							

B7	B6	B5	B4	B3	B2	B1	B0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
TMR1PR							

17.2.7 P_Timer1_CCPR ($704A): Timer1 Capture/Compare/PWM Register

B15	B14	B13	B12	B11	B10	B9	B8
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
TMR1CCPR							

B7	B6	B5	B4	B3	B2	B1	B0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
TMR1CCPR							

17.2.8 P_Timer1_CCPR2 ($704B): Timer1 Capture/Compare/PWM Register

B15	B14	B13	B12	B11	B10	B9	B8
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
TMR1CCPR2							

B7	B6	B5	B4	B3	B2	B1	B0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
TMR1CCPR2							

17.3 Operation

17.3.1 Capture Mode

The corresponding GPIO pin for CCPx is configured as input pin automatically when TimerX is set as capture mode. The value of TimerX is latched and stored in Capture/Compare/PWM registers (P_Timerx_CCPR) at the selected edge of CCPx. If the clock source of TimerX is synchronized by CPU clock, the clock rate of CCPx has to be half of CPUCLK or less in timer or counter mode. Nevertheless, if the clock source of TimerX is not synchronized by CPU clock, the clock rate of CCPx has to be half of clock source in timer or counter mode. There are two CCPRs in timer modules. The value of TimerX can be latched to the first CCP register (P_Timerx_CCPR) for every 1, 4 or 16 CCPx rising/falling edges. In case of every rising/falling edge capture mode selected, the value of TimerX can be latched to the second CCP register (P_Timerx_CCPR2) at every falling/rising edge, which is the opposite edge as the first selected. In this way, CCPx pulse width can be measured by this approach. When a value is captured into the first CCP register (P_Timerx_CCPR), the interrupt flag is set (TMRxIF). The interrupt flag must be cleared by firmware. If another capture occurs before the value in register CCP register (P_Timerx_CCPR) is read, the old captured value will be lost. The relationship of the capture mode shows in figure 15-1.

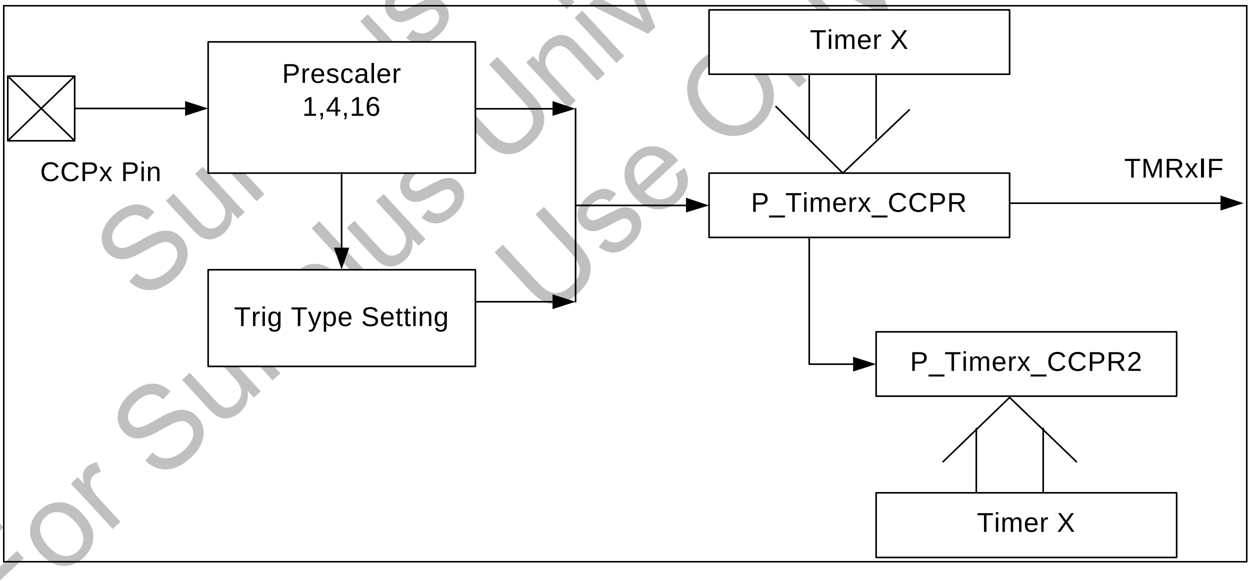


Figure 15-1 The relationship of the capture mode

17.3.2 Compare Mode

In compare mode, the value to be compared with TimerX is stored in CCP Register (**P_Timerx_CCPR**). When a match occurs, the TimerX is reloaded by the value in **P_Timerx_Preload**. The corresponding CCPx pin is configured as an output pin automatically when compare mode is set. When a match occurs, the CCPx pin can be set output or clear output according to configuration. In compare mode, the overflow of TimerX will not reload **P_Timerx_Preload** value to TimerX, and interrupt flag (TMRxIF) is set when a match occurs. The figure 15-2 shows the two mode of reload timer methods.

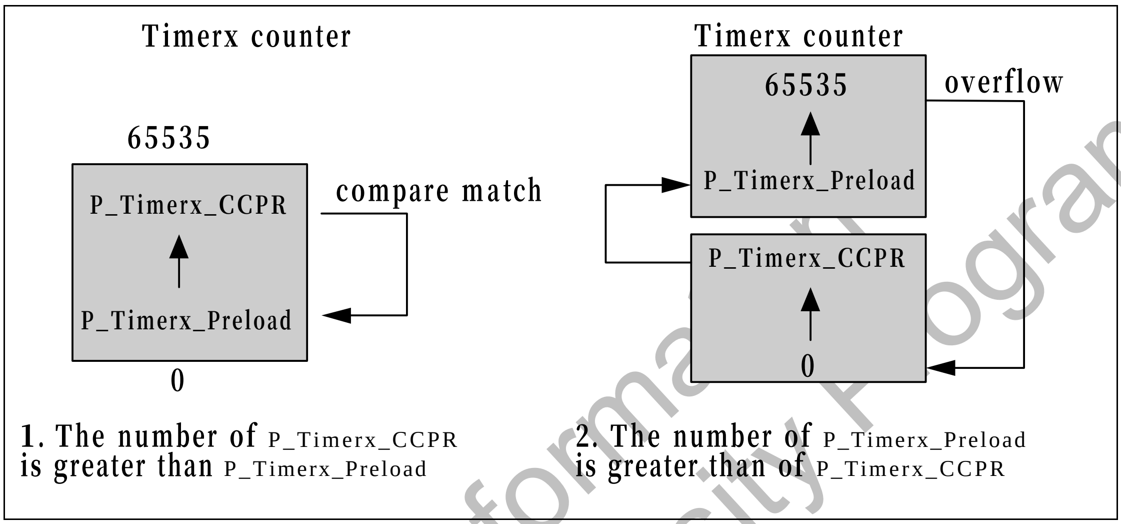

Timerx counter
65535
P_Timerx_CCPR
P_Timerx_Preload
0
compare match
1. The number of P_Timerx_CCPR is greater than P_Timerx_Preload
Timerx counter
65535
overflow
P_Timerx_Preload
P_Timerx_CCPR
0
2. The number of P_Timerx_Preload is greater than of P_Timerx_CCPR

Figure 15-2 reload timer methods

The compare mode has two output modes: set and clear output. When it is set to output mode, the CCPx pin will keep at low and as comparison matched, CCPx pin outputs a high plus, and the high plus width is the same as the reference timer. When output mode is cleared, the CCPx pin will keep at high and as comparison matched, CCPx pin outputs a low plus, and the low plus width is the same as the reference timer in comparison mode, showing in figure 15-3.

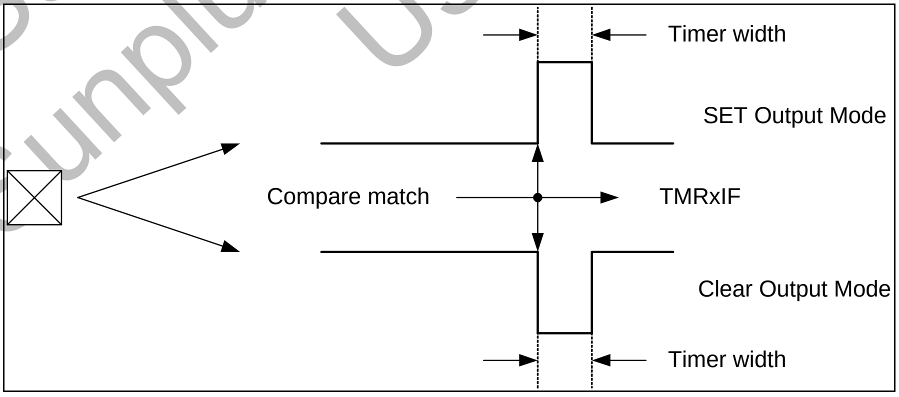


Figure 15-3

17.3.3 PWM Mode

When TimerX is set as PWM mode, the CCPx pin is configured as output pin. The PWM mode also can be separate two modes. One is “not return to one or zero mode”, the other is “return to one or zero mode”. The “not return to one or zero mode” will let CCPx pin changes low/high while P_TimerX_CCPR compares matched with

TimerX, and while TimerX overflow, the CCPx pin can be changed low/high and TimerX reload P_TimerX_Preload value. The "return to one or zero mode" will let CCPx pin changes low/high while P_TimerX_Preload compares matched with TimerX, and the CCPx pin can be changed low/high while P_TimerX_CCPR compares matched with TimerX. While TimerX overflow, then TimerX reload the P_TimerY_Preload value. Example, While Timer1 overflows then reloads the P_Timer0_Preload and While Timer0 overflow then reloads the P_Timer1_Preload. The relationship timing between PWM mod and other mode show on figure 15-4.

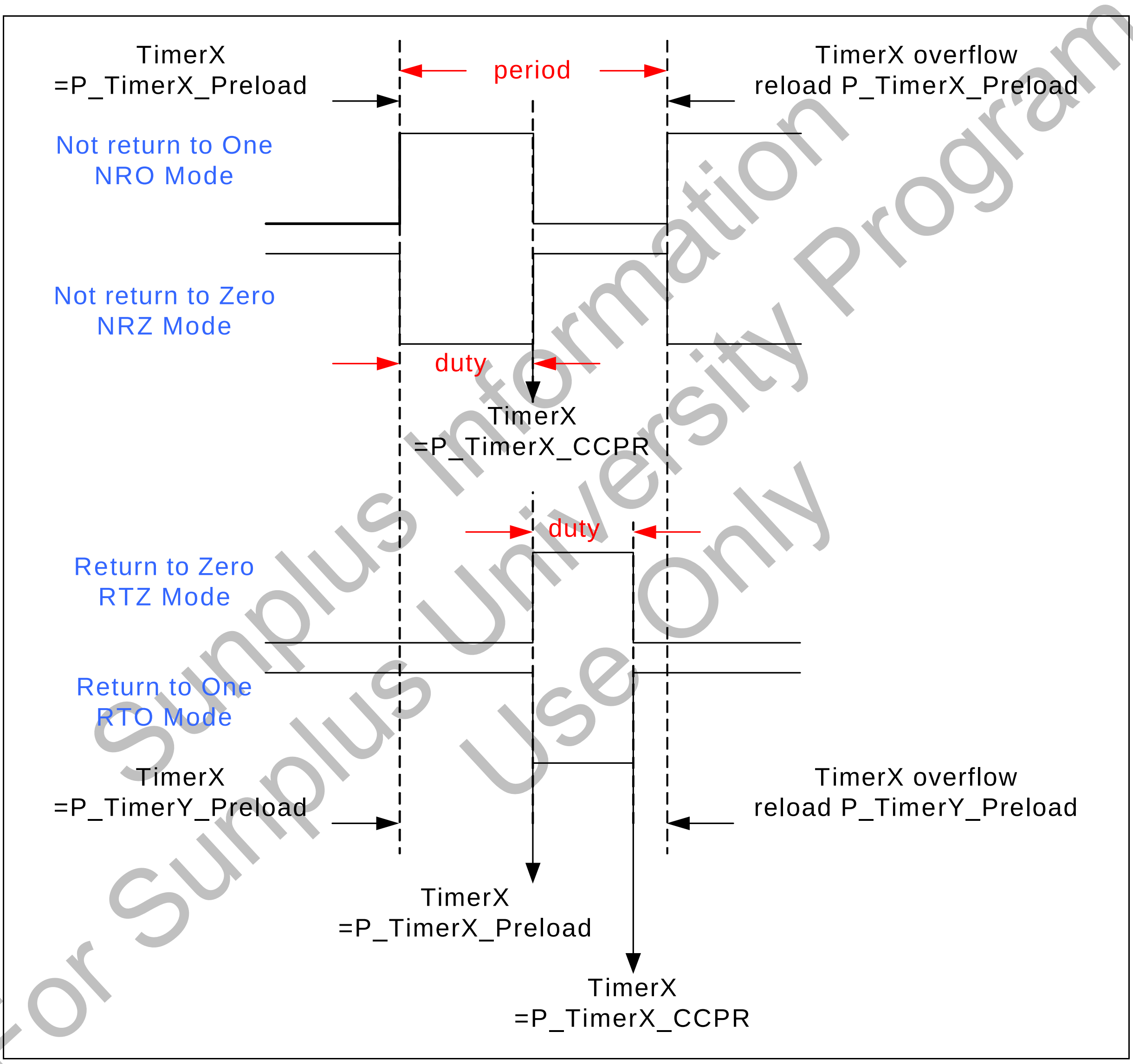


Figure 15-4

In Pulse-Width-Modulation (PWM) mode, the PWM period is set by the P_TimerX_Preload value.

PWM period = (Selected Clock Period) x (65536 –P_TimerX_Preload)

PWM frequency is defined as 1/ [PWM period].

Frequency = 1 / [(Selected Clock Period) x (65536 –P_TimerX_Preload)]

CCPRx sets the duty cycle as follows:

Duty cycle = (Selected Clock Period) x (P_TimerX_CCPR – P_TimerX_Preload + 1)

Note: If P_TimerX_CCPR has to be equal or larger than P_TimerX_Preload or zero for 0% duty cycle.

17.4 CCP interrupt

The CCP interrupt can be two ways entry. IRQ4 is default entry, and FIQ set by P_INT_Priority Bit 7 (TMR1IP).

17.4.1 Capture mode interrupt

The capture mode interrupt is generated when the CCPx pin capture the TimerX into P_TimerX_CCPR. This interrupt is same with this timer's interrupt. When the capture condition occurs, then sets bit TMRxIF when the TMRxEN bit setting. The interrupt can be masked by clearing bit (TMRxIE). In addition, when the TMRxIF sets and goes in interrupt , TMRxIF must be cleared at software before re-enabling this interrupt. The capture mode interrupt can awaken the processor from sleep since the timer is shut-off during sleep.

17.4.2 Compare mode interrupt

The compare mode interrupt is generated when the TimerX match the P_TimerX_CCPR. This interrupt is same with this timer's interrupt. When the compare condition occurs then sets bit TMRxIF when the TMRxEN bit setting. The interrupt can be masked by clearing bit (TMRxIE). In addition, when the TMRxIF set and go in interrupt and TMRxIF must be cleared at software before re-enabling this interrupt. The capture mode interrupt can awaken the processor from sleep mode since the timer is shut-off during sleep.

17.4.3 PWM mode interrupt

The PWM mode interrupt is generated when the TimerX overflow . This interrupt is same with this timer's interrupt. When the overflow condition occurs then sets bit TMRxIF when the TMRxEN bit setting. The interrupt can be masked by clearing bit (TMRxIF). Moreover, when the TMRxIF set and go in interrupt and TMRxIF must be cleared at software before re-enabling this interrupt. The capture mode interrupt can awaken the processor from sleep since the timer is shut-off during sleep.

17.5 Design Tips

17.5.1 Capture Mode

The following example shows how to start capture mode and as capture condition occur, you can read the timer value from P_Timer0_CCPR and P_Timer0_CCPR2. User can use this function to check clock period or frequency. Please pay attention on setting timer0 source.

【Example 15-1】 Capture Test.

```
        R1 = 0x8230;            //Enable Capture mode as every falling
        [P_Timer1_Ctrl]   = R1;
WaitSignalInput1:
        R1 = [P_Timer1_Ctrl];   //Check Capture signal input
        R1 &= 0x4000;
        jz WaitSignalInput1
        R2 = [P_Timer1_Ctrl];   //Clear indicate flag
        [P_Timer1_Ctrl] = R2;
        R2 = [P_Timer1_CCPR]//Read Timer Value
WaitSignalInput2:
        R1 = [P_Timer1_Ctrl];   //Check Capture signal input
        R1 &= 0x4000;
        jz WaitSignalInput2
        R1 = [P_Timer1_Ctrl];   //Clear indicate flag
        [P_Timer1_Ctrl] = R1;
        R3 = [P_Timer1_CCPR2]
        R4 = [P_Timer1_CCPR]
        jmp WaitSignalInput1
```

B) Use C language

```
#include "SPMC701FM0.H"
#include "unSPMACRO.H"

main()
{
    unsigned int uiDataTemp,uiData1,uiData2;
    Set_Timer1_Ctrl(C_TMR1EN+C_CCP1MS_CAP1R+C_CLKAFS_Fosc2+C_CLKBFS_1);
    while(1)
```

```
            {
                while(Get_Timer1_Ctrl() & C_TMR1IF)
                 {
                     Set_Timer1_Ctrl(Get_Timer1_Ctrl());
                     uiDataTemp = Get_Timer1_CCPR();
                 }

                 while(Get_Timer1_Ctrl() & C_TMR1IF)
                 {
                     Set_Timer1_Ctrl(Get_Timer1_Ctrl());
                     uiData1 = Get_Timer1_CCPR2();
                     uiData2 = Get_Timer1_CCPR();
                     uiData1 -= uiDataTemp;         //pulse   width
                     uiData2 -= uiDataTemp;         //pulse   cycle
                  }
             }
        }
```

17.5.2 Compare Mode

The following example shows how to start compare mode and it's output condition. Please pay attention on setting timer0 source. The compare mode timing will be reference by timer0 source input. In addition, as compare match condition occur, the CCP0 pin will output signal as user set. The signal pulse equals the timer0 clock source.

【Example 15-2】 Compare_Test:

A) Use assemble language

```
     R1 = 0x5555;
     [P_Timer0_Preload] = R1;
     R1 = 0x555A;
     [P_Timer0_CCPR] = R1;
     R1 = 0x8431;           //Enable Compare mode as output high
     [P_Timer0_Ctrl] = R1;
 WaitCompareMatch:
     R1 = [P_Timer0_Ctrl];   //Check Compare match
     R1 &= 0x4000;
     jz WaitCompareMatch
```

```
        R2 = [P_Timer0_Ctrl];    //Clear indicate flag
        [P_Timer0_Ctrl] = R2;
        jmp WaitCompareMatch
```

B) Use C language

```
    #include        "SPMC701FM0.H"
    #include        "unSPMACRO.H"

    main()
    {
      Set_Timer0_Ctrl(C_TMR0EN+C_CCP0MS_CMPSET+C_CLKBFS_1+C_CLKAFS_Fosc2);
      Set_Timer0_Preload(0xfff5);
      Set_Timer0_CCPR(0xfff8);
      while(1)
      {
            if (Get_Timer0_Ctrl() & C_TMR0IF)
            Set_Timer0_Ctrl(Get_Timer0_Ctrl());     //Clear indicate flag
      }
}
```

17.5.3 PWM Mode

The following example shows how to start PWM mode and it's output condition. Please pay attention on setting timer0 source. The PWM mode timing will be reference by timer0 source input.

【Example 15-3】 PWM_Test:

A) Use assemble language

```
        R1 = 0xFFCC;
        [P_Timer1_CCPR] = R1;
        R1 = 0xFFAA;
        [P_Timer1_Preload] = R1;
        R1 = 0x8631;             //Enable PWM mode as Not Return to One output
        [P_Timer1_Ctrl] = R1;
   WaitPWMMatch:
        R1 = [P_Timer1_Ctrl];    //Check Compare match
        R1 &= 0x4000;
        jz WaitPWMMatch
```

```
R2 = [P_Timer1_Ctrl];    //Clear indicate flag
[P_Timer1_Ctrl] = R2;
jmp WaitPWMMatch
```

B) Use C language

```
    #include      "SPMC701FM0.H"
    #include      "unSPMACRO.H"

    main()
    {
        *P_Timer1_CCPR= 0xFFCC;
        *P_Timer1_Preload=0xFFAA;
        *P_Timer1_Ctrl=C_TMR1EN+C_CCP1MS_PWMNRO+C_CLKBFS_1+C_CLKAFS_4096Hz;
         while(1)
        {
            if (Get_Timer1_Ctrl() & C_TMR1IF)
            Set_Timer1_Ctrl(*P_Timer1_Ctrl);
        }
    }
```

18 Standard Peripheral Interface, SPI

18.1 Introduction

The SPMC70xxx series provide a Standard Peripheral Interface for transmission and receiving data. We name this as SPI function. The SPI supports full-duplex synchronous transfer between a Master device and a Slave device. The IO pins of SPI function are shared with PA15, PB15, PB14 and PB13. The relationship shows bellow Table 16-1.

Table 16-1

PA15	SS	SPI SS pin
PB15	SCK	SPI clock
PB14	SDI	SPI input pin
PB13	SDO	SPI output pin

The SPMC70xxx series supports both master and slave modes for SPI transfer, which is controlled by the SPIMOD bit of the P_SPI_Ctrl($7090) register and the hardware slave select pin $\overline{SS}$. In case of SPIMOD =0, SPI works in master mode. When SPIMOD =1 and SSEN=0, SPI works in slave mode. When SPIMOD =1 and SSEN=1, SPI works in slave mode depending on $\overline{SS}$. If $\overline{SS}$=0, SPI works in slave mode. In this case, SIO must be disabled. The Application Circuit shows in figure 16-1.

The $\overline{SS}$ pin is selected by GPIO of master, and controlled by software.

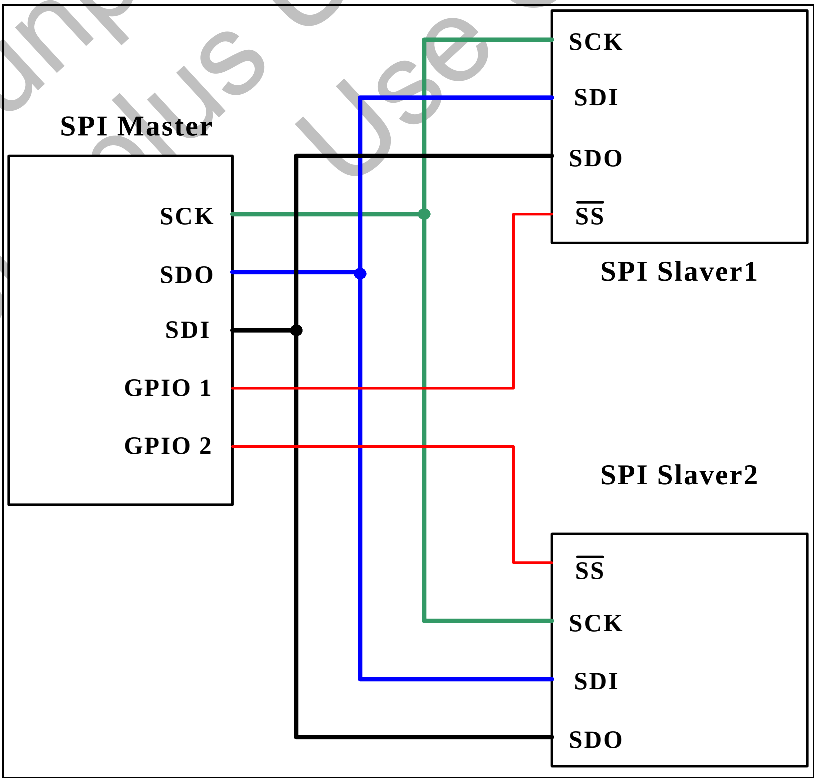


Figure 16-1 The Application Circuit

18.2 Control Register

18.2.1 P_SPI_Ctrl ($7090): SPI control register

B15	B14	B13	B12	B11	B10	B9	B8
-	-	-	-	-	-	R/W	R/W
-	-	-	-	-	-	0	0
-	-	-	-	-	-	SPICSEN	SPIEN

B7	B6	B5	B4	B3	B2	B1	B0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
SPIIEN	SPIMS	SPISMPS	SPIPHA	SPIPOL	SPIFS		

Bit 15:10 Reserved

Bit 9 SPICSEN: $\overline{SS}$ control enable only in slave mode

1= Enable, IOA15 is shared with SIO clock pin (SCL) and $\overline{SS}$ pin

0= Disable

Bit 8 SPIEN: SPI enable

1= Enable

0= Disable

Bit 7 SPIIEN:SPI interrupt enable

1= Enable

0= Disable

Bit6 SPIMS: SPI mode selection

0= Master mode

1= Slave mode

Bit 5 SPISMPS:SPI sample mode selection

1= input data bit sampled at the end of data output time

0= input data bit sampled at the middle of data output time

Bit 4 SPIPHA: SPI clock phase

SPI clock phase select,see SPI Master Mode Timing.

Bit 3 SPIPOL: SPI clock polarity

SPI clock polarity select, ,see SPI Master Mode Timing.

Bit 2:0 SPIFS: Master mode clock frequency selection

000= CPU clock / 4

001= CPU clock / 8

010= CPU clock / 16

011= CPU clock / 32

100= CPU clock / 64

101= CPU clock / 128

110 or 111 reserved

18.2.2 P_SPI_Status $(7091): SPI status register

B15	B14	B13	B12	B11	B10	B9	B8
-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-

B7	B6	B5	B4	B3	B2	B1	B0
-	-	-	-	-	R	R	R/W
-	-	-	-	-	0	0	0
-	-	-	-	-	SPIROR	SPITBF	SPIIF

Bit 15~3 Reserved

Bit 2 SPIROR: Receiving Overrun error indication, meaning the P_SPI_RxBuf has been updated by a new data, whereas the old data has not been read back by the software yet. This bit will be cleared to 0 by the hardware after an I/O read the P_SPI_RxBuf register.

1= Overrun Error occurs

0= No overrun error occurs

Bit 1 SPITBF: Transmission buffer full flag, this bit is set to 1 by the hardware when the transmission buffer is full, it is cleared to 0 when the byte written to the P_SPI_TxBuf has loaded to the shift register. The software should not update SPIBUF when TXBF bit is set, otherwise the new data will overwrite the old data to be send.

1= Transmission buffer full

0= Transmission buffer is empty

Bit 0 SPIIF: SPI Interrupt flag, this bit is set by hardware when the 8 bit TX data has been shifted out through SDO or the 8 bit data has been loaded into SPIBUF. This bit must be cleared by the firmware

18.2.3 P_SPI_TxBuf ($7092) (Write port): SPI Transmission Buffer

B15	B14	B13	B12	B11	B10	B9	B8
-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-

B7	B6	B5	B4	B3	B2	B1	B0
W	W	W	W	W	W	W	W
0	0	0	0	0	0	0	0
SPITXBUF							

Bit 15~8 Reserved

Bit 7~0 SPITXBUF: Write data sends to SDO pin

18.2.4 P_SPI_RxBuf ($7092) (Read port): SPI Receive Buffer

B15	B14	B13	B12	B11	B10	B9	B8
-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-

-	-	-	-	-	-	-	-

B7	B6	B5	B4	B3	B2	B1	B0
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0
SPIRXBUF							

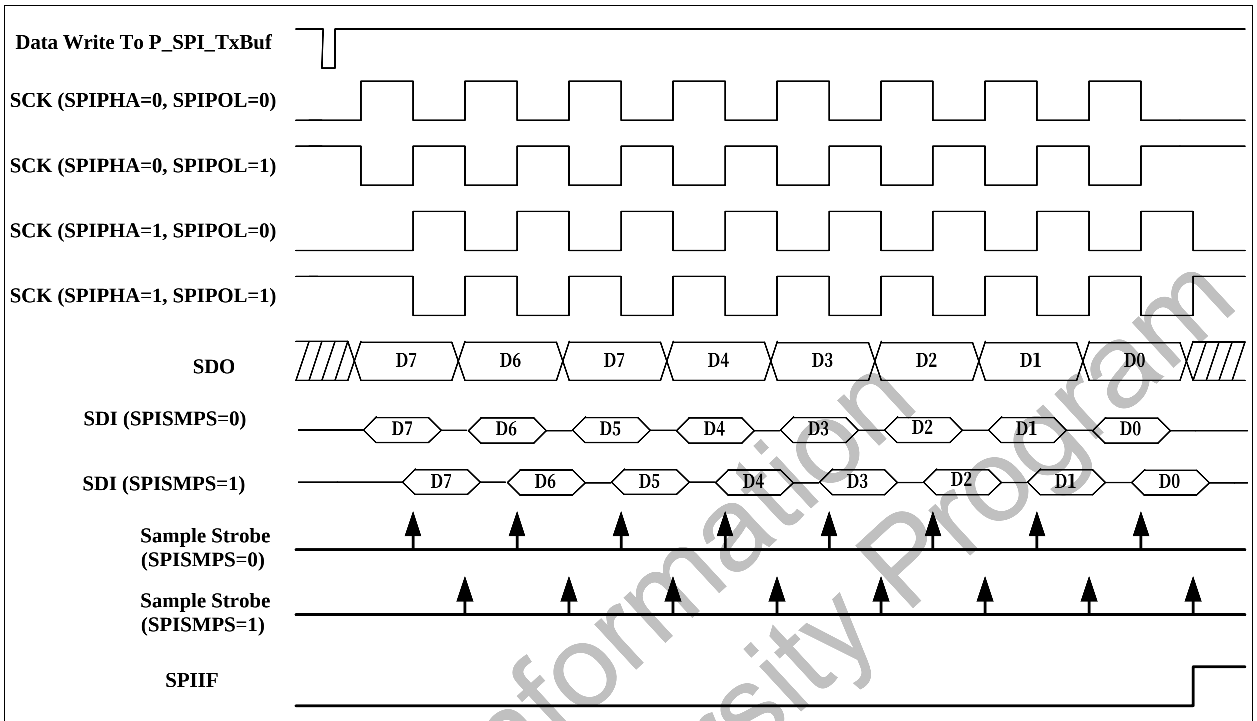
Bit 15~8	reserved
Bit 7~0	SPIRXBUF: Read data from SDI pin

18.3Operation

18.3.1 Master Mode

As in master mode, the shift clock (SCK) is generated by the master's SPI module. There are two register bits (SPIPHA and SPIPOL) to control the clock phase and polarity. The transmission starts from an I/O write to the P_SPI_TxBuf register (write port). If the user just wants to receive the data from the slave device, the firmware still needs to do a dummy write to the P_SPI_TxBuf to initiate the transfer.

After the firmware writes one byte to P_SPI_TxBuf, there are two situation as follows: If the shift register is not shifting data currently, the data will be loaded from P_SPI_TxBuf to the shift register at the first SCK phase and start transmitting. If the shift register is busy shifting data, SPITBF flag in P_SPI_Statue register will be set. The new data will be loaded from P_SPI_TxBuf to shift register when the transmission has finished. When the transmission session is in progress, the data from the slave device is also shifted in through the SDI pin. The data from SDI pin can be sampled according to SPISMPS bit. The received data is then latched to the P_SPI_RxBus (read port) for the firmware reading. When each 8-bit transfer has completed, the SPIIF flag bit of the P_SPI_Status register will be set and an interrupt will be generated if the SPIIEN bit is set.

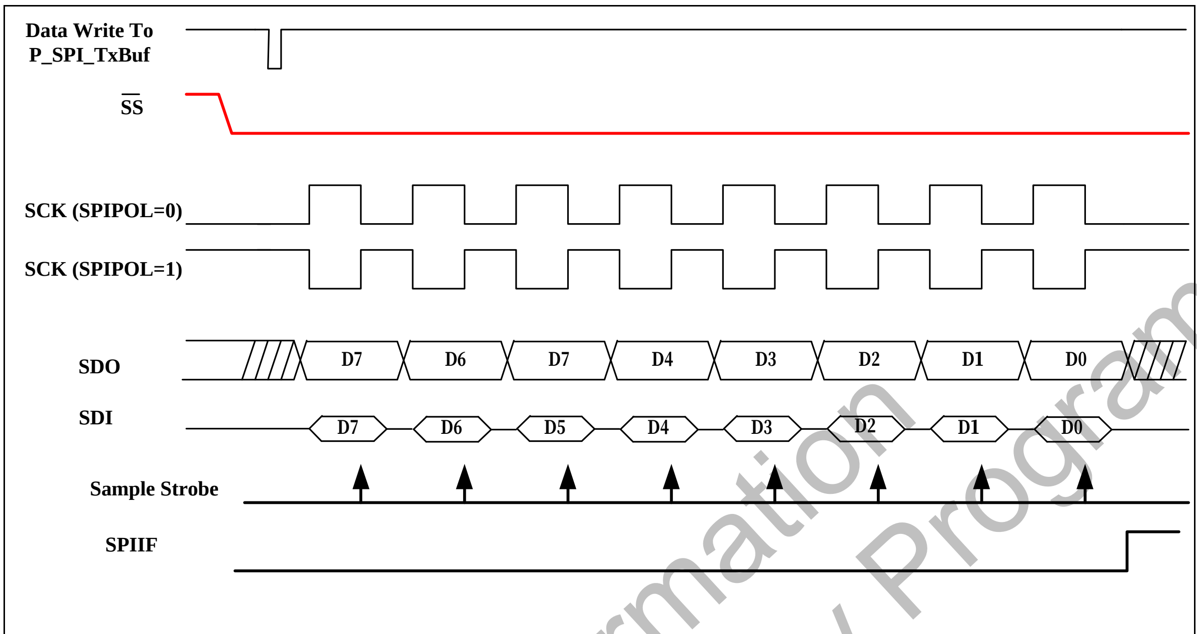


SPI Master Mode Timing

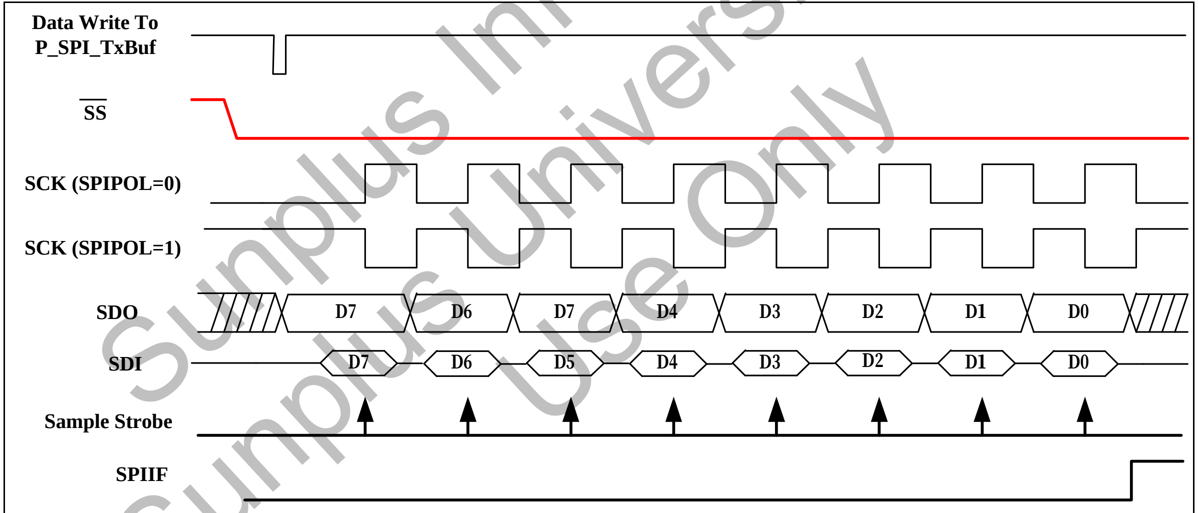
18.3.2 Slave Mode

As in slave mode, the shift clock (SCK) comes from external master's SPI module, so the transmission starts from the first SCK event. To transmit, the firmware should write the data to the P_SPI_TxBuf register before the first SCK from the master. Both master and slave device must be programmed to the same SCK polarity to send and receive the data at the same time. If the clock phase bit "SPIPHA" is set to 1, the 1st data bit being shifted out starts right after the I/O write to P_SPI_TxBuf register. If the clock phase bit SPIPHA is set to 0, the 1st data bit being shifted out starts after first SCK edge.

Note: SPI slave mode and SIO mode cannot be activated at the same time.



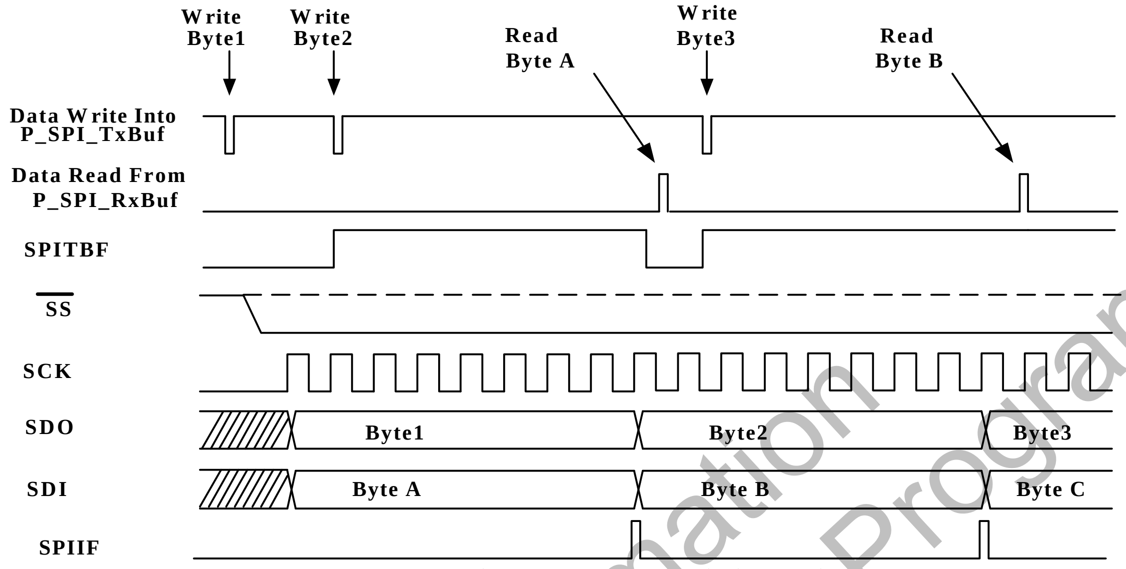
SPI Salve Mode Timing at SPIPHA=0



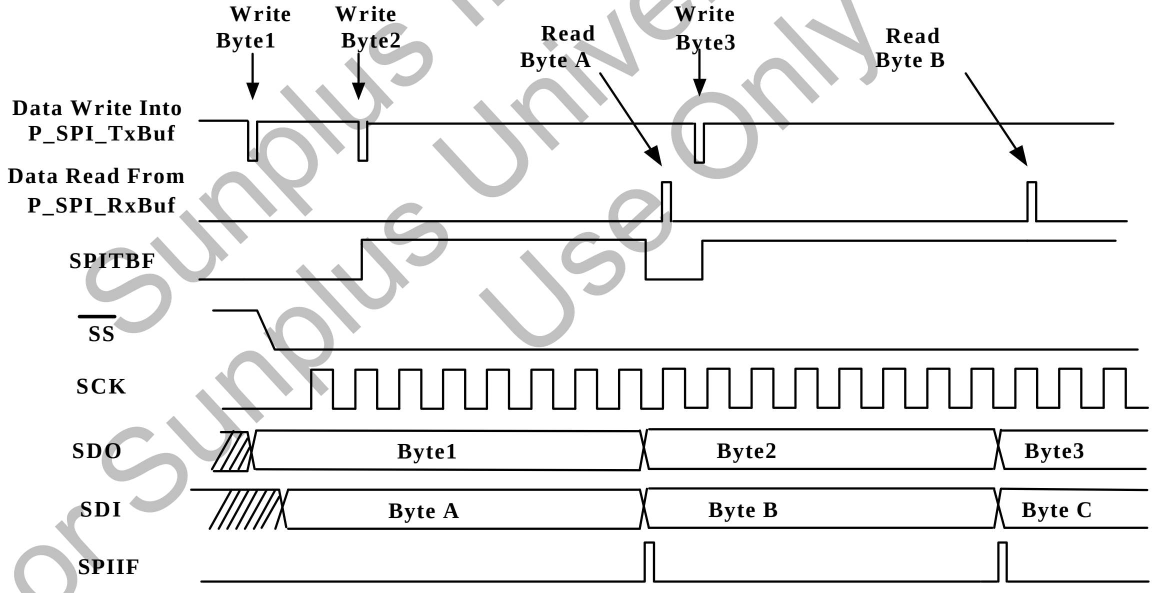
SPI Salve Mode Timing at SPIPHA=1

18.3.3 Consecutive Bytes Transfer

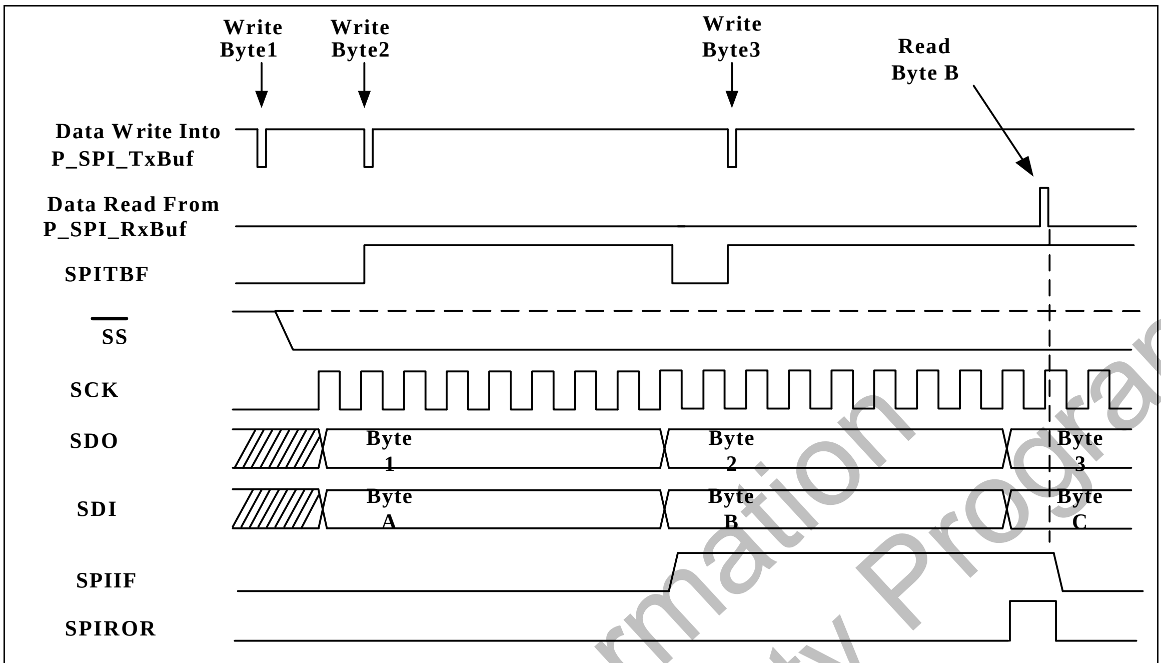
Consecutive bytes transfer is available in both master and slave modes. For transmission, the firmware could send the data consecutively as long as the SPITBF flag is not set. For reception, the firmware should check for overrun error to monitor if there are any missing data due to the polling rate is too low.



Consecutive Bytes Receiving Timing
(Master or Slave Mode in SPIPOL=0,SPIPHA=0)



Consecutive Byte Receiving Timing
(Slave Mode At SPIPOL=0,SPIPHA=1)



Consecutive Byte Receiving Timing with Overrun Error

18.4SPI Interrupt

The SPMC70xxx series provide the SPI interrupt to save CPU on polling SPIIF. The interrupt vector of SPI can be two ways entry. One is IRQ3 and the other is FIQ. The default SPI interrupt is IRQ3, and user can change it as FIQ by **P_INT_Priority Bit 11.** The SPI interrupt enabled by P_SCI_Ctrl's B4 (SPIIE). In master mode,

When the P_SPI_RxBuf received data from slave, the master's P_SPI_Status's SPIIF bit will be enabled and enter the interrupt. And in slave mode, when the P_SPI_RxBuf received data from master or P_SPI_TxBuf send out to master, the slave's P_SPI_Status's SPIIF bit will be set and enter the interrupt. If user doesn't use the SPI interrupt to indicate the date in or out, user must poll the SPIIF bit in program.

18.5Design Tips

18.5.1 SPI Master Mode Example

【Example 16-1】SPI Master Mode Test.

A) Use assemble language

```
        R1 =  0x0102;           // enable SPI master mode,middle smp, phase 1, polarity 0, SPI CPU clock / 16
        [P_SPI_Ctrl] = R1;
        R4 = 0x0000;            //Set transmission initial data
LoopSendSPIR:
        //---Write Command
        R3 = 0x0003;            //Set Read Command
        [P_SPI_TxBuf] = R3;
WaitSPISend1R:
        R2 = [P_SPI_Status];    //Wait Transmission OK
        R2 &= 0x0001;
        jz   WaitSPISend1R
        R2 = [P_SPI_Status];    //Clear SPIIF
        [P_SPI_Status] = R2;

         //---Write Address
        [P_SPI_TxBuf] = R3;
WaitSPISend2R:
        R2 = [P_SPI_Status];    //Wait Transmission OK
        R2 &= 0x0001;
        jz   WaitSPISend2R
        R2 = [P_SPI_Status];    //Clear SPIIF
        [P_SPI_Status] = R2;

         //---Write Data
        [P_SPI_TxBuf] = R4;
WaitSPISend3R:
        R2 = [P_SPI_Status];    //Wait Transmission OK
        R2 &= 0x0001;
        jz   WaitSPISend3R
        R2 = [P_SPI_Status];    //Clear SPIIF
        [P_SPI_Status] = R2;
        R2 = [P_SPI_RxBuf];     //For verify transmission data
        R4 += 1;
        jmp   LoopSendSPIR
```

A) Use C language

```
#include            "SPMC701FM0.h"
#include            "unSPMACRO.h"

main()
{
      unsigned int uiData = 0;
        *P_SPI_Ctrl=C_SPIEN+C_SPIMS_M+C_SPIFS_CLK16;
        while(1)
        {
              Set_SPI_TxBuf(0x0003);                  //---Write Command
           while(!(Get_SPI_Status() & C_SPIIF));
           Set_SPI_Status (Get_SPI_Status());
           Get_SPI_RxBuf();

           Set_SPI_TxBuf(0x0003);                 //---Write Address
           while(!(Get_SPI_Status() & C_SPIIF));
           Set_SPI_Status (Get_SPI_Status());
           Get_SPI_RxBuf();

           Set_SPI_TxBuf(uiData);                 //---Write Data
           while(!(Get_SPI_Status() & C_SPIIF));
           Set_SPI_Status (Get_SPI_Status());
           Get_SPI_RxBuf();

          uiData ++;
          uiData &= 0xFF;
       }
   }
```

18.5.2 SPI Slaver Mode Example

【Example 16-1】 SPI Slave Mode Test.

A) Use assemble language

```
       R1 =    0x0142;               // enable SPI Slave mode,middle smp, phase 0, polarity 0, SPI CPU clock /
16
       [P_SPI_Ctrl] = R1;

    LoopRecevedSPI:
    WaitSPIReceive:
        R2 = [P_SPI_Status];          //Wait Input for checking SPIIF
        R2 &= 0x0001;
        jz WaitSPIReceive
        R2        = [P_SPI_Status];          //Clear SPIIF flag
        [P_SPI_Status] = R2;
        R1 = [P_SPI_RxBuf];              //Get receiving data
        [P_SPI_TxBuf]       = R1;        //Send receiving data to master
        jmp   LoopRecevedSPI
```

A) Use C language

```
#include            "SPMC701FM0.h"
#include            "unSPMACRO.h"

main()
{
    unsigned int uiData;
    *P_SPI_Ctrl = C_SPIEN+C_SPIMS_S+C_SPIFS_CLK16;
    while(1)
    {
            while(!(*P_SPI_Status & C_SPIIF));
           *P_SPI_Status = *P_SPI_Status;
           uiData = *P_SPI_RxBuf;
           *P_SPI_TxBuf = uiData;
     }
}
```

19 Serial Input And Output (SIO)

19.1 Introduction

The SIO supports half-duplex synchronous serial transfer between a Master device and a Slave device. The SIO module of SPMC70xxx series fully implements all slave functions, except general call support. To facilitate firmware implementations of the master functions, the interrupts on start and stop bits are asserted by hardware. Two pins are used for data transfer. These are the SCL pin that is the clock (SCL), and the SDA pin, which is the data (SDA). The two pins shared with GPIO PortA[15:14] as [SCL,SDA], and the two pins are configured as the required attributes (open drain, input/output) automatically when SIO is enabled. The relationship shows bellow Table 17-1.

Table 17-1

PA15	SCL	SIO serial clock pin
PA14	SDA	SIO serial data /address

Note: SPI slave mode and SIO mode cannot be activated at the same time.

The SIO of SPMC70xxx series have following features:

1. Supports Slave 7-bit/10-bit address mode
2. Supports 7-bit/10-bit address Slave mode, with start and stop bit interrupts enabled to support firmware Master mode
3. Supports 7-bit/10-bit address Master mode.

When the MCU set as slave mode, the hardware has to do follow:

1. Detect the START bit
2. Receive the address (7-bit mode or 10-bit mode) (SDA output disabled)
3. Update SIOUA flag for 10-bit mode
4. Compare the received address
5. Receive data when slave is receiver (SDA output disabled)
6. Store address into buffer for firmware reading but doesn't set SIOBF and SIOROR
7. Store data into buffer for firmware reading and set SIOBF and SIOROR
8. Sent acknowledge when address/data byte is received (SDA output enabled)
9. Send data bit when slave is a transmitter (SDA output enable active)
10. Pull SCL low to wait P_SIO_Buf filled when Slave is a transmitter. Master must issues a not-acknowledge before STOP condition.
11. Update the buffer full (SIOBF) flag
12. Detect the overflow condition and Overflow (SIOROR) flag
13. Generate interrupt after transferring

14. Detect the STOP bit

The following table 17-2 shows the condition of SIO ACK signal on the relations of SIO buffer full flag (SIOBF) and SIO overflow flag (SIOROR). When the SIOBF and SIOROR are emptied, the SIO shift register will shift data into SIO buffer.

Table 17-2 Salve Hardware Received Byte Actions In Data Transfer

Status Bits as Data is Received		SIO Shift Register →SIO BUF	Generate ACK	Set bit SIOINT
SIOBF	SIOROR			
0	0	Yes	ACK = 0	Yes
1	0	No	NACK = 1	Yes
1	1	No	NACK = 1	Yes
0	1	No	NACK = 1	Yes

The application of SIO shows in figure 17-1, and when active the SIO module, the SCL and SDA pin will be set as open drain. When using the SIO module, please add the pull high resister on SCL and SDA pin. The range of resister value is from 5K ~ 50K.

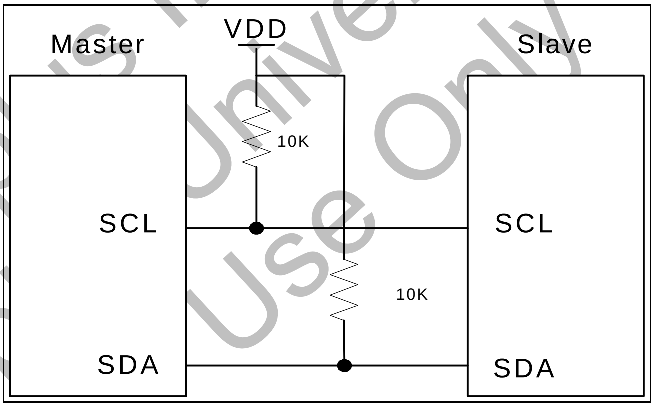


Figure 17-1 Application Circuit for SIO

19.2 Control Register

19.2.1 P_SIO_Ctrl ($70B0): SIO control register

B15	B14	B13	B12	B11	B10	B9	B8
-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-

B7	B6	B5	B4	B3	B2	B1	B0
R/W	R/W	R/W	R/W	-	R/W	R/W	R/W
0	0	0	0	-	0	0	0
SIOEN	SIOSCKOLB	SIOSDAOLB	SIOIEN	-	SIOADRMS	SIOMS	

Bit 15:8 Reserved

Bit 7 SIOEN: SIO enable

1=Enables the serial port and configures the SDA and SCL pins as the source of the serial port pins

0= Disables serial port and configures these pins as I/O port pins

Bit6 SIOSCKOLB: SIO SCL clock output control

1= Release SCL

0= Holds clock low

Bit 5 SIOSDAOLB: SIO SDA release control

1= Release SDA

0= Holds SDA low

Bit 4 SIOIEN: SIO interrupt enable

1= enable

0= disable

Bit 3 Reserved

Bit 2 SIOADRMS: Address mode selection

1= 10-bit address mode

0= 7-bit address mode

Bit 1:0 SIOMS: SIO operation mode selection

00 = Slave mode

01 = Slave mode with start and stop bit interrupts enabled

1X = Firmware controlled master mode, slave idle

19.2.2 P_SIO_Status ($70B1): SIO status register

B15	B14	B13	B12	B11	B10	B9	B8
-	-	-	-	-	-	-	R
-	-	-	-	-	-	-	0
-	-	-	-	-	-	-	NACK

B7	B6	B5	B4	B3	B2	B1	B0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
SIOIF	SIOROR	SIODAT	SIOSTOP	SIOSTRT	SIODIRT	SIOUA	SIOBF

Bit 15:9 Reserved

Bit 8 NACK: SIO Not-acknowledge status

1= The flag is set when Not-Acknowledge conditions occur

The flag is clear if start or stop condition

Bit 7 SIOIF: SIO interrupt flag

1= The flag is set when octet is transferred or both start and stop conditions occur

NOTE: The flag must be cleared by firmware by writing this bit “1”.

Bit6 SIOROR: SIO receive Overrun

1= A byte is received while the SIOBUF register is still holding the previous byte. SIOROR is a “don’t care” in transmit mode. SIOROR must be cleared by software in master or slave mode.

0= No overflow

Bit 5 SIODAT: Data type for selection Data or Address bit

1= Indicates that the last byte received or transmitted was data

0= Indicates that the last byte received or transmitted was address

Bit 4 SIOSTOP: Stop bit

1= Indicates that a stop bit has been detected last (this bit is cleared on RESET or SIO disabled)

0= Stop bit is not detected last

Bit 3 SIOSTRT: Start bit

1= Indicates that a start bit has been detected last (this bit is cleared on RESET or SIO disabled)

0= Start bit is not detected last

Bit 2 SIODIRT: Direction selection for Read or Write information

The bit holds the R/W bit information following the Last address match. This bit is only valid from the address match to the next start bit, stop bit, or not ACK bit

1= Read

0= Write

Bit 1 SIOUA: Update Address (10-bit address mode only)

1 = Indicates that the user needs to update the address in the SIOADDR register

0 = Address does not need to be updated

Bit 0 SIOBF: Buffer Full Status bit

Receive:

1= Receive complete, SIOBUF is full

0= Receive not complete, SIOBUF is empty or is cleared by reading from firmware

Transmit:

1= Transmit in progress, SIOBUF is full

0= Transmit complete, SIOBUF is empty

19.2.3 P_SIO_Buf($70B2): SIO buffer

B15	B14	B13	B12	B11	B10	B9	B8
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0
-							

B7	B6	B5	B4	B3	B2	B1	B0

R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
SIOBUF							

Bit 15~8	reserved
Bit 7~0	SIOBUF

19.2.4 P_SIO_Addr ($70B3): SIO address register

B15	B14	B13	B12	B11	B10	B9	B8
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0
-							

B7	B6	B5	B4	B3	B2	B1	B0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
SIOADR							

Bit 15~8	reserved
Bit 7~0	SIOADR

19.3Operation

19.3.1 Master Mode

When we choice the MCU as the master function, the P_SIO_Ctrl SIOMS bits must be set as master mode. In addition, the Master functions are implemented by firmware. In order to support master function START bit and STOP bit are generated by hardware. The master MCU has two modes, one send address and data to slave, the other send address to slave and get data from slave.

At any mode of master MCU, the address must be send first. The SIOSDAOLB bit of P_SIO_Ctrl must be release. As 7-bit address mode the address must add the 0 (SIODIRT bit) at LSB bit for write address into slave. As 10-bit address mode the address must add 11110 before A9,A8 bit and 0 (SIODIRT bit) at LSB bit for first address write, and the second write have not need add SIODIRT bit and direct send A7~A0. Any send address to slave need to wait slave ACK, if slave send NACK to SDA pin, the P_SIO_Status NACK bit will be set as “High”. In addition, master must send the STOP timing to clear NACK bit and resend again.

When master send data to slave, the SIOSDAOLB bit of P_SIO_Ctrl must be release. As 8-bit data complete send, send STOP bit to end transmission data. When the master receiving data, the SIOSDAOLB bit of P_SIO_Ctrl must be release between first SCL clock and set SIOSDAOLB bit as low to ACK slave. After 8-bit data received, the STOP bit need send by master. The acknowledge bit state is detected to make sure that the addressed slave receives the previous data. If Not-Acknowledge is detected, Not-Acknowledge bit will be set. The firmware can monitor Not-Acknowledge bit to identify not-acknowledge condition and issues a STOP condition.

If user changes the slave mode to master mode, the STOP bit must be set then change to master mode by P_SIO_Ctrl's SIOMS bit.

In master mode, the SCL and SDA lines are manipulated by clearing the corresponding P_SIO_Ctrl 's SIOSCKOLB and SIOSDAOLB bits. In case of no master controlling bus lines, SCL and SDA keep high. First of all, master issues a START conditions, then data transfer by controlling the SCL states (write port to pull-low SCL and release SCL to generate the required clock pulses). Before generating clock pulses, the address or data has to be written into transmission buffer (P_SIO_Buf) first. At the same time, the address or data is also written into transmit shift register. The address/data is shifted out at the falling edge of SCL to keep data valid when SCL is high. The data shifted out is also received in receive shift register. If transmit has been finished, the interrupt will be issued and the data in receive shift register be loaded in P_SIO _Buf. The firmware can compare the data in P_SIO_Buf with the data that is just transmitted. ,By this way, CPU can identify if arbitration condition has occurred and terminate the transmit session.

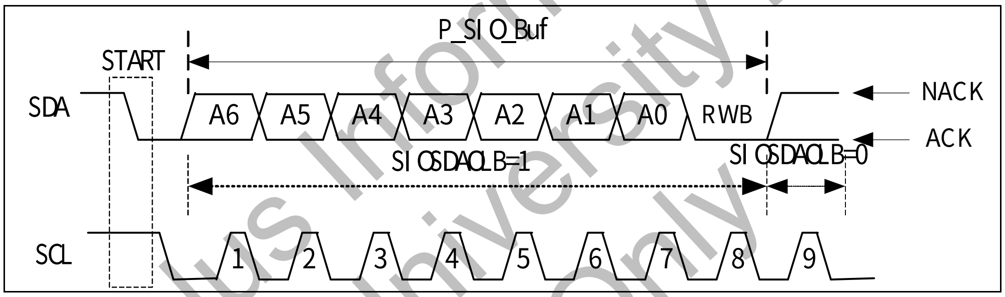


Figure 17-2 7- bit address write timing

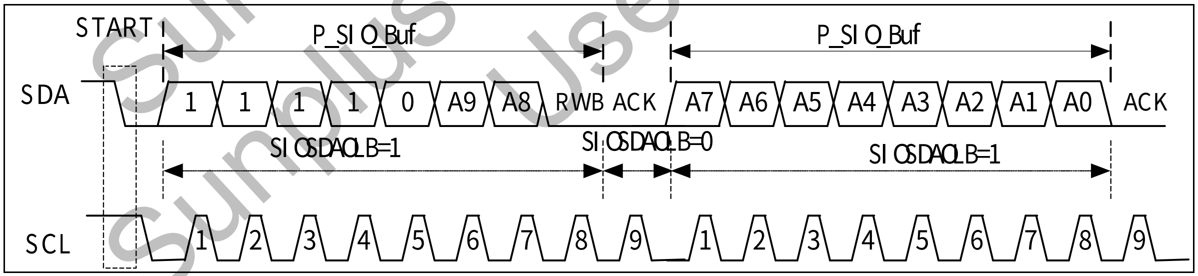
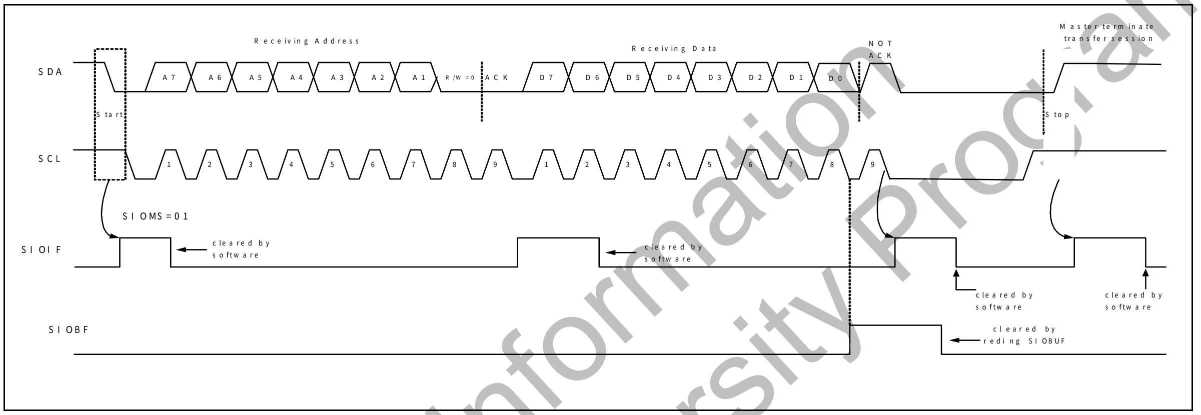


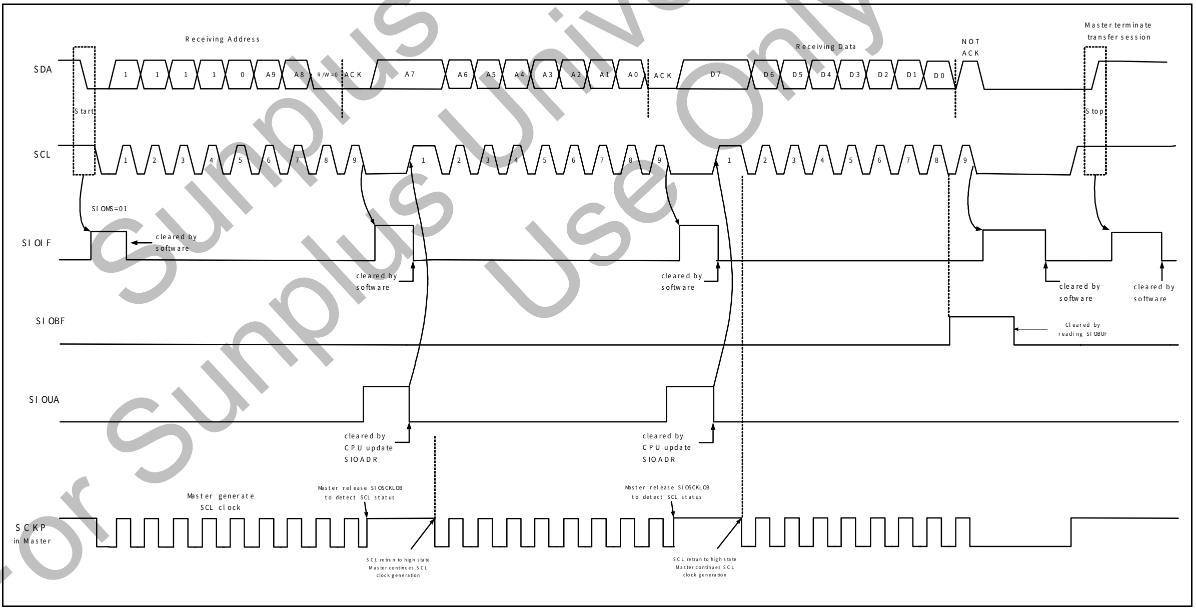
Figure 17-3 10- bit address write timing

19.3.2 Slave Mode

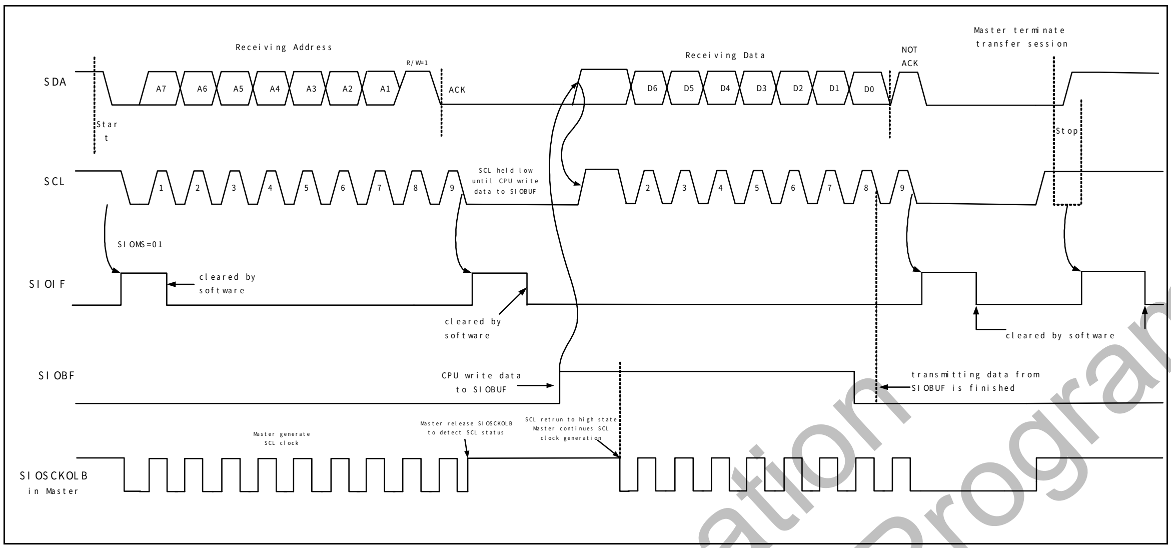
Slave-receiver receives the address, compares the address and response an acknowledgment bit on SDA line, records the 8th bit (read/write) status, and asserts an interrupt to CPU. In case of received address mismatched with P_SIO_ADDR, an interrupt will not be issued. In 10-bit address mode, two address bytes need to be received by the slave. The first five bits of the first address byte specify if this is a 10-bit address. The R/W bit must specify a write so the slave device will receive the second address byte. For a 10-bit address the first byte would equal '1111 0 A9 A8 0', where A9 and A8 are the first two bits of the address. After the first address byte is received, the SIOUA bit in status register and SCL pin is held low until CPU write the second address byte to address register. CPU on Slave can responses the interrupt from SIO module and read the status register and P_SIO_Buf. When the SIODIRT bit of the incoming address byte is set and an address match occurs, the SIODIRT bit in P_SIO_Status is set. Slave operates as a transmitter. The ACK pulse will be sent on the ninth bit, and the SCL pin is held low. The transmitted data must be loaded into the P_SIO_BUF register, then SCL pin is released by slave. The master should monitor the SCL pin prior to asserting another clock pulse. The slave devices may be holding off the master by stretching the SCL clock.



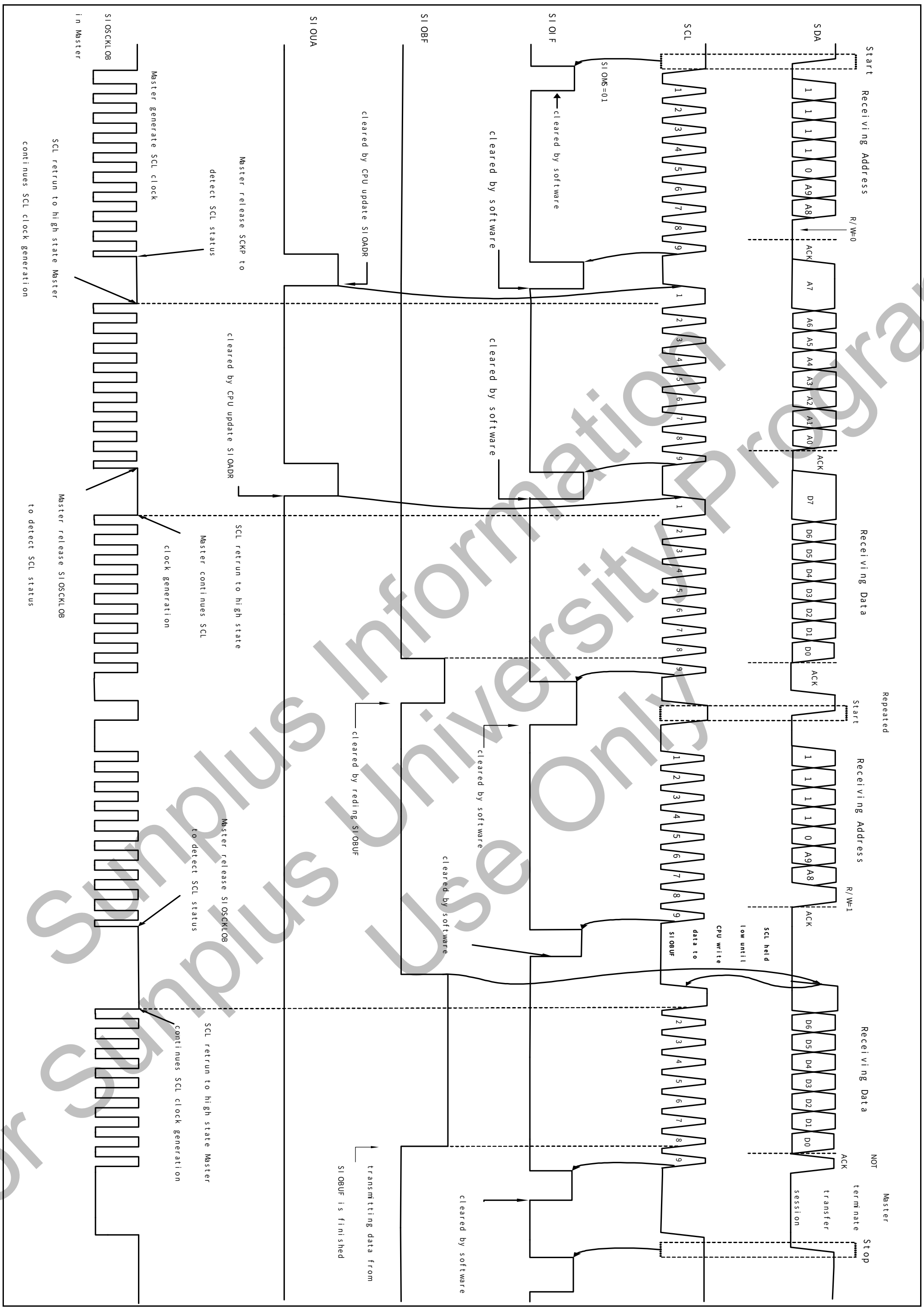
SIO waveforms reception for Slave mode (7-bit Address)


Receiving Address
Receiving Data
NOT ACK
Master terminate transfer session
SDA
A9
A8
R/W=0
ACK
A7
A6
A5
A4
A3
A2
A1
A0
D7
D6
D5
D4
D3
D2
D1
D0
Start
Stop
SCL
SIOMS=01
SIOIF
cleared by software
SIOBF
Cleared by reading SIOBUF
SIOUA
cleared by CPU update SIOADR
Master generate SCL clock
Master release SIOSCKLOB to detect SCL status
SCL return to high state Master continues SCL clock generation
SCKP in Master

SIO waveforms reception for Slave mode (10-bit Address)


Receiving Address
R/W=1
Receiving Data
NOT ACK
Master terminate transfer session
SDA
A7
A6
A5
A4
A3
A2
A1
ACK
D6
D5
D4
D3
D2
D1
D0
Start
Stop
SCL
SCL held low until CPU write data to SIOBUF
SIOMS=01
SIOIF
cleared by software
cleared by software
cleared by software
SIOBF
CPU write data to SIOBUF
transmitting data from SIOBUF is finished
Master generate SCL clock
Master release SIOSCKOLB to detect SCL status
SCL retrun to high state Master continues SCL clock generation
SIOSCKOLB in Master

SIO waveforms transmission for Slave mode (7-bit Address)


SDA
SCL
SIOIF
SIOBF
SIOUA
SIOSCKLOB in Master
Start
Receiving Address
1 1 1 1 0 A9 A8
R/W=0
ACK
A7 A6 A5 A4 A3 A2 A1 A0
ACK
D7 D6 D5 D4 D3 D2 D1 D0
Receiving Data
ACK
Repeated Start
Receiving Address
R/W=1
ACK
Receiving Data
NOT ACK
Master terminate transfer session
Stop
SIOMS=01
cleared by software
cleared by CPU update SIOADR
Master release SCKP to detect SCL status
Master generate SCL clock
SCL retrun to high state Master continues SCL clock generation
Master release SIOSCKLOB to detect SCL status
SCL retrun to high state
Master continues SCL clock generation
cleared by reding SIOBUF
SCL held low until CPU write data to SIOBUF
transmitting data from SIOBUF is finished

SIO waveforms transmission for Slave mode (10-bit Address)

19.4SIO Interrupt

The SPMC70xxx provides an SIO interrupt to save CPU on polling SIOINT or START and STOP bits. The interrupt vector of SIO can be two ways entry. One is IRQ3 and the other is FIQ. The default SIO interrupt is IRQ3, and user can change it as FIQ by P_INT_Priority Bit 10. The SIO interrupt enabled by P_SIO_Ctrl's B4 (SIOIE). In master mode, when the P_SIO_Buf transmit or received data to or from slave, the master's P_SIO_Status's SIOIINT bit will be enabled and enter the interrupt. In addition, there are two modes can be selected in slave mode, which can be set by P_SIO_Ctrl SIOMOD bits. One is when the P_SIO_Buf received or transmit data from or to master, the slave's P_SIO_Status's SIOIF bit will be enable and enter the interrupt. The other is when receive the START or STOP bit from master.

19.5Design Tips

The SIO interface of SPMC70xxx series is a software control interface. User must control the start, stop and SCK clock toggled by software.

? **Example17-1? :** This example control the standard EEPROM(AT24C04) with SPMC701FM0 SIO.

A) Use assemble language

```
  .include SPMC701FM0.inc

  .RAM
  T_DataBuff: .DW 100 DUP(0)        //for check the the data reading from EEPROM is correct

  .CODE
  .public _main
  _main:
             //Marster write Data to EEPROM
             R3 = 0;         // R3 --- Byte Address
             R4 = 0;          // R4 --- Data
       ?L_LoopWrite:
             CALL F_MasterWriteData;
             R4 += 2;
             R3 += 1;
             CMP r3,40
             JBE ?L_LoopWrite

             //Marster read Data from EEPROM
             R3 = 0;
       ?L_LoopRead:
             CALL F_MasterReadData;
             R1 = T_DataBuff;
             R1 += R3;
             [R1] = R4;
             R3 += 1;
             CMP r3,40
             JBE ?L_LoopRead
       L_MainLoop:
                   NOP
                    JMP    L_MainLoop

   //=====================================================================
   //-- Function: F_MasterWriteData
   //-- Description:Master write Data to SIO EEPROM
   //-- Syntax: CALL F_MasterWriteData
   //-- Parameter: R3 --- Byte Address
   //                 R4 --- Data
   //-- Return: None
   //-- Register:R1,R2
   //=====================================================================
   .PUBLIC F_MasterWriteData
   F_MasterWriteData:
             // Enable SIO, Master 7-bit mode, Disable INT,SCL,SDA Release
             R1 = (C_SIOEN+C_SIOSCKOLB1+C_SIOSDAOLB1+C_SIOADRMS7+C_SIOMS_M);
             [P_SIO_Ctrl] = R1;

             // Set Start Signal Produce
             CALL F_SendStartSignal

             //Send Device Address
             R1 = 0x00A0;              //Set Device Address 0xA0(10100000B) to slave
             [P_SIO_Buf] = R1;
             CALL   F_Send9WrClock
             CALL F_WaitINTFlag
             CALL F_CheckACK;

              //Write Byte address
             [P_SIO_Buf] = R3;          // Write word address to slave
             CALL F_Send9WrClock
              CALL F_WaitINTFlag
              CALL F_CheckACK;

              //Write Data
             [P_SIO_Buf] = R4;          // write Data to SIO buffer
             CALL F_Send9WrClock
             CALL F_WaitINTFlag
             CALL F_CheckACK;

             // Set Stop Signal Produce
             CALL F_SendStopSignal
             CALL F_SIOWaitTwClyle;          //Write Cycle time
```

```
          RETF

//=========================================================================
//-- Function: F_MasterReadData
//-- Description:Master read Data From SIO EEPROM with random read mode
//-- Syntax: CALL F_MasterReadData
//-- Parameter: R3 --- Byte Address
//-- Return:     R4 --- Data
//-- Register:R1,R2,R4
//=========================================================================
.PUBLIC F_MasterReadData
F_MasterReadData:
          // Enable SIO, Master 7-bit mode, Disable INT,SCL,SDA Release
         R1 = (C_SIOEN+C_SIOSCKOLB1+C_SIOSDAOLB1+C_SIOADRMS7+C_SIOMS_M);
          [P_SIO_Ctrl] = R1;

          // Set Start Signal Produce
          CALL    F_SendStartSignal

          //Send Device Address
          R1 = 0x00A0;                  //Set Device Address 0xA0(10100000B) to slave
          [P_SIO_Buf] = R1;
          CALL    F_Send9WrClock;
          CALL    F_WaitINTFlag;
          CALL    F_CheckACK;

           //Write word address
          [P_SIO_Buf] = R3;        //Write word address to slave
          CALL    F_Send9WrClock;
          CALL    F_WaitINTFlag;
          CALL    F_CheckACK;

          //Write Read Command(Device Address)
          CALL F_SendStartSignal
          R1 = 0x00A1;             //Set Device Address 0xA1(10100001B) to slave as Read
command
          [P_SIO_Buf] = R1;
          CALL F_Send9WrClock;
          CALL F_WaitINTFlag;
          CALL F_CheckACK;

           // read data from slave
          CALL F_Send9RdClock
?L_WaitReceiveComplete:
          R2 = [P_SIO_Status];  //read SIOBF status
          TEST R2,0x0001
          JZ    ?L_WaitReceiveComplete;
          R4    = [P_SIO_Buf];          //read buffer

          CALL F_SendStopSignal    //Set Stop Signal Produce
          RETF


/////////////////////////////////////////////////////////////
F_SendStartSignal:
```

```
            R1 = [P_SIO_Ctrl];
            R1 |= 0x0060;           // SCL=release SDA=release
            [P_SIO_Ctrl] = R1;
            CALL    F_DelayClk;

            R1 &= 0x00DF;                   // SCL=release SDA=low
            [P_SIO_Ctrl] = R1;
            CALL    F_DelayClk;

            R1 &= 0x009F;          // SCL=low    SDA=low
            [P_SIO_Ctrl] = R1;
            CALL    F_DelayClk;

            R1 |= 0x0020;           // SCL=low SDA=release
            [P_SIO_Ctrl] = R1;
            RETF;


//////////////////////////////////////////////////////////////////
F_SendStopSignal:
            R1 = [P_SIO_Ctrl];
            R1 &= 0x009F;           // SCL=low    SDA=low
            [P_SIO_Ctrl] = R1;
            CALL    F_DelayClk;
            R1 |= 0x0040;           // SCL=release SDA=low
            [P_SIO_Ctrl] = R1;
            CALL    F_DelayClk;
            R1 |= 0x0060;           // SCL=release SDA=release
            [P_SIO_Ctrl] = R1;
            RETF;


//////////////////////////////////////////////////////////////////
F_Send9WrClock:
            R2 = 0x0009;
            R1 = [P_SIO_Ctrl];
?L_Loop9WrClock:
            R1 |= 0x0060;           // SCL=Release         SDA=Release
            [P_SIO_Ctrl]    = R1;
            CALL    F_DelayClk;
            R1 &= 0x00BF;           // SCL=Low       SDA=Release
            [P_SIO_Ctrl] = R1;
            CALL    F_DelayClk;
            R2 -=        0x0001;
            JNZ    ?L_Loop9WrClock;
            RETF;


//////////////////////////////////////////////////////////////////
F_Send9RdClock:
            R2 = 0x0008;
            R1 = [P_SIO_Ctrl];
?L_Loop8RdClock:
            R1 |= 0x0060;           // SCL=Release         SDA=Release
            [P_SIO_Ctrl] = R1;
            CALL    F_DelayClk;
```

```
        R1 &= 0x00BF;       // SCL=Low      SDA=Release
        [P_SIO_Ctrl] = R1;
        CALL   F_DelayClk;
        R2 -= 0x0001;
        JNZ   ?L_Loop8RdClock;
        //Marster read operature does not need ACK signal
        RETF;


////////////////////////////////////////////////////////////////
F_WaitINTFlag:
        R1 = [P_SIO_Status];        // Read INT status
        TEST R1,0x0080;
        JZ F_WaitINTFlag
        [P_SIO_Status] = R1;
        RETF;


////////////////////////////////////////////////////////////////
F_CheckACK:
         // Read nack status to check if not-ACK occur
        R1 = [P_SIO_Status];
        TEST R1,0x0100
        JZ ?L_RightSIOACK

        // Produce stop condition
        R1 = 0x0092;        // SCL=low SDA=low
        [P_SIO_Ctrl] = R1;
        R1 = 0x00d2;        // SCL=release SDA=low
        [P_SIO_Ctrl] = R1;
        R1 = 0x00f2;        // release SCL/SDA
        [P_SIO_Ctrl] = R1;
?L_RightSIOACK:
         RETF


////////////////////////////////////////////////////////////////
F_DelayClk:
      PUSH R1 TO [SP]
      R1 = 8;
?L_DelayClkLoop:
      R1 -= 1;
      JNZ ?L_DelayClkLoop
      POP R1 FROM [SP]
      RETF


 ////////////////////////////////////////////////////////////////
F_SIOWaitTwClyle:
        R2 = 0x2A00;
?L_WaitTwLoop:
        R2 -= 1;
        JNZ   ?L_WaitTwLoop;
        RETF;
```

```
////////////////////////////////////////////////////////////////
```

B) Use C language

```
//========================= mian.C ==================================//
   #include   "SPMC701FM0.H"
   #include   "unSPMACRO.H"


   extern void SP_MasterWriteData(unsigned int,unsigned int);
   extern unsigned int SP_MasterReadData(unsigned int);

   unsigned int uiTempData[20];

   main()
   {
        unsigned int uiAddr,uiData=0;
        {
            for(uiAddr=0;uiAddr<30;uiAddr++,uiData++)
            SP_MasterWriteData(uiAddr,uiData);

            for(uiAddr=0;uiAddr<30;uiAddr++)
            uiTempData[uiAddr] = SP_MasterReadData(uiAddr);

        }
        while(1);
   }

   //========================= SIODrive.C =================================//
   ////////////////////////////////////////////////////////////////
   void SP_SIOWaitTwClyle()
   {
        unsigned int i;
        for(i=0;i<0x2A00;i++) ;
   }


   ////////////////////////////////////////////////////////////////
   void SP_DelayClk()
   {
        unsigned int i;
        for(i=0;i<0x8;i++) ;
   }


   ////////////////////////////////////////////////////////////////
   void SP_SendStartSignal()
   {
        *P_SIO_Ctrl = *P_SIO_Ctrl | 0x0060;    // SCL=release SDA=release
        SP_DelayClk();

        *P_SIO_Ctrl = *P_SIO_Ctrl & 0x00DF;    // SCL=release SDA=low
        SP_DelayClk();

        *P_SIO_Ctrl = *P_SIO_Ctrl & 0x009F;    // SCL=low    SDA=low
        SP_DelayClk();
        *P_SIO_Ctrl = *P_SIO_Ctrl | 0x0020; // SCL=low SDA=release
        SP_DelayClk();
}


////////////////////////////////////////////////////////////////////
void SP_SendStopSignal()
{
        *P_SIO_Ctrl = *P_SIO_Ctrl & 0x009F;            // SCL=low    SDA=low
        SP_DelayClk();
        *P_SIO_Ctrl = *P_SIO_Ctrl | 0x0040;            // SCL=release SDA=low
        SP_DelayClk();

        *P_SIO_Ctrl = *P_SIO_Ctrl | 0x0060;            // SCL=release SDA=release
        SP_DelayClk();
}


////////////////////////////////////////////////////////////////////
void SP_Send9WrClock()
{
        unsigned int i,uiTemp1;
        uiTemp1 = *P_SIO_Ctrl;
        for(i=9;i!=0;i--)
        {
                *P_SIO_Ctrl = uiTemp1 | 0x0060;        // SCL=release SDA=release
                SP_DelayClk();
                *P_SIO_Ctrl = uiTemp1 & 0x00BF;        // SCL=Low SDA=Release
                SP_DelayClk();
        }
        //wait Transmit interrupt
        while(!(*P_SIO_Status & C_SIOIF)) ;
        *P_SIO_Status = *P_SIO_Buf;

        // Read nack status to check if not-ACK occur
        if((*P_SIO_Status & C_NACK) != 0)
        SP_SendStopSignal();
}


////////////////////////////////////////////////////////////////////
void SP_Send9RdClock()
{
        unsigned int i,uiTemp2;
        uiTemp2 = *P_SIO_Ctrl;
        for(i=8;i!=0;i--)
        {
                *P_SIO_Ctrl = uiTemp2 | 0x0060;        // SCL=release SDA=release
                SP_DelayClk();
                *P_SIO_Ctrl = uiTemp2 & 0x00BF;        // SCL=Low SDA=Release
                SP_DelayClk();
        }
        //Marster read operature does not need ACK signal
}
```

```
//======================================================================
//-- Function: SP_MasterWriteData
//-- Description:Master write Data to SIO EEPROM
//-- Syntax: void SP_MasterWriteData(unsigned int,unsigned int)
//-- Parameter: R3 --- Byte Address
//                    R4 --- Data
//-- Return: None
//-- Register:
//======================================================================
void SP_MasterWriteData(unsigned int uiAddr,unsigned int uiData)
{
      // Enable SIO, Master 7-bit mode, Disable INT,SCL,SDA Release
      *P_SIO_Ctrl = C_SIOEN+C_SIOSCKOLB1+C_SIOSDAOLB1+C_SIOADRMS7+C_SIOMS_M;

      // Set Start Signal Produce
      SP_SendStartSignal();

      //Send Device Address
      *P_SIO_Buf = 0xA0;         //Set Device Address 0xA0(10100000B) to slave
      SP_Send9WrClock();

      //Write Byte address
      *P_SIO_Buf = uiAddr;        //Set Device Address 0xA0(10100000B) to slave
      SP_Send9WrClock();

      //Write Data
      *P_SIO_Buf = uiData;        //Set Device Address 0xA0(10100000B) to slave
      SP_Send9WrClock();

      // Set Stop Signal Produce
      SP_SendStopSignal();
      SP_SIOWaitTwClyle();     //Write Cycle time
}


//======================================================================
//-- Function: SP_MasterReadData
//-- Description:Master read Data From SIO EEPROM with random read mode
//-- Syntax: unsigned int SP_MasterReadData(unsigned int);
//-- Parameter: R3 --- Byte Address
//-- Return:       R4 --- Data
//-- Register:
//======================================================================
unsigned int SP_MasterReadData(unsigned int uiAddr)
{
      unsigned int uiTemp;
      // Enable SIO, Master 7-bit mode, Disable INT,SCL,SDA Release
      *P_SIO_Ctrl = C_SIOEN+C_SIOSCKOLB1+C_SIOSDAOLB1+C_SIOADRMS7+C_SIOMS_M;

      // Set Start Signal Produce
      SP_SendStartSignal();

      //Send Device Address
      *P_SIO_Buf = 0xA0;         //Set Device Address 0xA0(10100000B) to slave
```

```
        SP_Send9WrClock();

        //Write Byte address
        *P_SIO_Buf = uiAddr;          //Set Device Address 0xA0(10100000B) to slave
        SP_Send9WrClock();

        //Write Read Command(Device Address)
        SP_SendStartSignal();
        *P_SIO_Buf = 0x00A1;    //Set Device Address 0xA1(10100001B) to slave as Read command
        SP_Send9WrClock();

        // read data from slave
        SP_Send9RdClock();
        while(!(*P_SIO_Status & C_SIOBF)) ;
        uiTemp = *P_SIO_Buf;

        // Set Stop Signal Produce
        SP_SendStopSignal();
        return (uiTemp);
    }
```

20 Universal Asynchronous Receiver/Transmitter - UART

20.1 Introduction

The SPMC70xxx equips a Universal Asynchronous Receiver/Transmitter for exchange data. The UART block provides full-duplex asynchronous serial communication with other devices. With this interface, SPMC70xxx can transmit and receive simultaneously. The baud rate can be programmed from 1200 bps up to 115200bps. The TXD (transmitting) and RXD (receiving) of UART are shared with PA13 and PA12, respectively.

PA13	TXD	UART Transmission
PA12	RXD	UART Receiving

The basic features listing at follow:

1. Provides standard asynchronous, full-duplex communication.
2. Supports 10-bit (start + 8bit data + stop) or 11-bit (start + 8-bit data + a programmable 9th bit + stop) modes.
3. Programmable baud rate from 1200 bps to 115200 bps.
4. Interrupt occurs after an octet is received.
5. Interrupt occurs after an octet transmission is completed.
6. High noise rejection for bit reception (majority decision of 3 consecutive samples in the middle of received bit time).
7. Supports multiprocessor communications in 11-bit mode

The UART transmits data on the TXD pin in the following sequence: start bit, 8 data bits (LSB first), 9th bit (11-bit mode only), and stop bit. The timing shows in figure 18-1.

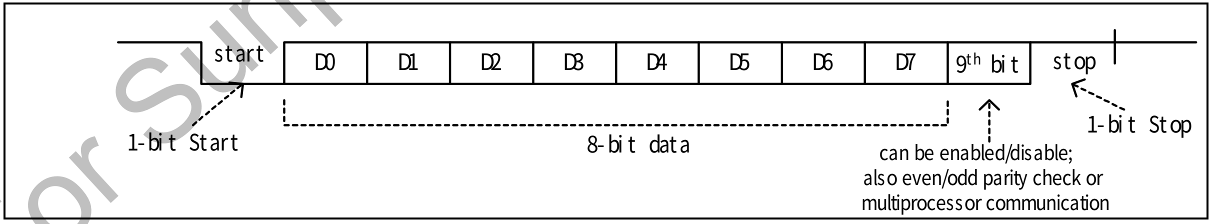

start
D0
D1
D2
D3
D4
D5
D6
D7
9th bit
stop
1-bit Start
8-bit data
can be enabled/disable; also even/odd parity check or multiprocessor communication
1-bit Stop

Figure 18-1

Reception begins at the falling edge of a start bit received on RXD pin when UART function is enabled. For this purpose, RXD is sampled 16 times per bit in any baud rate. For noise rejection, the serial port establishes the content of each received bit by a majority decision of three consecutive samples in the middle of each bit time. This is especially true for the start bit. If the falling edge on RXD is not verified by a majority decision of three consecutive samples (low), the serial port stops reception and waits for another falling edge on RXD. Moreover, UART is capable of performing parity check and multiprocessor communication in 11-bit mode. A typical use of the multiprocessor communication feature is the situation when a master intends to send a block of data to one of several slaves.

The UART begins transmitting after the first rollover in the divided-by-16 counter, after the software writes to the P_UART_TxBuf register. The UART transmits data on the TXD pin in the following order: start bit, 8 data bits (LSB first), parity bit (11bit mode only) or UARTBMPT, and stop bit. The UARTTIF bit is set cycles after the stop bit is transmitted.

At the stop bit time, the serial port checks for the following conditions: UARTMPEN = 1, UARTBMPT bit is "1" or UARTMPEN = 0

If the above conditions are met, the serial port writes the received byte to the P_UART_TxBuf register; loads the 9th bit into UARTBMPT, and set the UARTRIF bit. If the above conditions are not satisfied, the received data is lost; the UARTRBUF and UARTBMPT bit are not loaded; the UARTRIF bit is not set. After stop bit, the serial port waits for another high-to-low transition on the RXD pin.

The UART interface normal used to communicate with PC computer. The following figure 18-2 show the application circuit between PC and SPMC70xxx MCU by using ICL232 chip.

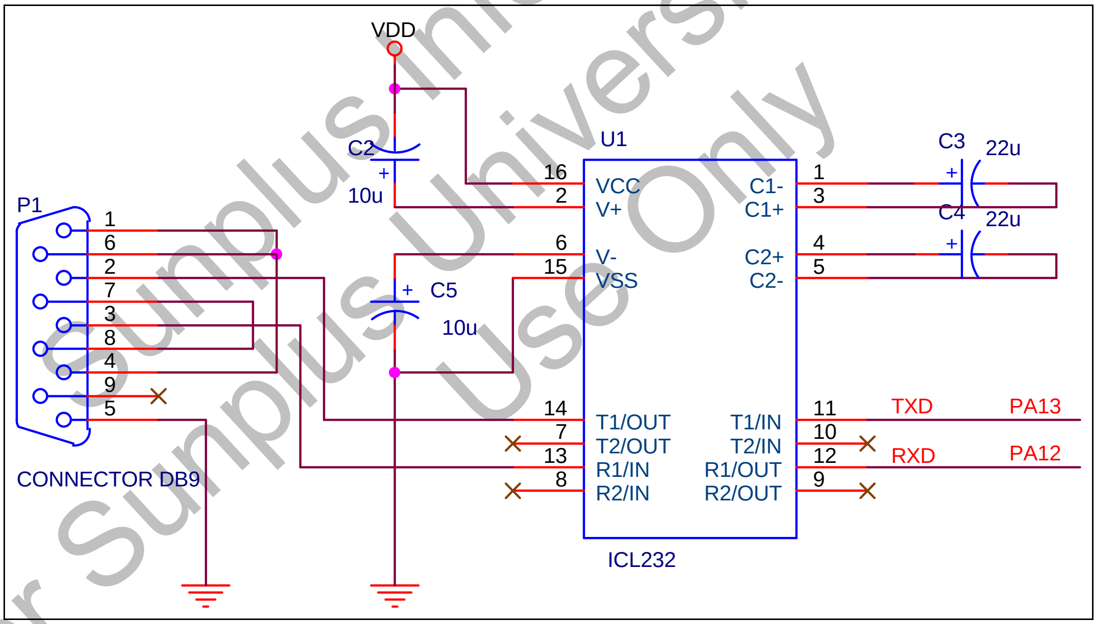


Figure 18-2 Application Circuit

20.2 Control Register

20.2.1 P_UART_Ctrl ($70A0): UART control register

B15	B14	B13	B12	B11	B10	B9	B8
-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-

B7	B6	B5	B4	B3	B2	B1	B0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

0	0	0	0	0	0	0	0
UARTTEN	UARTREN	UARTMS	UARTMPEN	UARTPEN	UARTPMS	UARTTIEN	UARTRIEN

B15~B8	Reserve
Bit 7	UARTTEN: UART transmission enable 1= Enable 0= Disable
Bit6	UARTREN: UART receiving enable 1= Enable 0= Disable
Bit 5	UARTMS: UART mode selection 1= 11-bit mode, Start+8bit data+9th data bit + Stop 0= 10-bit mode, Start+8bit data + Stop
Bit 4	UARTMPEN: UART multiprocessor communication enable / disable 1= multiprocessor communication enable 0= multiprocessor communication disable
Bit 3	UARTPEN: Parity check enable, the 9th bit is parity 1= Enable 0= Disable
Bit 2	UARTPMS: Parity mode selection, if Parity check is enabled this bit indicates parity type 1= Even 0= Odd Note: If Parity check is enabled this bit is the 9th data bit to be transmitted
Bit 1	UARTTIEN: UART transmission interrupt enable 1= Enable 0= Disable
Bit 0	UARTRIEN: UART receiving interrupt enable 1= Enable 0= Disable

20.2.2 P_UART_Status ($70A1): UART status register

B15	B14	B13	B12	B11	B10	B9	B8
-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-

B7	B6	B5	B4	B3	B2	B1	B0
R	R	R	R	R	R	R/W	R/W
0	0	0	0	0	0	0	0
UARTRBF	UARTTBY	UARTBMPT	UARTROR	UARTFE	UARTPE	UARTTIF	UARTRIF

Bit 15~8 Reserved

Bit 7 UARTRBF: Receive Buffer full

1= RX buffer is full

0= RX buffer is empty

Bit 6 UARTTBY: Transmitter is busy

1= Transmitter is busy

0= Transmitter is idle

Bit 5 UARTBMPT: The state of bit 9 in 11-bit mode

1= the received 9th data bit is 1

0= the received 9th data bit is 0

Bit 4 UARTROR: Receive overrun Error,

1= Overrun error occurs

0= No overrun error occurs

Bit 3 UARTFE : Frame Error, stop bit is not detected

1= Frame error occurs

0= No frame error occurs

Bit 2 UARTPE: Parity Error flag

1= Parity error occurs

0= No parity error occurs

Bit 1 UARTTIF: UART transmission interrupt flag

1= a byte transmission complete, an interrupt is asserted if UARTTIEN is set as 1

0= No transmission interrupt event

Bit 0 UARTRIF: UART receiving interrupt flag

1= a valid byte receiving complete, an interrupt is asserted if UARTRIEN is set as 1

0= No receiving interrupt

The bit is set when:

a. an octet is received in 10-bit mode, or

b. an octet is received in 11-bit mode and UARTMPEN=0, or

c. an octet is received in 11-bit mode and UARTMPEN=1 and UARTBMPT=1 (address byte)

20.2.3 P_UART_Reset ($70A2): UART software reset

B15	B14	B13	B12	B11	B10	B9	B8
W	W	W	W	W	W	W	W
0	1	0	1	0	1	0	1
-	-	-	-	-	-	-	-

B7	B6	B5	B4	B3	B2	B1	B0
W	W	W	W	W	W	W	W
0	1	0	1	0	1	0	1
UARTRST							

Write this port to generate synchronous reset of UART module.

20.2.4 P_UART_BaudRateL ($70A3): Baud rate scale low byte register

B15	B14	B13	B12	B11	B10	B9	B8
-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-

B7	B6	B5	B4	B3	B2	B1	B0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
UARTBRSL							

Bit 15~8	reserved
Bit 7~0	UARTBRSL: Baud rate scale low byte

20.2.5 P_UART_BaudRateH ($70A4) Baud rate scale high byte register

B15	B14	B13	B12	B11	B10	B9	B8
-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-

B7	B6	B5	B4	B3	B2	B1	B0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
UARTBRSH							

Bit 15~8	reserved
Bit 7~0	UARTBRSH: Baud rate scale high byte

20.2.6 P_UART_TxBuf ($70A5):UART transmission buffer

B15	B14	B13	B12	B11	B10	B9	B8
-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-

B7	B6	B5	B4	B3	B2	B1	B0
W	W	W	W	W	W	W	W
0	0	0	0	0	0	0	0
UARTTXBUF							

Bit 15~8	reserved
Bit 7~0	UARTTXBUF: Transmission buffer

20.2.7 P_UART_RxBuf ($70A6): UART receive buffer

B15	B14	B13	B12	B11	B10	B9	B8
-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-

B7	B6	B5	B4	B3	B2	B1	B0

R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0
UARTRXBUF							

Bit 15~8	reserved
Bit 7~0	UARTRXBUF: Receiving buffer

20.3Operation

There exists a baud rate register and a 16-bit timer to generate the baud rate. Each times the timer increments from its maximum count (0xFFFF), a clock is sent to the baud rate circuit. The clock is through divid-by-16 counter to generate the baud rate. The timer is reloaded automatically the value in baud rate register.

Baud Rate = 24MHz / [16 x (65536 – baud rate register)]

The content in baud rate register is taken as a 16-bit unsigned number..

To derive the required baud rate register values from a known baud rate, use the equation:

Baud Rate Scale Register = 65536 – 24MHz / (16 x Baud Rate)

Baud Rate	Baud Rate Timer Reload Register Value @ 24MHz
115200 bps	FFF3
57600 bps	FFE6
19200 bps	FFB2
9600 bps	FF64
4800 bps	FEC8
2400 bps	FD8F
1200 bps	FB1E

(Note: The UART clock input is always 24MHz.)

The 11-bit mode also can be separated to two conditions: parity check mode and multiprocessor mode. In parity mode, the P_UART_Ctrl UARTMPEN bit must set multiprocessor communication to be disabled and it must enable the UARTPEN bit. In multiprocessor mode, the UARTMPEN bit of P_UART_Ctrl, multiprocessor communication, must be enabled and the UARTPEN bit must be disable at the same time.

The multiprocessor communication feature is enabled in 11-bit mode when UARTMPEN bit is set in UART control register (P_UART_Ctrl). In multiprocessor communication mode, the received 9th bit is stored in UARTBMPT and, after stop bit is received, the UART interrupt is activated if UARTBMPT = 1.

A typical use of the multiprocessor communication is in the condition when a master intends to send a block of data to one of several slaves. When master first transmits an address byte, the master sets the 9th bit to "1". In addition, when master sends the data, the master sets the 9th to 0 for data bytes.

When UARTMPEN = 1, no slave will be interrupted by a data byte. However, an address byte interrupts all slaves so that each slave can examine the received address byte to determine whenever that slave is being addressed. Address decoding must be done by software during the interrupt service routine. The addressed slave clears its UARTMPEN bit preparing to receive the data bytes. For slaves not addressed, it leaves the UARTMPEN bit set and ignores the incoming data bytes.

The figure 18-3 shows the multiprocessor block.

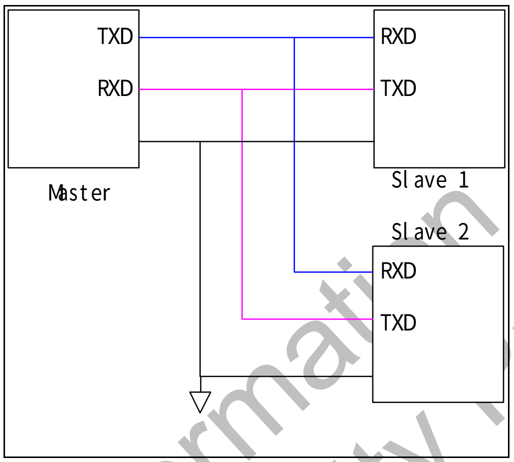


Figure 18-3 Multiprocessor Block

Keep the following 4 steps to control multiprocessors.

Step 1: Set master and slave as 11-bit and multiprocessor communication enable

Step 2: Master sends address data + address bit (PARITY)

Step 3: Slave checks address data; if address matches slave address, selected salve device changes UARTMPEN as multiprocessor communication disable.

Step 4: Master sends data + data bit (PARITY); the only slave changing to UARTMPEN = 0 will receive the data.

20.4UART Interrupt

The SPMC70xxx series provide the UART interrupt to save CPU on polling UART transmission or receiving interrupt flag (UARTTIF or UARTRIF) bits. The interrupt vector of UART can be two ways entry. One is IRQ3 and the other is FIQ. The default UART interrupt is IRQ3, and user can change it as FIQ by P_INT_Priority Bit 12. The UART transmission and receiving interrupt enabled by P_SIO_Ctrl's B1 (UARTTIEN) and B0 (UARTRIEN). In transmission mode, when the P_UART_TxBuf transmit data to other device, the P_UART_Status UARTTIF bit will be enabled and enter the interrupt. In receiving mode, when P_UART_RxBuf receive data from other device, the P_UART_Status UARTRIF bit will be enable and enter the interrupt.

20.5Design Tips

? **Example18-1**? UART transmission test(10 bites), The following example shows how to set UART transmission and set data out from TX pin.

A) Use assemble language

```
        R1 = 0x0080;            //UART Tx,10-bit,disable interrupt
        [P_UART_Ctrl] = R1;

        R1 = 0x00FE;            // baudrate 4800 : 0xFEC8
        [P_UART_BaudrateH] = R1;
        R1 = 0x00C8;
        [P_UART_BaudrateL] = R1;

        R1 = 0x0000;
L_SendTxData:
        [P_UART_TxBuf] = R1;
L_WaitTxReady:                  //wait send complete
        R2 = [P_UART_Status];
        R2 &= 0x0002;
        jz   L_WaitTxReady
        R2 = [P_UART_Status];
        [P_UART_Status] = R2;

        R1 += 0x0001;
        R1 &= 0x00FF
        jmp   L_SendTxData
```

B) Use C language

```
    #include      "SPMC701FM0.H"
    #include      "unSPMACRO.H"

    main()
    {
        unsigned int iData = 0;
        Set_UART_Ctrl(C_UARTTEN+C_UARTMS_10bit+C_UARTRDIS);
        Set_UART_BaudrateL(C_UARTBRSL_4800);
        Set_UART_BaudrateH(C_UARTBRSH_4800);
        while(1)
        {
             Set_UART_TxBuf(iData);
             while(!(Get_UART_Status() & C_UARTRIF))    ;      //wait send complete
             Set_UART_Status(Get_UART_Status());
             iData ++;
             iData &= 0x00FF;
         }
    }
```

21 10-bit A/D Converter

21.1Introduction

In SPMC70xxx, the analog-to-digital converter has up to eight analog inputs. ADC channel input selection: Port B[7:0] has to be configured as **input floating** according to channel selection, respectively. In SPMC70xxx, an eight-channel 10-bit ADC is built-in for variety of applications. SPMC70xxx is embedded an 8-channel ADC with 10-bit resolution. It is used for many applications such as touch panel, battery power detection, …etc. For speech record, an external AGC is needed.

The channel inputs of ADC are shared with GPIO PortB bit7-bit0, each pin can be controlled to disable digital function when corresponding ADC channel is enabled. The A/D allows conversion of an analog input signal to corresponding 10-bit digital scale. The output of sample and hold is the input into the converter. This converter generates a result via successive approximation. The analog reference voltage is selectable to either the device's AVDD, 1/2 AVDD or 2V reference voltage. The block diagram of A/D converter is shown in Figure 19-1.

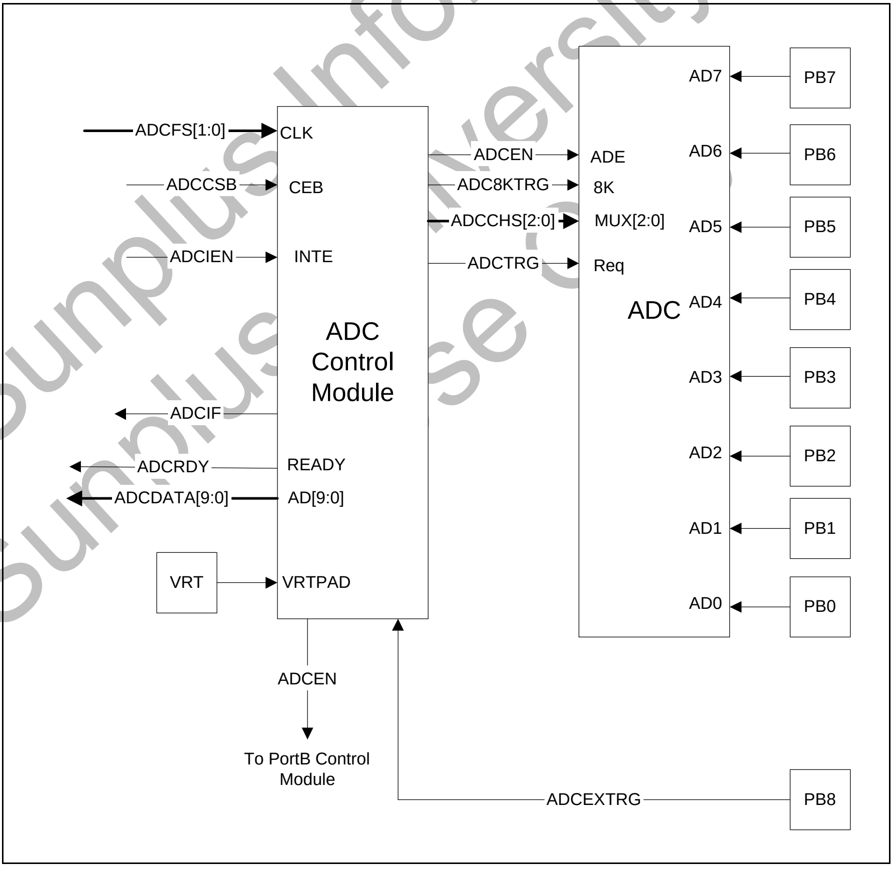


Figure 19-1 ADC function block

The A/D converter has two registers:

A/D Control Register (P_ADC_Ctrl)

A/D Result Register (P_ADC_Data)

21.2 Control Register

21.2.1 P_ADC_Ctrl (R/W) ($7070)

B15	B14	B13	B12	B11	B10	B9	B8
-	-	R/W	R/W	R/W	R/W	R/W	R/W
-	-	0	0	0	0	0	0
-	-	ADCIF	ADCTRG	ADCEXTRG	ADC8KTRG	ADCIEN	ADCTREF

B7	B6	B5	B4	B3	B2	B1	B0
-	R/W	R/W	R/W	R/W	R/W	R/W	R/W
-	0	0	0	0	0	0	0
-	ADCCHS			ADCFS		ADCCSB	ADCEN

Bit 15:14 Reserved

Bit 13 ADCIF: ADC conversion ready status

1: conversion ready

0: conversion not ready

Bit12 ADCTRG:software ADC conversion request

1: enable

0: disable

Bit 11 ADCEXTRG: external ADC conversion request trigger from PB8 pad

1: enable

0: disable

Note: PB8 can be set as floating, pull-up and pull-down configuration.

Bit 10 ADC8KTRG: ADC conversion request with constant frequency 8KHz

1:enable

0:disable

Note: it prefers to microphone record or needs constant-frequency-conversion application.

Bit 9 ADCIEN: ADC conversion ready status interrupt enable

1: enable

0: disable

Bit 8 ADCTREF: ADC VDD reference select

1: ADC top reference voltage equal to VRT pin

0: ADC top reference voltage equal to AVDD.

Bit 7 Reserved

Bit 6:4 ADCCHS: Select A/D converter channel input

000: AD0 (PB0)

001: AD1 (PB1)

010: AD2 (PB2)

011: AD3 (PB3)

100: AD4 (PB4)

101: AD5 (PB5)

110: AD6 (PB6)

111: AD7 (PB7)

Bit 3:2 ADCFS: A/D converter clock selection

00: CPUCLK /16

01: CPUCLK /32

10: CPUCLK /64

11: CPUCLK /128

Bit 1 ADCCSB: A/D converter chip select

0: select ADC block

1: un-select ADC block

Bit 0 ADCEN: A/D converter enable

0: disable ADC block

1: enable ADC block

21.2.2 P_ADC_Data(R)($7071H)

B15	B14	B13	B12	B11	B10	B9	B8
R	-	-	-	-	-	R	R
0	-	-	-	-	-	0	0
ADCRDY	-	-	-	-	-	ADCDATA	

B7	B6	B5	B4	B3	B2	B1	B0
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0
ADCDATA							

Bit 15 ADCRDY: ADC conversion ready

1: conversion ready

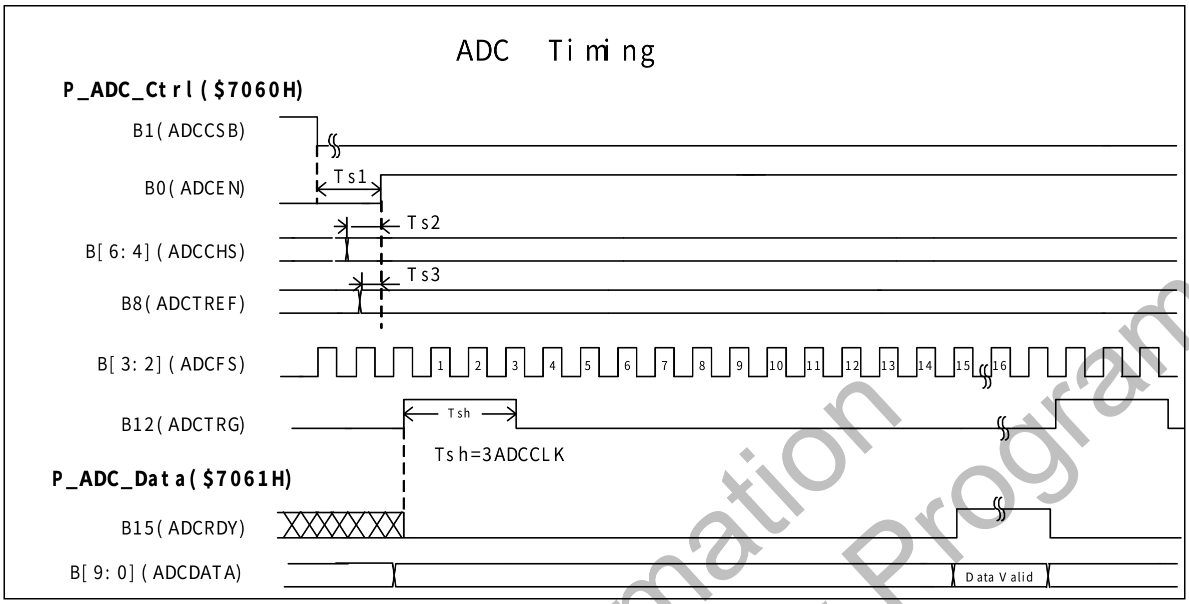
0: conversion not ready

Bit 14:10 Reserved

Bit 9:0 ADCData: ADC conversion data

21.3Operation

Analog-To-Digital Converter (ADC)



NOTE:

A: Ts1: the settle time of ADC chip enabled; min. 2ms @CL=0.1uF for VCM.

B: Ts2: the settle time of the changed ADC channel; min. 7us @CL=1000pF [4] for PB[7:0] ADC channel loading. Ts2 is proportional to the ADC channel input loading.

C: the settle time of the external ADC top voltage; min. 2us (VRT voltage need settle first.)

21.4Design Tips

The following example shows how to use AD.

? **Example19-1**? Sample AD with 8KHz frequency, convert 10 times and save to buffer named T_Index,user can compare the 50 times AD Value.

A) Use assemble language

```
.iram
T_Index: .dw 10 dup (0)

.code
.public _main
_main:
        R1 = 0xfffe         //set IOB0 as input(floating)
        [P_IOB_Dir] = R1
        R1 = 0xffff
        [P_IOB_Attrib] = R1
        R1 = 0xffff
        [P_IOB_Data] = R1

        //enable software ADC conversion request, ADC   conversion frequency 8KHz,
        //ADC top reference voltage equal to AVDD,Select AD0,enable ADC block
        R1 = 0x1401
        [P_ADC_Ctrl] = R1
         R3 = 0;

    L_ADC_Loop:
        R2 = [P_ADC_Ctrl]
        test R2,0x2000
        jz L_ADC_Loop
        R1 = [P_ADC_Data]
        R1 &= 0x03ff          //Bit[9:0]   ADC Data
         R4 = R3 + T_Index
        [R4] = R1             //save to buffer
        R3 += 1;
        cmp R3,10
        jbe L_ADC_Loop
       nop
    L_MainLoop:
       nop
       jmp L_MainLoop
```

A) Use C language

```
      #include "SPMC701FM0.H"
      #include "unSPMACRO.H"

      main()
      {
        unsigned int i,iData;
        unsigned int uiAdcData[10] = {0};

        *P_IOB_Dir = 0x0000;      //set IOB0 input with floating
        *P_IOB_Attrib = 0x0001;
        *P_IOB_Data = 0x0000;
```

```
     *P_ADC_Ctrl=0x1401;          //set ADC contrl part

    for(i=0;i<=10;i++)
    {
        while(!(*P_ADC_Ctrl & C_ADCIF)) ; //check the ADC ready
        {
            iData = *P_ADC_Data;
            uiAdcData[i] = iData & 0x03FF;    //low 10 bit is effective
        }
    }
    while(1) i++;
}
```

22 Instruction Set

22.1Introduction

The following table 20-1 shows all instruction types, operation styles, and execution cycles. The contents of table means:

1. RW: Memory Read Waiting cycle, RW if no wait state insertion and RW = N if wait state = N.
2. SW: Memory Write Waiting Cycle, SW= 0 ~N. = 0 ~ N
3. RW : store or read waiting cycle, SRW = SW when ALU = store else SRW = RW.

Table 20-1 instruction types

Type	Operation	Cycles
JMPF	Goto label	5+2RW
DSI6	DS=I6	2+RW
JMPR	Goto MR	4+RW
CALL	CALL label	9+2RW+2SW
FIR_MOV	FIR_MOV_ON/OFF	2+RW
Fraction	Fraction ON/OFF	2+RW
INT SET	INT FIQ/IRQ	2+RW
IRQ	IRQ ON/OFF	2+RW
SECBANK	SECBANK ON/OFF	2+RW
FIQ	FIQ ON/OFF	2+RW
IRQ Nest Mode	IRQNEST ON/OFF	2+RW
BREAK	BREAK	10+2RW+2SW
CALLR	CALL MR	8+RW+2SW
DIVS	DIVS MR,R2	2+RW
DIVQ	DIVQ MR,R2	3+RW
EXP	R2= EXP R4	2+RW
NOP	NOP	2+RW
DS Access	DS=Rs/ Rs=Ds	2+RW
FR Access	FR=Rs/ Rs=FR	2+RW
MUL	MR = Rd* Rs,{ss,us,uu}	12+RW / 13+RW (uu)
MULS	MR = [Rd]*[Rs], size,{ss,us,uu}	us,ss : 10*N+6 + (N+1)*2*RW + {N*SW} / uu: 11*N+6 + (N+1)*2*RW + {N*SW}
Register BITOP	BITOP Rd,Rs	4+RW
Register BITOP	BITOP Rd,offset	4+RW
Memory BITOP	BITOP DS: [Rd],offset	7+2RW+SW
Memory BITOP	BITOP DS: [Rd],Rs	7+2RW+SW
Shift	Rd=Rd LSFT Rs	8+RW
RETI	RETI	8+3RW / 10+4RW (IRQ NEST ON)
RETF	RETF	8+3RW

Type	Operation	Cycles
Base+Disp6	Rd = Rd op [BP+IM6]	6+2RW
Imm6	Rd = Rd op IM6	2+RW
Branch	Jxx label	2+RW / 4+RW (taken)
Indirect	Push/Pop Rx,Ry to [Rs]	4+ 2N + (N+1)RW
DS_Indirect	Rd = Rd op DS: [Rs++]	6+RW+SRW / 7+RW+SRW (PC)
Imm16	Rd = Rs op IMM16	4+2RW / 5+2RW (PC)
Direct16	Rd = Rs op A16	7+2RW+SRW / 8+2RW+SRW (PC)
Direct6	Rd = Rd op A6	5+RW+SRW / 6+RW+SRW (PC)
Register	Rd = Rd op Rs SFT sfc	3+RW / 5+RW (PC)

(a) MULS Cycle: 10*N+6 + (N+1)*2*RW + {N*SW} (signedxsigned, unsignedxsigned)

11*N+6 + (N+1)*2*RW + {N*SW} (unsignedxunsigned) where N=1..16, (N*SW) = 0 if FIR_MOVE OFF

(b) DS_Indirect Cycle: 6 + RW + SRW / 7 + RW + SRW (write to PC)

(c) Direct16 Cycle: 7 + 2*RW + SRW / 8 + 2*RW + SRW (write to PC)

(d) Direct6 Cycle: 5+ RW +SRW / 6+ RW +SRW (write to PC)

(e) RW: Memory Read Waiting cycle, RW= 0 ~ N, SW: Memory Write Waiting Cycle, SW= 0 ~N.

SRW: store or read waiting cycle, SRW = SW when ALU = store else SRW = RW.

(f) D: 0 (forward jump) / 1 (backward jump)

W: 0 (not store) / 1 (store)

DS: 0 (not using DS) / 1 (using DS)

22.2 ALU Operation

The table 20-2 shows the relationship between operation and flag. The flag have 4 components, N, Z, S, C.

N:	Negative
Z:	Zero
S:	Sign
C:	Carry

Table 20-2 relationship between operation and flag

Operation Type	Operation	N	Z	S	C	Example
Addition	a + b	√	√	√	√	Rd=Rd + Rs
Add with carry	a + b + c	√	√	√	√	Rd=Rd + Rs,c
Subtraction	a +~ b + 1	√	√	√	√	Rd=Rd - Rs
Sub with carry	a +~ b + c	√	√	√	√	Rd=Rd - Rs,c
Compare	a +~ b + 1	√	√	√	√	Cmp Rd,Rs
Negative	~ b + 1	√	√	-	-	Rd= ~Rs
Exclusive OR	a xor b	√	√	-	-	Rd=Rd ^ Rs
Load	a = b	√	√	-	-	Rd=Rs
OR	a or b	√	√	-	-	Rd=Rd | Rs

Operation Type	Operation	N	Z	S	C	Example
AND	a and b	√	√	-	-	Rd=Rd & Rs
Test	Test a, b	√	√	-	-	Test Rd, Rs
Store	Store	-	-	-	-	[Rd]=Rs

22.3 Register

The SPMC series provide two kind of internal register for calculation. One is R1 ~ R4, the other is SR1 ~ SR4.User can switch secondary register bank mode by secbank on/off to change two of them, when shadow register mode is on, all operation with R1-R4 will map to SR1-SR4. The register address and relation are shown in the table 20-3.

Table 20-3 two kind of internal register for calculation

Rd/Rs	SECBANK OFF	SECBANK ON
000	SP	SP
001	R1	SR1
010	R2	SR2
011	R3	SR3
100	R4	SR4
101	BP	BP
110	SR	SR
111	PC	PC

Note: Rd: Destination Register Rs: Source Register

The SPMC also has general registers: SP (Stack pointer), SR (Status register), and FR (Flag Register). The SP is FILO structure and indicates the stack depth. Normally, user must set the SP value from start address of MCU. The SR shows the status of MCU. It indicates the code or data segment and flag for “negative”, “zero”, “sign” and “carry”. The bit relation is shown in Table 20-4.

Table 20-4 **SR: (Status Register)**

F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
DS						N	Z	S	C	CS					

CS: Code Segment

DS: Data Segment

N: Negative Flag

Z: Zero Flag

S: Sign Flag

C: Carry Flag

The FR shows the MCU statue flag and the description is shown in Table 20-5.

Table 20-5 **FR: (Flag Register)**

F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
-	AQ	BNK	FRA	FIR	SFT BUF				F	I	INE	IRQ PRI			

AQ:	DIVS/DIVQ AQ bit flag.
BNK:	Secondary Bank Registers Mode.
FRA:	FRACTION MODE.
FIR:	FIR_MOVE MODE.
SFT BUF:	Shift buffer/Guard Bit (FIR).
F:	FIQ enable flag.
I:	IRQ enable flag.
INE:	IRQ Nest MODE.
IRQ PRI:	IRQ Priority register.

22.4 Branch condition

The following table 20-6 shows the branch conditions. The condition match must refer to SR (Status register). Four types of branch flags are: N (negative), Z(zero), S(sign), and C(Carry). When user uses some operations to distinguish the program flow, the NZSC flag will be changed. And programmer can use the following instruction to make program jump for matching address. program jump instruction is shown in Table 20-6.

Table 20-6 program jump instruction

Syntax	Description	Branch
JCC	carry clear	C==0
JB	Below (unsigned)	C==0
JNAE	not above and equal (unsigned)	C==0
JCS	carry set	C==1
JNB	not below (unsigned)	C==1
JAE	above and equal (unsigned)	C==1
JSC	sign clear	S==0
JGE	great and equal (signed)	S==0
JNL	not less (signed)	S==0
JSS	sign set	S==1
JNGE	not great than (signed)	S==1
JL	Less (signed)	S==1
JNE	not equal	Z==0
JNZ	not zero	Z==0
JZ	zero	Z==1
JE	equal	Z==1
JPL	plus	N==0
JMI	minus	N==1
JBE	below and equal (unsigned)	Not (Z==0 and C==1)
JNA	not above (unsigned)	Not (Z==0 and C==1)
JNBE	not below and equal (unsigned)	Z==0 and C==1
JA	above (unsigned)	Z==0 and C==1
JLE	less and equal (signed)	Not (Z==0 and S==0)
JNG	not great (signed)	Not (Z==0 and S==0)
JNLE	not less and equal (signed)	Z==0 and S==0
JG	great (signed)	Z==0 and S==0
JVC	not overflow (signed)	N == S
JVS	Overflow (signed)	N != S
JMP	Unconditional branch	Always

22.5 Shift

ASR: (Shift Right with MSB of Rs)

Rd	E_2	E_1	E_0	B_F	B_E	B_D	B_C	B_B	B_A	B_9	B_8	B_7	B_6	B_5	B_4	B_3

SB	B_2	B_1	B_0	S_3

$E_2E_1E_0$ are signed extension bit of the most significant bit in Rs

LSL: (Shift Left)

SB	S_0	B_F	B_E	B_D

Rd	B_C	B_B	B_A	B_9	B_8	B_7	B_6	B_5	B_4	B_3	B_2	B_1	B_0	0	0	0

LSR: (Shift Right)

Rd	0	0	0	B_F	B_E	B_D	B_C	B_B	B_A	B_9	B_8	B_7	B_6	B_5	B_4	B_3

SB	B_2	B_1	B_0	S_3

ROL: (Rotate Left with SB)

SB	S_0	B_F	B_E	B_D

Rd	B_C	B_B	B_A	B_9	B_8	B_7	B_6	B_5	B_4	B_3	B_2	B_1	B_0	S_3	S_2	S_1

ROR: (Rotate Right with SB)

Rd	S_2	S_1	S_0	B_F	B_E	B_D	B_C	B_B	B_A	B_9	B_8	B_7	B_6	B_5	B_4	B_3

SB	B_2	B_1	B_0	S_3

22.6 Instruction Set

N	Z	S	C
-	-	-	-

? **Note? :** "-" isn't effect to sign; "√" is effect to sign;

22.6.1 Segmented Far Call (CALL)

N	Z	S	C
-	-	-	-

? **Description? :** A far call instruction with code segment assigned; both PC and SR are pushed to stack automatically.

? **Cycles ? :** 9 + 2*RW + 2*SW (RW: Read Waiting Cycle . SW: Write Waiting Cycle)

? **Word ? :** 2

? **Version ? :** ALL Version.

【 **Syntax** 】: CALL Label;

? **Example ? :**

```
.CODE
.PUBLIC      _main
_main:
          r1 = 0x0001
          r1 += 0x0002
          call   F_Sub                  //Call subroutine
          r1 += r4
L_Loop:
          nop
          jmp   L_Loop

F_Sub:
          r2 = 0x5555
          r3 = 0x5555
          mr = r2*r3                //r2 multiply r3
          retf
```

22.6.2 Segmented Far Indirect Call (CALL MR)

N	Z	S	C
-	-	-	-

? **Description? :** A far call instruction with MR register; both PC and SR are pushed to stack automatically.

? **Cycles ? :** 8 + RW + 2*SW

? **Word ? :** 1

? **Version ? :** ISA 1.2 and above.

? **Syntax ? :** CALL MR;

? **Example ? :** In this example the value of MR points to F_Sub.

```
.CODE
.PUBLIC  _main
_main:
          r1 = 0xefff;
          r2 = 0x1713;
          r3 = OFFSET   F_Sub              // the value of r3 is the "offset"
          r4 = SEG   F_Sub                 //the value of r4 is the "segment"
          call   mr
L_MainLoop:
          nop
```

```
            jmp     L_ MainLoop
        F_Sub:
            r2 = 0x5555
            r3 = 0x5555
            mr = r2*r3              //r2 multiply r3
            retf
```

22.6.3 GOTO (Far Jump)

N	Z	S	C
-	-	-	-

? **Description**? **:** Go to user's specified address. Target address was not be limited by 64K words.

? **Cycles** ? **:** 5 + 2*RW

? **Version** ? **:** ISA 1.1 and above.

? **Syntax** ? **:** GOTO LABEL;

? **Example** ? **:**

```
        .CODE
        .PUBLIC _main
        _main:
                r1 = 0x000a
                r2 = 0x0001
                cmp r1,r2                   //compare the bigger between r1 And r2
                jnae   L_OK                 //r1 is bigger
                goto   L_Exchange_Value     //Exchange r1 And r2 While r1 Smaller r2
            L_OK:
                nop
                jmp   L_OK
            L_Exchange_Value:
                r3 = r1
                r1 = r2                     //Exchange r1 And r2
                r2 = r3
                jmp   L_OK
```

22.6.4 GOTO(Far Indirect JMP)

N	Z	S	C
-	-	-	-

? **Description**? **:** A far indirect jump instruction with MR register. The content of MR {R4[5: 0],R3[15: 0]} will be used as destination address.

? **Cycles** ? **:** 4 + RW

? **Version** ? **:** ISA 1.2 and above.

? **Syntax** ? : GOTO MR;

? **Example** ? **:**

```
.CODE
.PUBLIC   _main
_main:
        r4 =   SEG   L_GOTO_Sub           //the value of r4 is the "segment"
        r3 =   OFFSET   L_GOTO_Sub          //the value of r3 is the "offset"
        goto   mr                           // the value of mr(r4: r3) is   L_GOTO_Sub
        nop
  L_GOTO_Sub:
        nop
        jmp    L_GOTO_Sub
```

22.6.5 RETF (Return from subroutine)

N	Z	S	C
-	-	-	-

? **Description**? **:** RETF pops SR and PC from stack and return from subroutine. Note that the SR and PC are popped back after RETF. Therefore, the SR is the same as at the time when Call is made.

? **Cycles** ? **:** 8 + RW + 2*SW

? **Word** ? **:** 1

? **Version** ? **:** All Version.

? **Syntax** ? **:** RETF;

? **Example** ? **:**

```
.CODE
.PUBLIC _main
_main:
        nop
        call   F_Delay1
L_loop:
        nop
        jmp   L_loop

F_Delay1:
        nop
```

```
                    retf
```

22.6.6 RETI (Return from interrupt)

N	Z	S	C
-	-	-	-

? **Description**? **:** if IRQ nest mode on RETI pops FR, SR, PC from stack else RETI pops SR and PC from stack then return from interrupt service routine and clear internal FIQ or IRQ interrupt flag. Note that the SR is pop back after RETI. Therefore, the SR is the same as at the time when IRQ is executed. Also, a post-FIQ is unable to interrupt existed FIQ.

? **Cycles** ? **:** 8 + 3*RW (IRQ Nest Mode OFF) / 10 + 4*RW (IRQ Nest Mode ON)

? **Word** ? **:** **1**

? **Version** ? **:** All Version.

? **Syntax** ? **:** RETI;

22.6.7 Break (Software Interrupt)

N	Z	S	C
-	-	-	-

? **Description**? **:** Generate a software interrupt. System will go to [$0x00fff5].

? **Cycles** ? **:** 10 + 2*RW + 2*SW

? **Word** ? **:** 1

? **Version** ? **:** ALL Version.

? **Syntax** ? **:** BREAK;

? **Example**? **:**

```
.CODE
.PUBLIC   _main
_main:
        r1 = 0x1234
        BREAK                    // generate software interrupt
  L_MainLoop:
        nop
        jmp      L_ MainLoop


.TEXT
.PUBLIC    _BREAK
_BREAK:
        push    r1, r5    to    [sp]
        nop                          //Write Break Function In this Area
        pop   r1, r5   from   [sp]
        reti                            //return to the main function
```

22.6.8 FIR_MOV ON/OFF

N	Z	S	C
-	-	-	-

? **Description? :** Enable/Disable automatic data movement for FIR operations. It affects the behavior of FIR. The status is a global behavior. Hence, use it in interrupt with care.

? **Cycles ? :** 2 + RW

? **Word ? :** 1

? **Version ? :** ALL Version.

? **Syntax ? :** FIR_MOV ON/OFF;

? **Example ? :**

```
    .IRAM
        .VAR    NO_1=0x0001, NO_2=0x0002,    NO_3=0x0003, NO_4=0x0004
        .VAR    NO_5=0x0005, NO_6=0x0006,    NO_7,        NO_8
    .CODE
    .PUBLIC   _main
    _main:
          FIR_MOV ON            //Enable automatic data movement for FIR operations.
          r1 =   NO_2            //Give the address of "NO_2" to 'r1'
          r2 =   NO_5            //Give the address of "NO_5" to 'r2'
      Loop_Muls:
          mr = [r1]*[r2],us,2     //The result of operation is stored in r4: r3
          [NO_7] =r3              //Read the result (low 16 bite) to NO_7
          [NO_8] = r4             //Read the result (high 16 bite) to NO_8
          nop                     //The value of r1,r2 both added 2
          jmp     Loop_Muls
```

22.6.9 Secondary Register Bank Mode ON/OFF

N	Z	S	C
-	-	-	-

? **Description? :** Switch secondary register bank mode ON/OFF, 4 shadow registers SR1-SR4 are added in µ 'nSP™ ISA Version 1.2 , user can use this instruction to switch secondary register bank mode on/off, when shadow register mode is on, all operation with R1-R4 will map to SR1-SR4.

? **Cycles ? :** 2 + RW

? **Word** ? **:** 1

? **Version** ? **:** ISA 1.2 and above.

? **Syntax** ? **:** SECBANK ON/OFF;

? **Example** ? **:**

```
.CODE
.PUBLIC     _main
_main:
        SECBANK   ON
        r1 =   0x1234           // 0x1234 is sent to SR1 , no value in r1
        r2 =   0x5678           // 0x5678 is sent to SR2 , no value in r2
        r3 =   0x0f0f                // 0x0f0f is sent to SR3 , no value in r3
        r4 = 0xf0f0             // 0xf0f0 is sent to SR4 , no value in r4
L_Loop:
        nop
        jmp   L_Loop
```

22.6.10 Fraction Mode ON/OFF

N	Z	S	C
-	-	-	-

? **Description**? **:** Switch Fraction Mode ON/OFF, when Fraction mode is ON, the 32 bit result of multiplication will be shift 1 bit left.

? **Cycles** ? **:** 2 + RW

? **Word** ? **:** 1

? **Version** ? **:** ISA 1.2 and above.

? **Syntax** ? **:** FRACTION ON/OFF;

? **Example** ? **:**

```
.CODE
.PUBLIC    _main
_main:
        r1 = 0x0002
        r2 = 0x2244
        FRACTION ON
        mr = r1*r2          //Case1:   "FRACTION OFF" the result(R4R3) is 0x00004488
                            //Case2:   "FRACTION ON" the result(R4R3) is 0x00008910
L_Loop:
        nop
        jmp     L_Loop
```

22.6.11 IRQ Nested Mode ON/OFF

N	Z	S	C
-	-	-	-

? **Description? :** Switch IRQ Nest Mode ON/OFF, when IRQ Nest Mode is ON, CPU will push FR/SR/PC into stack before enter IRQ routine and pop them out when leaving, and higher priority IRQ interrupting is allowed when executing IRQ routine, user can set-up IRQ PRI register to change the interrupt priority

? **Cycles ? :** 2 + RW

? **Word ? :** 1

? **Version ? :** ISA 1.2 and above.

? **Syntax ? :** IRQNEST ON/OFF;

22.6.12 IRQ ON/OFF

N	Z	S	C
-	-	-	-

? **Description? :** Enable/Disable IRQ

? **Cycles ? :** 2 + RW

? **Word ? :** 1

? **Version ? :** ALL Version.

? **Syntax ? :** IRQ ON/OFF; //Open or close IRQ interrupt

? **Example ? :**

```
.include hardware.inc
.RAM
.VAR   R_Value

.CODE
.PUBLIC   _main
_main:
        r1 = 0xffff                 //Set PORTA as output
        [P_IOA_Dir] = r1
        [P_IOA_Attrib] = r1
        [P_IOA_Data] = r1
        r1 = 0x0001                 //Set 2Hz Inturrupt
        [P_INT_Ctrl] = r1
        IRQ ON                      //Set IRQ ON
L_Loop:
```

```
                    nop
                    jmp    L_ Loop
            .TEXT
            .PUBLIC    _IRQ7
            _IRQ7:
                   push    r1,r5    to    [SP]
                   r4 = [R_Value]
                   r4 ^=  0xffff
                   [R_Value] = r4
                   [P_IOA_Data] = r4
                   r1 =    0x0001                //Clear 2Hz interrupt flag
                   [P_INT_Clear] = r1
                   pop     r1 ,r5    from       [SP]
                   reti
```

22.6.13 FIQ ON/OFF

N	Z	S	C
-	-	-	-

? **Description**? **:** Enable/Disable FIQ

? **Cycles** ? **:** 2 + RW

? **Word** ? **:** 1

? **Version** ? **:** ALL Version.

? **Syntax** ? **:** FIQ ON/OFF; //Open or close FIQ interrupt

22.6.14 Interrupt Set

N	Z	S	C
-	-	-	-

? **Description**? **:** Set FIQ/IRQ flags.

? **Cycles** ? **:** 2 + RW

? **Word** ? **:** 1

? **Version**? **:** All Version.

? **Syntax** ? **:**

```
INT     FIQ;                      //enable FIQ, (disable IRQ)
INT     IRQ;                      //enable IRQ, (disable FIQ)
INT     FIQ, IRQ;                  //enable FIQ and IRQ
INT     OFF;                       //disable FIQ and IRQ
```

? **Example** ? **:**

```
        INT     FIQ;                        // enable FIQ, (disable IRQ)
        INT     FIQ, IRQ;                   // enable IRQ, FIQ
```

22.6.15 DS Segment Direct Access Instruction

N	Z	S	C
-	-	-	-

? **Description**? : Set and get DS segment register instruction, Since the DS-Segment is a 6 bit register, only the LSB 6 bit of Rs will be put on DS when executing set DS operation and zero-extended is used when executing get DS operation.

? **Cycles** ? : 2 + RW

? **Word** ? : 1

? **Version** ? : ISA 1.2 and above.

? **Syntax** ? :

```
DS = RS;
RS = DS;
```

? **Example** ? :

```
.CODE
.PUBLIC   _main
_main:
        r1 =   0xff1f              //Send LSB6 bite of r1 to DS
        DS = r1                    //From the register of "SR" we can find it
L_Loop :
        nop
        jmp   L_Loop
```

22.6.16 Change DS Segment with immediate 6-bit value

N	Z	S	C
-	-	-	-

? **Description**? : Set DS segment register with 6-bit immediate value instruction.

? **Cycles** ? : 2 + RW

? **Word** ? : 1

? **Version** ? : ISA 1.2 and above.

? **Syntax** ? : DS = I6;

? **Example** ? :

```
.CODE
.PUBLIC   _main
_main:
```

```
            DS = 0xffff              //From the register of "SR" we can find it's change

    L_Loop :
            nop
            jmp   L_Loop
```

22.6.17 NOP

N	Z	S	C
-	-	-	-

? **Description**? **:** No operation, only increase PC to next address.

? **Cycles** ? **:** 2 + RW

? **Word** ? **:** 1

? **Version** ? **:** ISA 1.2 and above.

【 **Syntax** 】: NOP;

22.6.18 Processor Flag Access Instruction*

N	Z	S	C
-	-	-	-

? **Description**? **:** Direct access the Processor Flag Register, the content of register is show below.

F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
-	AQ	BNK	FRA	FIR	SFT BUF				F	I	INE	IRQ PRI			

AQ:	DIVS/DIVQ AQ bit flag, default is 0.
BNK:	Secondary Bank Registers Mode, default is 0.
FRA:	FRACTION MODE, default is 0.
FIR:	FIR_MOVE MODE, default is 0. (FIR_MOVE ON : 0 FIR_MOVE OFF : 1)
SFT BUF:	Shift buffer/Guard Bit (FIR), default is 4'b0000.
F:	FIQ flag, default is 0.
I:	IRQ flag, default is 0.
INE:	IRQ Nest MODE, default is 0.
IRQ PRI:	IRQ Priority register, default is 1000 after reset, if any IRQ occurred, IRQ PRI register will be set as the IRQ priority, only the IRQ with higher priority can interrupt it, user can customize the IRQ Nesting with setting priority register.

Priority: IRQ0>IRQ1>IRQ2>IRQ3>IRQ4>IRQ5>IRQ6>IRQ7

IRQ Enable algorithm : 0 – PRI -1

Note: FIQ still has highest priority than any IRQ if FIQ is enable.

For example:

1? if PRI is 1000, all IRQ 0-7 are enable

2? f PRI is 0000, all IRQ 0-7 are disable

3? IRQ3 occurred, PRI will be set to 0011, only IRQ0-2 are enable

? **Cycles** ? **:** 2 + RW

? **Word** ? **:** 1

? **Version** ? **:** ISA 1.2 and above.

? **Syntax** ? **:**

FR = Rs ;

Rs = FR ;

22.6.19 DIVS/DIVQ

N	Z	S	C
-	-	-	-

? **Description**? **:** These instructions implement 32-bit division. There are two divide primitives, DIVS and DIVQ. A single precision divide, with a 32-bit numerator and a 16-bit denominator, yielding a 16-bit quotient, executes in 16*3 cycles. Higher precision divides are also possible. The division can be either signed or unsigned, but both the numerator and denominator must be the same. Place the 32-bit numerator at R4: R3, the 16-bit denominator at R2 and clear the AQ flag then executed with the divide primitives, DIVS and DIVQ. Repeated execution of DIVQ implements a non-restoring conditional add-subtract division algorithm. At the conclusion of divide operation the quotient will be placed at R3.

To implement a signed divide, first execute the DIVS instruction once, which computes the sign of the quotient. Then execute the DIVQ instruction for as many times as there are bits remaining in the quotient.

? **Cycles**? **:** 2 + RW (DIVS) / 3 + RW (DIVQ)

? **Word** ? **:** 1

? **Version** ? **:** ISA 1.2 and above.

? **Syntax** ? **:**

DIVS MR,R2;

DIVQ MR,R2;

? **Example1** ? **:** Divide a 32 bit signed number(0xffff1713) with 16-bit divisor(divide 0x0625)

```
.CODE
.PUBLIC   _main
_main:
        r4 = 0xffff;
        r3 = 0x1713;
        r2 = 0x0625;
```

```
          r1 = 0x0001
          r4 = r4 lsl r1;          //shift the dividend 1 bit left for (31.1) format
          mr |= r3 lsl r1      //change to follows
          r1 = fr;
          clrb r1,0xe;        //clear AQ flag
          fr = r1;
          r1 = 0;
          DIVS mr,R2;               //divide the signed bit
      L_Signed:
          DIVQ mr,R2;          //divide the remainder
          r1 += 1;
          cmp r1,0xf;
          jne L_Signed;
          r1 = r3;              //the quotient is in R3,that is 0xffda

      L_Loop:
          nop;
          jmp   L_Loop
```

? **Example2 ?** **:** Divide a 32 bit unsigned number(0x00031713) with 16-bit divisor(0x0625)

```
   .CODE
   .PUBLIC   _main
   _main:
       r4 = 0x0003;
       r3 = 0x1713;
       r2 = 0x0625;
       r1 = 0x0001
       r4 = r4 lsl r1;          // shift the dividend 1 bit left for (31.1) format
       mr |= r3 lsl r1     // change to follows
       r1 = fr;
       clrb r1,0xe;       // clear AQ flag
       fr = r1;
       r1 = 0;
   L_unsigned:
       DIVQ mr,R2;      // divide the remainder
       r1 += 1;
       cmp r1,0x10;
       jne L_unsigned;
```

```
        r1 = r3;            //the quotient is in R3,that is 0x0080

    L_Loop:
        nop;
        jmp   L_Loop
```

22.6.20 EXP

N	Z	S	C
-	-	-	-

? **Description?** **:** The EXP instruction derives the effective exponent of the R4 register to prepare for the normalization operation, and places the result in the R2. The result is equal to the number of redundant unsign(sign) bits in the input. The truth table is as follows:

Rsrc	Rdst	Result
SNDDDDDD DDDDDDDD	0	
SSNDDDDD DDDDDDDD	1	
SSSNDDDD DDDDDDDD	2	
SSSSNDDD DDDDDDDD	3	
SSSSSNDD DDDDDDDD	4	
SSSSSSND DDDDDDDD	5	
SSSSSSSN DDDDDDDD	6	
SSSSSSSS NDDDDDDD	7	
SSSSSSSS SNDDDDDD	8	
SSSSSSSS SSNDDDDD	9	
SSSSSSSS SSSNDDDD	10	
SSSSSSSS SSSSNDDD	11	
SSSSSSSS SSSSSNDD	12	
SSSSSSSS SSSSSSND	13	
SSSSSSSS SSSSSSSN	14	
SSSSSSSS SSSSSSSS	15	

Note:

S = Sign Bit
N = Non-Sign Bit
D = Don't care Bit

? **Cycles** ? **:** 2 + RW

? **Word** ? **:** 1

? **Version?** **:** ISA 1.2 and above.

? **Syntax** ? **:** R2= EXP R4;

? **Example?** **:**

```
        .CODE
        .PUBLIC _main
        _main:
                r4 = 0x1000;        //the 15th bite of "r4" is "0"
                r2 = EXP     r4;        //the result is calculate by this way from 15th bit to the bit, which is
```

```
opposite
                                        //to 15th and sub 1,the result is 2
L_Loop:
        nop
        jmp     L_Loop
```

22.6.21 Bit Operation

N	Z	S	C
-	√	-	-

? **Description**? **:** Bit_op for Rs or Memory DS: Rs (0x000000-0x3fffff), the origin value before operation will affect the zero flag, that is if the origin bit is 1 then zero flag will be false else zero flag will be true. With registers bit operation, only the least significant 4 bits of register will be used, any other bits will be ignored.

? **Bit_op** ? **:** tstb / setb / clrb / invb

? **Cycles** ? **:** 4 + RW (BITOP with Registers) / 7 + 2*RW + SW (BITOP with memory)

? **Word** ? **:** 1

? **Version** ? **:** ISA 1.2 and above.

? **Syntax** ? **:**

1. Memory Bit Operation

```
BITOP {DS: }[Rd], offset;    //offset=0-15
BITOP {DS: }[Rd], Rs;              //Rs=R0-R7        Rd=R0-R6
```

2. Register Bit Operation

```
BITOP Rd, offset;            //offset=0-15
BITOP Rd, Rs;                //Rs=R0-R7            Rd=R0-R6
```

BITOP :

00	01	10	11
tstb	setb	clrb	invb

? **Example1**? **:**

```
tstb   D: [r2], 13;
setb        [r1], r3;
clrb   r3, 10;
invb   r4, r5;
```

? **Example2**? **:**

```
.CODE
.PUBLIC   _main
```

```
_main:
        r1 = 0x0f0f
        r5 = 0x0001
        setb   r1, 15          //turn "r1" to 0x8f0f
        clrb   r1, 15          //turn "r1" to 0x0f0f
        invb   r1, r5         //the R5 =1,so invert the 1th of r1, the result is 0x0f0d
        tstb   [r1], 8        //only change the "Z"

L_Loop:
        nop
        jmp    L_Loop
```

22.6.22 Shift Operation

N	Z	S	C
√	√	-	-

? **Description**? **:** Shift_op for Rd and Rs. A 16-bit multi-cycle shifter with one instruction, and support 32-bit shifter with combining 2 shift instruction, when using 32-bit shifter the result must place at R4: R3, the rotate ROR/ROL operation will shift with carry and the drop bit will place at carry flag after operation.

? **Shift_op** ? **:** supported ASR/ASROR/LSL/LSLOR/LSR/LSROR/ROL/ROR

? **Cycles** ? **:** 8 + RW

? **Word** ? **:** 1

? **Version** ? **:** ISA 1.2 and above.

? **Syntax** ? **:**

Rd = Rd shift_op Rs; Rd: destination register (only R0-R6 is available)

Rs: shift count register

Rs[4: 0] valid : ASR, ASROR, LSL, LSLOR, LSR, LSROR

Rs[3: 0] valid: ROL, ROR

Shift_OP :

000	001	010	011	100	101	110	111
ASR	ASR-OR	LSL	LSL-OR	LSR	LSR-OR	ROL	ROR

? **Example1** ? **:**

```
r2 = r2 asr r1;                  // 16-bit arithmetic right shift
r3 = r3 lsr r1;            // 32-bit arithmetic right shift
mr |= r4 asr r1                  // result will be put r4: r3
```

? **Example2** ? **:**

```
r1 = 0x1234
r1 = r1 rol 4          // the value is 0x2340
r1 = r1 rol 4          // the value is 0x3401
r1 = r1 rol 4          // the value is 0x4012
r1 = r1 rol 4          // the value is 0x0123
r1 = r1 rol 4          // the value is 0x1234

r1 = 0xffff
r1 = r1 lsr 4          // the value is 0x0fff
r1 = r1 rol 4          // the value is 0xffff
r1 = r1 lsl 4          // the value is 0xfff0
r1 = r1 ror 4          // the value is 0xffff
r1 |= r1 lsr 4         //First r1 lsr 4 bite then OR the origin"r1"

r1 = 0xff00            //the origin value of "r1" change to 0xff00
r2 = 0x000             //the origin value of "r2" is 0x0007
r1 = r1 asr r2         //the result is 0xfffe
```

? **Example3** ? **:**

```
r1 = 0x0006
r3 = 0x0f0f
r4 = 0xf0f0
r3 = r3 lsr r1         //first r3 lsr 6 bite
mr |= r4 asr r1             //put some bit of r4 to r3; r3=0xc03c,r4=0xffc3
```

22.6.23 Registers Multiplication (Mul)

N	Z	S	C
-	-	-	-

? **Description**? **:** Multiply two registers (signed to signed or unsigned to signed) and place result in registers R4: R3 (MR). If Fraction mode is enable the result of multiplication will be shift 1-bit left. The Rd register only support R0-R6

? **Cycles** ? **:** 12+RW (signedxsigned, unsignedxsigned) / 13+RW (unsignedxunsigned)

? **Word** ? **:** 1

? **Version** ? **:** (signedxsigned, unsignedxsigned) All Version / (unsignedxunsigned) ISA 1.2 and above.

? **Syntax** ? **:**

```
MR = Rd * Rs;          // signed-to-signed multiplication used if no indication
MR = Rd * Rs,uu;       // unsigned-to-unsigned
MR = Rd * Rs, ss;      // signed-to-signed
MR = Rd * Rs, us;      // Rd is unsigned and Rs is singed.   The first register is always
unsigned
```

// and the second register is signed.

? **Example1? :**

```
r1 = 0x8002
r2 = 0x0002            // the result is stored in R4: R3
mr = r1*r2               // 12 cycle, r4=0xffff,r3=0x0004
mr = r1*r2,ss           // 12 cycle, r4=0xffff,r3=0x0004
mr = r2*r1,us           // 12 cycle, r4=0xffff,r3=0x0004
mr =r1*r2,uu            // 14 cycle, r4=0x0001,r3=0x0004
```

22.6.24 Sum of Registers Multiplication (Muls)

N	Z	S	C
-	-	√	-

? **Description? :** Use a 36-bit Arithmetic Unit to sum up a consecutive register multiplication (FIR) and propagate coefficients for next FIR. If Fraction mode is enable, the result of every multiplication will be shift 1-bit left and then sums up. The Rd and Rs pointers are adjusted automatically, if FIR_MOVE flag is On and when n >1, the contents of memory pointed by Rd are also moved forward. After operation, the 4-bit MSB (guard bit) of ALU will be placed at shift-buffer and sign flag will be set if any overflow occurred of the result.

? **Cycles** ? : 10*N+6 + (N+1)*2*RW + {N*SW} (signedxsigned, unsignedxsigned)

11*N+6 + (N+1)*2*RW + {N*SW} (unsignedxunsigned)

where N=1..16, (N*SW) = 0 if FIR_MOVE OFF

? **Word** ? **:** 1

? **Version** ? **:** (signedxsigned, unsignedxsigned) All Version /(unsignedxunsigned) ISA 1.2 and above.

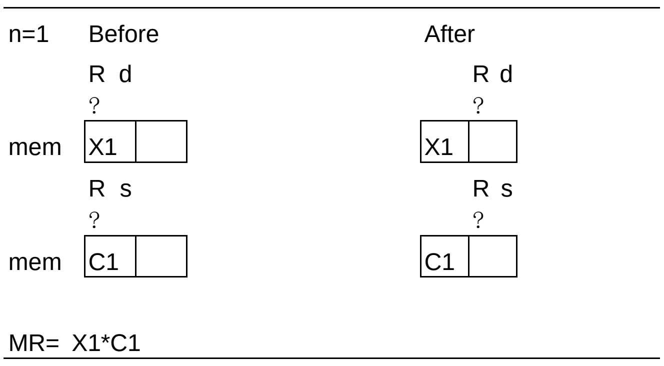
? **Syntax** ? **:**

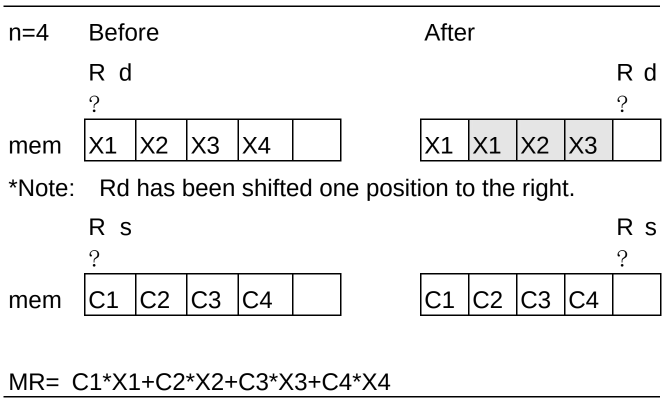
MR = [Rd] * [Rs]{, ss}{, n};

MR = [Rd] * [Rs], us{, n};

*Signed-to-signed multiplication used if no indication; MR is cleared before summation.

*n: [16: 1]. if n is omitted, 1 is used.




n=4
Before
After
R d
?
R d
?
mem
X1
X2
X3
X4
X1
X1
X2
X3
*Note: Rd has been shifted one position to the right.
R s
?
R s
?
mem
C1
C2
C3
C4
C1
C2
C3
C4
MR= C1*X1+C2*X2+C3*X3+C4*X4

Note: The FIR operation of previous version u'nSP (ISA 1.0 / ISA 1.1) does not change the sign flag, but the new version of u'nSP (ISA 1.2) will change this flag to indicate overflow occurren ce .

? **Limitation**? **:**

Result is incorrect if the following conditions are both met:

n >1 and (either Rs or Rd are set to R3 or R4) and (Rs and Rd are set to the same register.)

? **Example**? **:**

MR = [R1] * [R2], 8;

MR = [R1] * [R2], us, 2;

Note : Here you can refer the example in the FIR_MOV ON/OFF instruction

22.6.25 Conditional branch (Branch)

N	Z	S	C
-	-	-	-

? **Description**? **:** A short jump instruction to local label (address within ±63 words), and one non-conditional included. See Conditional Branch Table (Table 4) for details.

? **Cycles** ? **:** 2+RW (not-taken) / 4+RW (taken)

? **Word** ? **:** 1

? **Version** ? **:** All Version.

? **Syntax** ? **:**

Conditional_jump label;

jmp label;

? **Example**? **:**

```
jmp     LABEL;          // non-conditional jump
jne     LABEL;          // jump when not equal
```

22.6.26 Push/Pop operations (By Indirect Addressing)

N	Z	S	C

-	-	-	-

? **Description**? **:** Push/Pop a set of registers to/from memory, which is indicated by Rs consecutively. That is also used for RETF/RETI.

? **Cycles** ? **:** 4+ 2*N + (N+1)*RW // where N=1..7, the number of pop or push.

? **Word** ? **:** 1

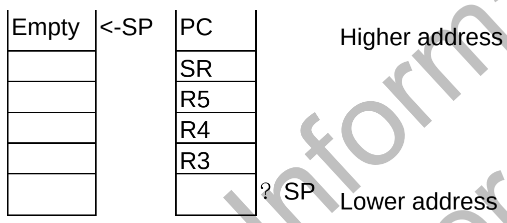
? **Version** ? **:** All Version.

? **Syntax** ? **:**

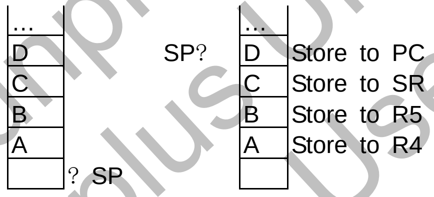
```
push Rx, Ry to [Rs] ;        //push Rx through Ry to [Rs]
pop Rx, Ry from [Rs] ;       //pop Rx through Ry from [Rs]
```

? **Example** ? **:**

```
Push R3, PC to [SP];          //Push R3 through PC (R7) to [SP]
```



```
pop R4, PC from [SP];          //pop R4 through PC (R7) from [SP]
pop SR, PC from [SP];          // retf
```



Note: push R1, R5 to [SP] is equivalent to push R5, R1 to [SP]

22.6.27 ALU operations with base plus displacement memory access (Base+Disp6)

N	Z	S	C
√	√	√	√

? **Description**? **:** Alu_op for Rd and memory [BP + IM6], or store Rd to [BP + IM6]. BP is R5.

? **ALU** ? **:** Supported sub/sbc/adc/add/cmp/and/or/xor/load/test/neg/store

? **Cycles** ? **:** 6+ 2*RW

? **Word** ? **:** 1

? **Version** ? **:** All Version.

? **Syntax** ? **:**

```
Rd alu_op = [BP + IM6];
[BP + IM6] = Rd;
```

Note: List of all possible alu_ops:

```
Rd -= [ BP + IM6];                  // sub
Rd -= [ BP + IM6] , carry;          // sbc
Rd += [ BP + IM6] , carry;          // adc
Rd += [ BP + IM6];                  // add
cmp Rd , [ BP + IM6];               // cmp
Rd &= [ BP + IM6];                  // and
Rd |= [ BP + IM6];                  // or
Rd ^= [ BP + IM6];                  // xor
Rd = [ BP + IM6];                   // load
test Rd , [ BP + IM6];              // test
Rd =– [ BP + IM6];                  // negative
[ BP + IM6] = Rd;                   // store
```

? **Example1**? **:**

```
r2 += [bp + 0x20], Carry;           // r2 add memory and carry
[bp + 0x06] = r1;                   // store
```

? **Example2**? **:**

```
.IRAM
T_Buffer:    .DW 0x1111, 0x1111, 0x1111, 0x1111;

......                          //The origin value of bp is 0
bp = T_Buffer;
r1 = [bp+1]                     //Get the value of memory address [bp+1] and store to r1
......
```

22.6.28 ALU operations with 6-bit immediate (Imm6)

N	Z	S	C
√	√	√	√

? **Description**? **:** Alu_op for Rd and IM6

? **ALU** ? **:** supported sub/sbc/adc/add/cmp/and/or/xor/load/test/neg

? **Cycles** ? **:** 2+RW

? **Word** ? **:** 1

? **Version** ? **:** All Version.

? **Syntax** ? **:**

```
Rd      ALU_OP =   IM6;
```

Note: List of all possible alu_ops:

```
Rd -= IM6;                  // Sub
Rd -= IM6 , Carry;          // Sbc
Rd += IM6 , Carry;          // Adc
Rd += IM6;                  // Add
cmp Rd , IM6;                       // Cmp
Rd &= IM6;                  // And
Rd |= IM6;                  // Or
Rd ^= IM6;                  // Xor
Rd = IM6;                   // Load
test   Rd , IM6;            // Test
Rd =– IM6;                  // Negative
```

? **Example1? :**

```
r1 += 0x20, Carry;                  //r1=r1+0x20+carry
```

? **Example2? :**

```
r1 = 0x00ef              // 3 cycles the origin value of r1 is 0x00ef
r2 = 0x0073
r3 = 0x0fff
r1 -= 0x0001                   // the result turns to 0x00ee
r2 += 0x0030                   // the result turns to 0x00a3
r3 &= 0x0002             // the result turns to 0x0002
```

? **Examples3? :**

```
        r1 = 0x0002              // 3 cycles
        cmp r1,0x0010             // 3 cycles compare r1 with 0x0010
        jbe L_Lower              // if smaller jump to the other place
        nop                      // 3 cycles, go down
        ……

L_Lower:
    nop
    nop
    ……
```

22.6.29 ALU operations with indirect memory access and optional increment/decrement (DS_Indirect)

N	Z	S	C
√	√	√	√

? **Description?:** Alu-op with indirect memory access and optional Increment/Decrement. This instruction is the only instruction that can access both page 0 and non-zero pages.

? **ALU ? :** supported sub/sbc/adc/add/cmp/and/or/xor/load/test/neg/store

? **Cycles ? :** 6 + RW + SRW / 7 + RW + SRW (write to pc)

SRW : store or read waiting cycle, SRW = SW when ALU = store else SRW = RW.

? **Word ? :** 1

? **Version ? :** All Version.

? **Syntax ? :**

Rd alu_op = {D: }[Rs];

Rd alu_op = {D: }[Rs--];

Rd alu_op = {D: }[Rs++];

Rd alu_op = {D: }[++Rs];

? **Example? :**

r1 &= D: [r2++]; // r1= r1 AND (The content in the address of DS and r2),r2=r2+1
// when r2 is 0xffff doing r2= r2 + 1 operation , make r2 be 0 and DS +1

cmp r1 , [++r2]; //Compare r1 with the content in the address of (r2+1)

D: [r2--] =r1; //Assignr1 to the address of DS and r2, r2=r2-1
// when r2 is 0, doing r2 – 1 operation , make r2 be ffff and DS -1

22.6.30 ALU operations with 16-bit immediate (Imm16)

N	Z	S	C
√	√	√	√

? **Description? :** Alu_op for Rd and Rs register.

? **ALU ? :** supported sub/sbc/adc/add/cmp/and/or/xor/load/test/neg

? **Cycles ? :** 4+2*RW / 5+2*RW (write to PC)

? **Word ? :** 2

? **Version ? :** All Version.

? **Syntax ? :**

Rd = Rs alu_op IM16;

? **Example2? :**

```
.CODE
.PUBLIC   _main
_main:
          r1 = 0xffff
          r2 = 0
L_Imm16:
          cmp r1,0x00ff               //compare r1 with 16 immediate 0x00ff
          jb   L_Loop
          r1 -= 0x00ff            //sub 0x00ff
          r2 += 1
          goto L_Imm16
L_Loop:
          jmp     L_Loop
```

22.6.31 ALU operations with direct memory addressing (Direct 16)

N	Z	S	C
√	√	√	√

? **Description**? **:** Alu_op for Rd, Rs and Memory A16. Two distinct modes applied by setting W bit.

? **ALU** ? **:** supported sub/sbc/adc/add/cmp/and/or/xor/load/test/neg

? **Cycles** ? **:** 7 + 2*RW + SRW / 8 + 2*RW + SRW (write to PC)

? **Word** ? **:** 2

? **Version** ? **:** All Version.

? **Syntax** ? **:**

```
Rd = Rs alu_op [A16];
[A16] = Rs alu_op Rd;
```

? **Example1**? **:**

```
bp = r1 + [Q_table];        // bp =r1 + content of Q_table where Q _table is in 0x0000 ~ 0xffff
[Hashing_tbl] = -bp;         // Assign –bp to Hashing_tbl
[Q_Table+10] = bp ^ r2;      // The result of bp xor r2 is assigned to Q_Table+10
r1 = [0x8079];               //Get the value of address 0x8079and send it to r1
```

22.6.32 ALU Operations with Direct Memory Addressing (Direct 6)

N	Z	S	C
√	√	√	√

? **Description**? **:** Alu_op for Rd, and Memory A6 (0x00 ~ 0x03f).

? **ALU** ? **:** supported sub/sbc/adc/add/cmp/and/or/xor/load/test/neg

? **Cycles** ? **:** 5+ RW +SRW / 6+ RW +SRW (write to PC)

? **Word** ? **:** 1

? **Version** ? **:** All Version.

? **Syntax** ? **:**

Rd = Rd alu_op [A6]

Rd = [A6]

[A6] = Rd

? **Example1**? **:**

```
.RAM
.VAR   Init_Q ;              // Init_Q is an variable located in memory 0x00~0x3f
.CODE
.public _main
_main:
        r1=   [0x0009]                //get the value of address 0x0009and send it to r1
        r2 = 0x5555;
        bp = [Init_Q];           // load content of Init_0 to bp
        [Init_Q] = R2;           // load r2 to Init_0
         ......
```

22.6.33 ALU Operations with Shift (Register)

N	Z	S	C
√	√	√	√

? **Description**? **:** Alu_op for Rd and Rs. Additional shift operation can be applied to Rs before alu_op.

? **ALU** ? **:** supported sub/sbc/adc/add/cmp/and/or/xor/load/test/neg/store

? **Cycles** ? **:** 3 + RW / 5 + RW (write to PC)

? **Word** ? **:** 1

? **Version** ? **:** All Version.

? **Syntax** ? **:**

Rd {alu_op} = Rs {, Carry}

Rd {alu_op} = Rs ASR n {, Carry}

Rd {alu_op} = Rs LSL n {, Carry}

Rd {alu_op} = Rs LSR n {, Carry}

Rd {alu_op} = Rs ROL n {, Carry}

Rd {alu_op} = Rs ROR n {, Carry}

where n is number of position shift and ranged in [4: 1]

Input:

Rs	B_F	B_E	B_D	B_C	B_B	B_A	B_9	B_8	B_7	B_6	B_5	B_4	B_3	B_2	B_1	B_0

SB	S_3	S_2	S_1	S_0

Suppose n=3, after shift op of

ASR: (Shift Right with MSB of Rs)

Rd	E_2	E_1	E_0	B_F	B_E	B_D	B_C	B_B	B_A	B_9	B_8	B_7	B_6	B_5	B_4	B_3

SB	B_2	B_1	B_0	S_3

$E_2E_1E_0$ are signed extension bit of the most significant bit in Rs

LSL: (Shift Left)

SB	S_0	B_F	B_E	B_D

Rd	B_C	B_B	B_A	B_9	B_8	B_7	B_6	B_5	B_4	B_3	B_2	B_1	B_0	0	0	0

LSR: (Shift Right)

Rd	0	0	0	B_F	B_E	B_D	B_C	B_B	B_A	B_9	B_8	B_7	B_6	B_5	B_4	B_3

SB	B_2	B_1	B_0	S_3

ROL: (Rotate Left with SB)

SB	S_0	B_F	B_E	B_D

Rd	B_C	B_B	B_A	B_9	B_8	B_7	B_6	B_5	B_4	B_3	B_2	B_1	B_0	S_3	S_2	S_1

ROR: (Rotate Right with SB)

Rd	S_2	S_1	S_0	B_F	B_E	B_D	B_C	B_B	B_A	B_9	B_8	B_7	B_6	B_5	B_4	B_3

SB	B_2	B_1	B_0	S_3

Note :

1. FIQ, IRQ and user routine has their own Shift buffers. User does not need to save shift buffe r in interrupt routines.
2. high buffer values are unknown after multiplication or filter operations. User should make no a ssumption to its value after the operations.

? **Example? :**

```
bp += r1 asr 4, Carry;            // bp = bp + (r1/(2**4)) with Carry
bp = r1 lsl 2;                    // bp = r1 << 2
```

23 HARDWARE SUMMARY

P_IOA_Data ($7000H)

B15	B14	B13	B12	B11	B10	B9	B8
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
P_IOA_Data							

B7	B6	B5	B4	B3	B2	B1	B0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
P_IOA_Data							

P_IOA_Buffer ($7001H)

B15	B14	B13	B12	B11	B10	B9	B8
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
P_IOA_Buffer							

B7	B6	B5	B4	B3	B2	B1	B0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
P_IOA_Buffer							

P_IOA_Dir ($7002H)

B15	B14	B13	B12	B11	B10	B9	B8
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
P_IOA_Dir							

B7	B6	B5	B4	B3	B2	B1	B0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
P_IOA_Dir							

P_IOA_Attrib ($7003H)

B15	B14	B13	B12	B11	B10	B9	B8
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
P_IOA_Attrib							

B7	B6	B5	B4	B3	B2	B1	B0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
P_IOA_Attrib							

P_IOA_Latch ($7004H)

B15	B14	B13	B12	B11	B10	B9	B8
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0
P_IOA_Latch							

B7	B6	B5	B4	B3	B2	B1	B0
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0
P_IOA_Latch							

P_IOB_Data ($7008H)

B15	B14	B13	B12	B11	B10	B9	B8
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
P_IOB_Data							

B7	B6	B5	B4	B3	B2	B1	B0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
P_IOB_Data							

P_IOB_Buffer ($7009H)

B15	B14	B13	B12	B11	B10	B9	B8
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
P_IOB_Buffer							

B7	B6	B5	B4	B3	B2	B1	B0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
P_IOB_Buffer							

P_IOB_Dir ($700AH)

B15	B14	B13	B12	B11	B10	B9	B8
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
P_IOB_Dir							

B7	B6	B5	B4	B3	B2	B1	B0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
P_IOB_Dir							

P_IOB_Attrib ($700BH)

B15	B14	B13	B12	B11	B10	B9	B8
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
P_IOB_Attrib							

B7	B6	B5	B4	B3	B2	B1	B0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
P_IOB_Attrib							

P_IOB_Latch ($700CH)

B15	B14	B13	B12	B11	B10	B9	B8
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0
P_IOB_Latch							

B7	B6	B5	B4	B3	B2	B1	B0
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0
P_IOB_Latch							

P_IOC_Data ($7010H)

B15	B14	B13	B12	B11	B10	B9	B8
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
P_IOC_Data							

B7	B6	B5	B4	B3	B2	B1	B0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
P_IOC_Data							

P_IOC_Buffer ($7011H)

B15	B14	B13	B12	B11	B10	B9	B8
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
P_IOC_Buffer							

B7	B6	B5	B4	B3	B2	B1	B0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
P_IOC_Buffer							

P_IOC_Dir ($7012H)

B15	B14	B13	B12	B11	B10	B9	B8
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
P_IOC_Dir							

B7	B6	B5	B4	B3	B2	B1	B0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
P_IOC_Dir							

P_IOC_Attrib ($7013H)

B15	B14	B13	B12	B11	B10	B9	B8
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
P_IOC_Attrib							

B7	B6	B5	B4	B3	B2	B1	B0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
P_IOC_Attrib							

P_IOC_Latch ($7014H)

B15	B14	B13	B12	B11	B10	B9	B8
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0
P_IOC_Latch							

B7	B6	B5	B4	B3	B2	B1	B0
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0
P_IOC_Latch							

P_IOD_Data ($7018H)

B15	B14	B13	B12	B11	B10	B9	B8
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
P_IOD_Data							

B7	B6	B5	B4	B3	B2	B1	B0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
P_IOD_Data							

P_IOD_Buffer ($7019H)

B15	B14	B13	B12	B11	B10	B9	B8
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
P_IOD_Buffer							

B7	B6	B5	B4	B3	B2	B1	B0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
P_IOD_Buffer							

P_IOD_Dir ($701AH)

B15	B14	B13	B12	B11	B10	B9	B8
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
P_IOD_Dir							

B7	B6	B5	B4	B3	B2	B1	B0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
P_IOD_Dir							

P_IOD_Attrib ($701BH)

B15	B14	B13	B12	B11	B10	B9	B8
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
P_IOD_Attrib							

B7	B6	B5	B4	B3	B2	B1	B0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
P_IOD_Attrib							

P_IOD_Latch ($701CH)

B15	B14	B13	B12	B11	B10	B9	B8
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0
P_IOD_Latch							

B7	B6	B5	B4	B3	B2	B1	B0
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0
P_IOD_Latch							

P_TimeBase_Setup ($7040H)

B15	B14	B13	B12	B11	B10	B9	B8
-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-

B7	B6	B5	B4	B3	B2	B1	B0
-	-	-	R/W	R/W	R/W	R/W	R/W
-	-	-	0	0	0	0	0
-	-	-	TMBENB	TMB2FS		TMB1FS	

P_TimeBase_Clr ($7041H)

B15	B14	B13	B12	B11	B10	B9	B8
W	W	W	W	W	W	W	W
0	1	0	1	0	1	0	1
TMBCLR							

B7	B6	B5	B4	B3	B2	B1	B0
W	W	W	W	W	W	W	W
0	1	0	1	0	1	0	1
TMBCLR							

P_Timer0_Ctrl ($7044H)

B15	B14	B13	B12	B11	B10	B9	B8
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
TMR0EN	TMR0IF	TMR0IEN	EXT2PS		CCP0MS		

B7	B6	B5	B4	B3	B2	B1	B0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
CCP0MS	SYNCLK	CLKBFS			CLKAFS		

P_Timer0_Preload ($7045H)

B15	B14	B13	B12	B11	B10	B9	B8
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
TMR0PR							

B7	B6	B5	B4	B3	B2	B1	B0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
TMR0PR							

P_Timer0_CCPR ($7046H)

B15	B14	B13	B12	B11	B10	B9	B8
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
TMR0CCPR							

B7	B6	B5	B4	B3	B2	B1	B0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
TMR0CCPR							

P_Timer0_CCPR2 ($7047H)

B15	B14	B13	B12	B11	B10	B9	B8
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
TMR0CCPR2							

B7	B6	B5	B4	B3	B2	B1	B0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
TMR0CCPR2							

P_Timer1_Ctrl ($7048H)

B15	B14	B13	B12	B11	B10	B9	B8
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
TMR1EN	TMR1IF	TMR1IEN	EXT1PS		CCP1MS		

B7	B6	B5	B4	B3	B2	B1	B0
R/W	R/W	-	-	-	R/W	R/W	R/W
0	0	-	-	-	0	0	0
CCP1MS	SYNCLK	-	-	-	CLKAFS		

P_Timer1_Preload ($7049H)

B15	B14	B13	B12	B11	B10	B9	B8
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
TMR1PR							

B7	B6	B5	B4	B3	B2	B1	B0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
TMR1PR							

P_Timer1_CCPR ($704AH)

B15	B14	B13	B12	B11	B10	B9	B8
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
TMR1CCPR							

B7	B6	B5	B4	B3	B2	B1	B0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
TMR1CCPR							

P_Timer1_CCPR2 ($704BH)

B15	B14	B13	B12	B11	B10	B9	B8
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
TMR1CCPR2							

B7	B6	B5	B4	B3	B2	B1	B0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
TMR0CCPR2							

P_Timer2_Ctrl ($704CH)

B15	B14	B13	B12	B11	B10	B9	B8
R/W	R/W	R/W	R/W	R/W	R	R	R
0	0	0	0	0	0	0	0
TMR2EN	TMR2IF	TMR2IE	CNTRM		-		
TMR2EN	TMR2IF	TMR2IEN	EXT1PS		-		

B7	B6	B5	B4	B3	B2	B1	B0
R	R/W	R	R	R	R/W	R/W	R/W
0	0	0	0	0	0	0	0
-	SYNC	-			CLKAS		
-	SYNCLK	-			CLKAFS		

P_Timer2_Preload ($704DH)

B15	B14	B13	B12	B11	B10	B9	B8
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
TMR2PR							

B7	B6	B5	B4	B3	B2	B1	B0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
TMR2PR							

P_INT_Ctrl ($7060H)

B15	B14	B13	B12	B11	B10	B9	B8
-	-	-	-	-	R/W	R/W	R/W
-	-	-	-	-	0	0	0
-	-	-	-	-	EXT2INEN	EXT1INEN	EXT2IEN

B7	B6	B5	B4	B3	B2	B1	B0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
EXT1IEN	EXT2IEDG	EXT1IEDG	KEYIEN	TMB2IEN	TMB1IEN	4HZIEN	2HZIEN

P_INT_Priority ($7061H)

B15	B14	B13	B12	B11	B10	B9	B8
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
ADCIP	EXT2IP	EXT1IP	UARTIP	SPIIP	SIOIP	PCIIP	TMR2IP

B7	B6	B5	B4	B3	B2	B1	B0
R/W	R/W	R/W	R/W	-	-	-	-
0	0	0	0	-	-	-	-
TMR1IP	TMR0IP	LVDIP	KEYIP	-	-	-	-

P_INT_Status ($7062H)

B15	B14	B13	B12	B11	B10	B9	B8
R	R/W	R/W	R	R	R	R	R
0	0	0	0	0	0	0	0
ADCINT	EXT2INT	EXT1INT	UARTINT	SPIINT	SIOINT	PCIINT	TMR2INT

B7	B6	B5	B4	B3	B2	B1	B0
R	R	R	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
TMR1INT	TMR0INT	LVDINT	KEYCIF	TMB2INT	TMB1INT	4HZINT	2HZINT

P_WatchDog_Ctrl ($7063)

B15	B14	B13	B12	B11	B10	B9	B8
-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-

B7	B6	B5	B4	B3	B2	B1	B0
-	-	-	-	R/W	R/W	R/W	R/W
-	-	-	-	0	0	0	0
-	-	-	-	WDTTMR			WDTENB

P_WatchDog_Clr ($7064)

B15	B14	B13	B12	B11	B10	B9	B8
W	W	W	W	W	W	W	W
0	1	0	1	1	0	1	0
WDTCLR							

B7	B6	B5	B4	B3	B2	B1	B0
W	W	W	W	W	W	W	W
0	1	0	1	1	0	1	0
WDTCLR							

P_Reset_Status ($7065)

B15	B14	B13	B12	B11	B10	B9	B8
-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-

B7	B6	B5	B4	B3	B2	B1	B0
-	-	R/W	R/W	R/W	R/W	R/W	R/W
-	-	0	0	0	0	0	0
-	-	PCIRST	ICERST	ILLADDR	WDTRST	LVRST	EXTRST

P_LVDLVR_Ctrl ($7066)

B15	B14	B13	B12	B11	B10	B9	B8
-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-

B7	B6	B5	B4	B3	B2	B1	B0
-	-	R/W	R/W	R/W	R/W	R/W	R/W
-	-	0	0	0	0	0	0
-	-	LVDENB	VREFENB	LVRENB	LVDIEN	LVDLS	

P_LVD_Status ($7067)

B15	B14	B13	B12	B11	B10	B9	B8
-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-

B7	B6	B5	B4	B3	B2	B1	B0
-	-	-	-	-	-	-	R/W
-	-	-	-	-	-	-	0
-	-	-	-	-	-	-	LVDIF

P_Flash_RW ($7068)

B15	B14	B13	B12	B11	B10	B9	B8
R	R/W	R/W	R/W	R/W	R/W	R/W	R/W
1	0	0	0	0	0	0	0
-	BK14WENB	BK13WENB	BK12WENB	BK11WENB	BK10WENB	BK9WENB	BK8WENB

B7	B6	B5	B4	B3	B2	B1	B0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
BK7WENB	BK6WENB	BK5WENB	BK4WENB	BK3WENB	BK2WENB	BK1WENB	BK0WENB

P_Clk_Ctrl ($7069)

B15	B14	B13	B12	B11	B10	B9	B8
-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-

B7	B6	B5	B4	B3	B2	B1	B0
-	-	-	-	-	R/W	R/W	R/W
-	-	-	-	-	0	0	0
-	-	-	-	-	CLKSEL		

P_Wakeup_Source ($706A)

B15	B14	B13	B12	B11	B10	B9	B8
R/W	R/W	R/W	R/W	R	R	R	R
0	0	0	0	0	0	0	0
KCAWEN	KCBWEN	KCCWEN	KCDWEN	UARTIWEN	SPIIWEN	SIOIWEN	PCIIWEN

B7	B6	B5	B4	B3	B2	B1	B0
R	R	R	R	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
TMR2WEN	TMR1WEN	TMR0WEN	LVDWEN	TMB2WEN	TMB1WEN	4HZWEN	2HZWEN

P_Sleep_Enter ($706B)

B15	B14	B13	B12	B11	B10	B9	B8
W	W	W	W	W	W	W	W
0	1	0	1	0	1	0	1
SleepCMD							

B7	B6	B5	B4	B3	B2	B1	B0
W	W	W	W	W	W	W	W
1	0	1	0	1	1	0	0
SleepCMD							

P_Wait_State ($706C)

B15	B14	B13	B12	B11	B10	B9	B8
-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-

B7	B6	B5	B4	B3	B2	B1	B0
-	-	-	-	R/W	R/W	R/W	R/W
-	-	-	-	0	0	0	0
-	-	-	-	WSCYCT			

P_Stdby_Enter ($706D)

B15	B14	B13	B12	B11	B10	B9	B8
W	W	W	W	W	W	W	W
0	1	0	1	1	0	1	0
StdbyCMD							

B7	B6	B5	B4	B3	B2	B1	B0
W	W	W	W	W	W	W	W
0	1	0	1	1	0	1	0
StdbyCMD							

P_Halt_Enter (($706E)

B15	B14	B13	B12	B11	B10	B9	B8
W	W	W	W	W	W	W	W
0	1	0	1	0	1	0	1
HaltCMD							

B7	B6	B5	B4	B3	B2	B1	B0
W	W	W	W	W	W	W	W
0	1	0	1	0	1	0	1
HaltCMD							

P_Wakeup_Status ($706F)

B15	B14	B13	B12	B11	B10	B9	B8
-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-

B7	B6	B5	B4	B3	B2	B1	B0
-	-	-	-	R/W	R/W	R/W	R/W
-	-	-	-	0	0	0	0
-	-	-	-	HALTSF	SLEEPSF	STDBYSF	WAKEUPSF

P_ADC_Ctrl ($7070)

B15	B14	B13	B12	B11	B10	B9	B8
-	-	R/W	R/W	R/W	R/W	R/W	R/W
-	-	0	0	0	0	0	0
-	-	ADCIF	ADCTRG	ADCEXTRG	ADC8KTRG	ADCIEN	ADCTREF

B7	B6	B5	B4	B3	B2	B1	B0
-	R/W	R/W	R/W	R/W	R/W	R/W	R/W
-	0	0	0	0	0	0	0
-	ADCCHS			ADCFS		ADCCSB	ADCEN

P_ADC_Data(R)($7071H)

B15	B14	B13	B12	B11	B10	B9	B8
R	-	-	-	-	-	R	R
0	-	-	-	-	-	0	0
ADCRDY	-	-	-	-	-	ADCDATA	

B7	B6	B5	B4	B3	B2	B1	B0
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0
ADCDATA							

P_SPI_Ctrl ($7090)

B15	B14	B13	B12	B11	B10	B9	B8
-	-	-	-	-	-	R/W	R/W
-	-	-	-	-	-	0	0
-	-	-	-	-	-	SPICSEN	SPIEN

B7	B6	B5	B4	B3	B2	B1	B0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
SPIIEN	SPIMS	SPISMPS	SPIPHA	SPIPOL	SPIFS		

P_SPI_Status ($7091)

B15	B14	B13	B12	B11	B10	B9	B8
-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-

B7	B6	B5	B4	B3	B2	B1	B0
-	-	-	-	-	R	R	R/W
-	-	-	-	-	0	0	0
-	-	-	-	-	SPIROR	SPITBF	SPIIF

P_SPI_TxBuf ($7092)

B15	B14	B13	B12	B11	B10	B9	B8
-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-

B7	B6	B5	B4	B3	B2	B1	B0
W	W	W	W	W	W	W	W
0	0	0	0	0	0	0	0
SPITXBUF							

P_SPI_RxBuf ($7092)

B15	B14	B13	B12	B11	B10	B9	B8
-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-

B7	B6	B5	B4	B3	B2	B1	B0
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0
SPIRXBUF							

P_UART_Ctrl ($70A0)

B15	B14	B13	B12	B11	B10	B9	B8
-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-

B7	B6	B5	B4	B3	B2	B1	B0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
UARTTEN	UARTREN	UARTMS	UARTMPEN	UARTPEN	UARTPMS	UARTTIEN	UARTRIEN

P_UART_Status ($70A1)

B15	B14	B13	B12	B11	B10	B9	B8
-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-

B7	B6	B5	B4	B3	B2	B1	B0
R	R	R	R	R	R	R/W	R/W
0	0	0	0	0	0	0	0
UARTRBF	UARTTBY	UARTBMPT	UARTROR	UARTFE	UARTPE	UARTTIF	UARTRIF

P_UART_Reset ($70A2)

B15	B14	B13	B12	B11	B10	B9	B8
W	W	W	W	W	W	W	W
0	1	0	1	0	1	0	1
UARTRST							

B7	B6	B5	B4	B3	B2	B1	B0
W	W	W	W	W	W	W	W
0	1	0	1	0	1	0	1
UARTRST							

P_UART_BaudRateL ($70A3)

B15	B14	B13	B12	B11	B10	B9	B8
-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-

B7	B6	B5	B4	B3	B2	B1	B0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
UARTBRSL							

P_UART_BaudRateH ($70A4)

B15	B14	B13	B12	B11	B10	B9	B8
-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-

B7	B6	B5	B4	B3	B2	B1	B0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
UARTBRSH							

P_UART_TxBuf ($70A5)

B15	B14	B13	B12	B11	B10	B9	B8
-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-

B7	B6	B5	B4	B3	B2	B1	B0
W	W	W	W	W	W	W	W
0	0	0	0	0	0	0	0
UARTTXBUF							

P_UART_RxBuf ($70A6)

B15	B14	B13	B12	B11	B10	B9	B8
-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-

B7	B6	B5	B4	B3	B2	B1	B0
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0
UARTRXBUF							

P_SIO_Ctrl ($70B0)

B15	B14	B13	B12	B11	B10	B9	B8
-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-

B7	B6	B5	B4	B3	B2	B1	B0
R/W	R/W	R/W	R/W	-	R/W	R/W	R/W
0	0	0	0	-	0	0	0
SIOEN	SIOSCKOLB	SIOSDAOLB	SIOIEN	-	SIOADRMS	SIOMS	

P_SIO_Status ($70B1)

B15	B14	B13	B12	B11	B10	B9	B8
-	-	-	-	-	-	-	R
-	-	-	-	-	-	-	0
-	-	-	-	-	-	-	NACK

B7	B6	B5	B4	B3	B2	B1	B0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
SIOIF	SIOROR	SIODAT	SIOSTOP	SIOSTRT	SIODIRT	SIOUA	SIOBF

P_SIO_BUF ($70B2)

B15	B14	B13	B12	B11	B10	B9	B8
-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-

B7	B6	B5	B4	B3	B2	B1	B0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
SIOBUF							

P_SIO_ADDR ($70B3)

B15	B14	B13	B12	B11	B10	B9	B8
-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-

B7	B6	B5	B4	B3	B2	B1	B0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
SIOADR							

P_PCI_Ctrl ($70C0)

B15	B14	B13	B12	B11	B10	B9	B8
-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-

B7	B6	B5	B4	B3	B2	B1	B0
R/W	R/W	R/W	R/W	R/W	-	-	-
0	0	0	0	0	-	-	-
PCIEN	PCIPS	PCIMS	PCIIEN	RSTOEN	-	-	-

P_PCI_Status ($70C1)

B15	B14	B13	B12	B11	B10	B9	B8
-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-

B7	B6	B5	B4	B3	B2	B1	B0
R/W	R/W	R/W	R/W	R	R	-	-
0	0	0	0	0	0	-	-
PCIRBF	PCITBF	PCIROR	PCIIF	PCIDAT	PWRSF	-	-

P_PCI_IBUF ($70C2)

B15	B14	B13	B12	B11	B10	B9	B8
-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-

B7	B6	B5	B4	B3	B2	B1	B0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
PCIRXBUF							

P_PCI_OBUF ($70C3)

B15	B14	B13	B12	B11	B10	B9	B8
-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-

B7	B6	B5	B4	B3	B2	B1	B0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
PCITXBUF							

P_PCI_COMM ($70C4): PCI send command to slave port

B15	B14	B13	B12	B11	B10	B9	B8
W	W	W	W	W	W	W	W
0	0	0	0	0	0	0	0
PCICMD							

B7	B6	B5	B4	B3	B2	B1	B0
W	W	W	W	W	W	W	W
0	0	0	0	0	0	0	0
PCICMD							

P_Flash_Ctrl ($7555)

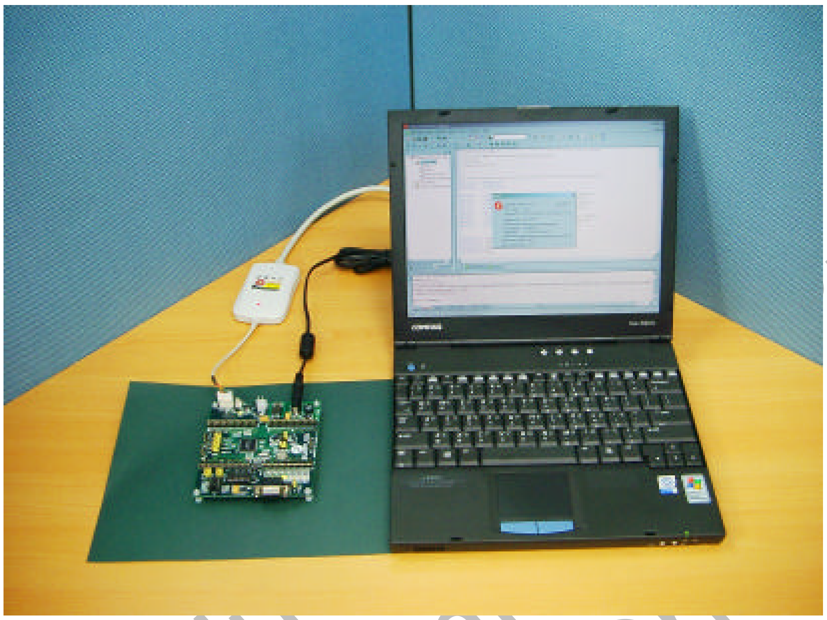
B15	B14	B13	B12	B11	B10	B9	B8
W	W	W	W	W	W	W	W
0	0	0	0	0	0	0	0
FlashCtrl							

B7	B6	B5	B4	B3	B2	B1	B0
W	W	W	W	W	W	W	W
0	0	0	0	0	0	0	0
FlashCtrl							

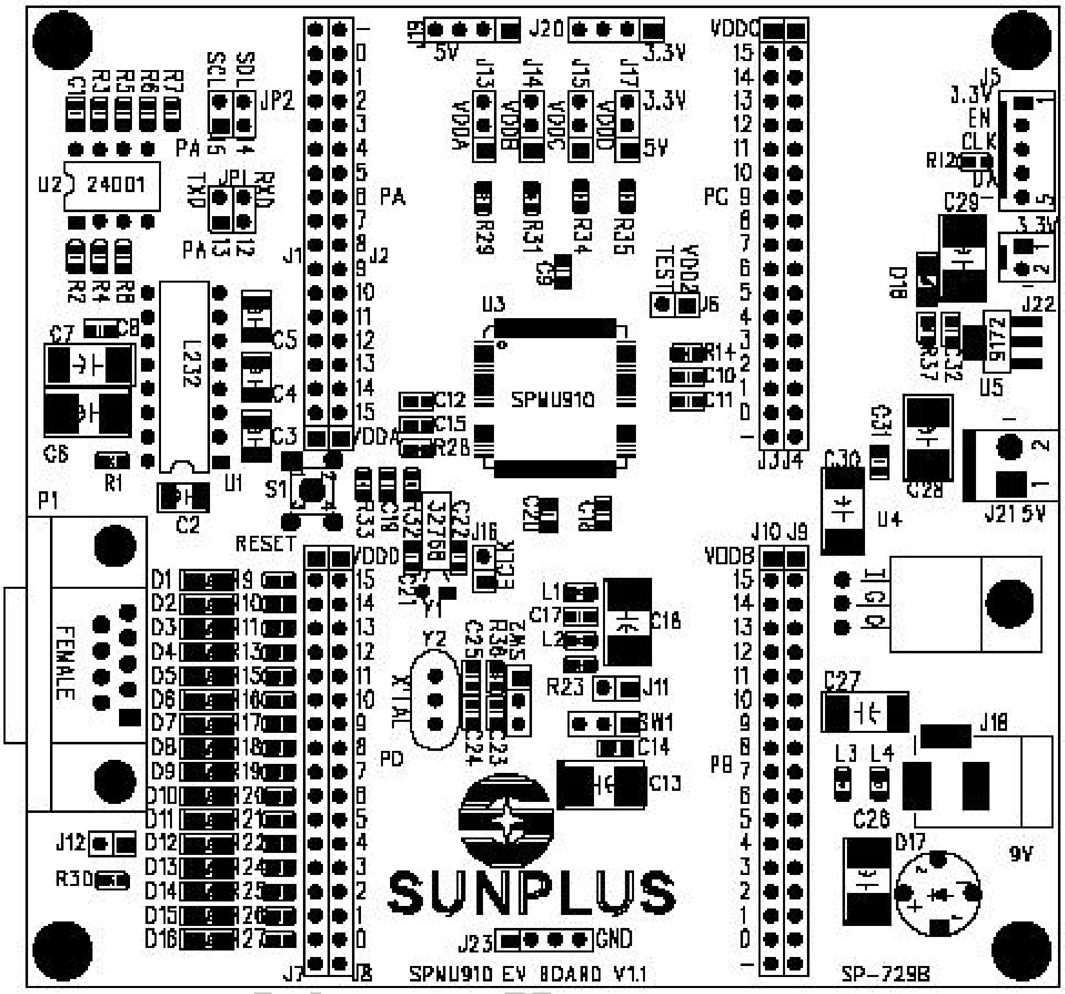
24 SPMC70xxx Development System

24.1 Development system Setup diagram

Connection: PC parallel port / USB port → Sunplus Probe → SPMC EMU board



24.2SPMC701FM0 EMU board v1.1



SPMC701FM0 EMU Board Quick setup guide

1. Check the jumpers and switches on SPMC701FM0 EMU board.

J12:	Short,		
SW2:	Short 2,3 pin,		
SW1:	Short 2,3 pin		
J13:	Short 1.2 pin	to	select VDDA as 5V.
J14:	Short 1.2 pin	to	select VDDB as 5V.
J15:	Short 1.2 pin	to	select VDDC as 5V.
J17:	Short 1.2 pin	to	select VDDD as 5V.

2. Power on the SPMC701FM0 EMU board through one of the following jacks
   J18 (DC9V from adapter),
   J21 (5V from power supply),
3. Connect SPMC701FM0 EMU board via Sunplus probe to PC from LPT1 port or USB port. (J5)
4. Launch *m'nSP*™ IDE on PC, load project and start developing the project. Make sure that the IC body is chosen as SPMC701FM0 correctly.

Jumper / Switch Setup

Switch/Jump Name	Description	Note
D18	POWER LED.	
J1, J2	IOA0~IOA15	
J3, J4	IOC0~IOC15	
J7, J8	IOD0~IOD15	
J9, J10	IOB0~IOB15	
J5	ICE interface connector.	Connect to PC parallel port / USB port through Sunplus probe for program download.

SW1	Select AD top reference voltage input signal pin
	VREF2V: The 2V voltage output pin of the internal 2V regulator.
	AVDD: Analog VDD input.
	J11: External Voltage input for VEXTREF
		Short 1,2 pin
		VEXTREF = AVDD , if R23 is short
		VEXTREF = J11 input, R23 is open
		Short 2,3 pin
		VEXTREF = VREF2V

J11	External Voltage input for VEXTREF	When SW1 1,2 pin short and R23 12 open
J16	External Clock input	
SW2	Select Oscillator input	Short 1,2 pin as RC mode Short 2,3 pin as X'TAL mode
J18	Power input for power adapter.	9V ~ 12V > 500 uA.
J21	Power input for 5V level power supply. It will further go through the U5 to generate 3.3V to the VDD.	
J13	Jumper switch for I/O's power supply VDDA.	Short 1,2 pin as set VDDA=5V level Short 2,3 pin as set VDDA=3V level
J14	Jumper switch for I/O's power supply VDDB.	Short 1,2 pin as set VDDB=5V level Short 2,3 pin as set VDDB=3V level
J15	Jumper switch for I/O's power supply VDDC.	Short 1,2 pin as set VDDB=5V level Short 2,3 pin as set VDDB=3V level
J17	Jumper switch for I/O's power supply VDDD.	Short 1,2 pin as set VDDB=5V level Short 2,3 pin as set VDDB=3V level
P1	UART(PC RS232 Com port connector).	Also refer to the JP1 jumper
JP1	Select PA12,PA13 as UART pin or GPIO	
JP2	Select PA14,PA15 as SIO pin or GPIO	SIO is used to control U2 device
S1	RESET key	
J12	Power for LED display of PD	Short J12 to GND to supply LED power
U2	24C01/24C02/24C04/24C16	I2C Device
U1	RS232 Control Chip	For USRT Tx & Rx
Y1	32.768 X'TAL	
Y2	6M X'TAL	
J23	GND pins	
J19	5V pins	
J20	3.3V pins	

24.3SPMC701FM0 code downloading application circuit

SPMC701FM0 is a MTP type MCU, featuring its flash ROM. The purpose of this section is to show how user can use a minimal set of pins on SPMC701FM0 to download program code onto the SPMC701FM0 flash ROM.

In order to launch a download session, the SPMC701FM0 A needs to be connected to *m'nSP*™ IDE via ICE. The ICE on SPMC needs to be connected to *m'nSP*™ IDE via ICESDA, ICESCK and ICE pin. The VDD and VSS on the probe cable need to be supplied from an external power.

The voltage operating range on I/Os is determined by I/O power, VDD<A,B,C,D> . The valid voltage operating range on I/Os can be set via VDD<A,B,C,D> as user's wish. For simplicity of downloading only, user can use 3.3V for both VDD and VDD<A,B,C,D>

Minimal set of pins required for SPMC701FM01A downloading: 5 pads

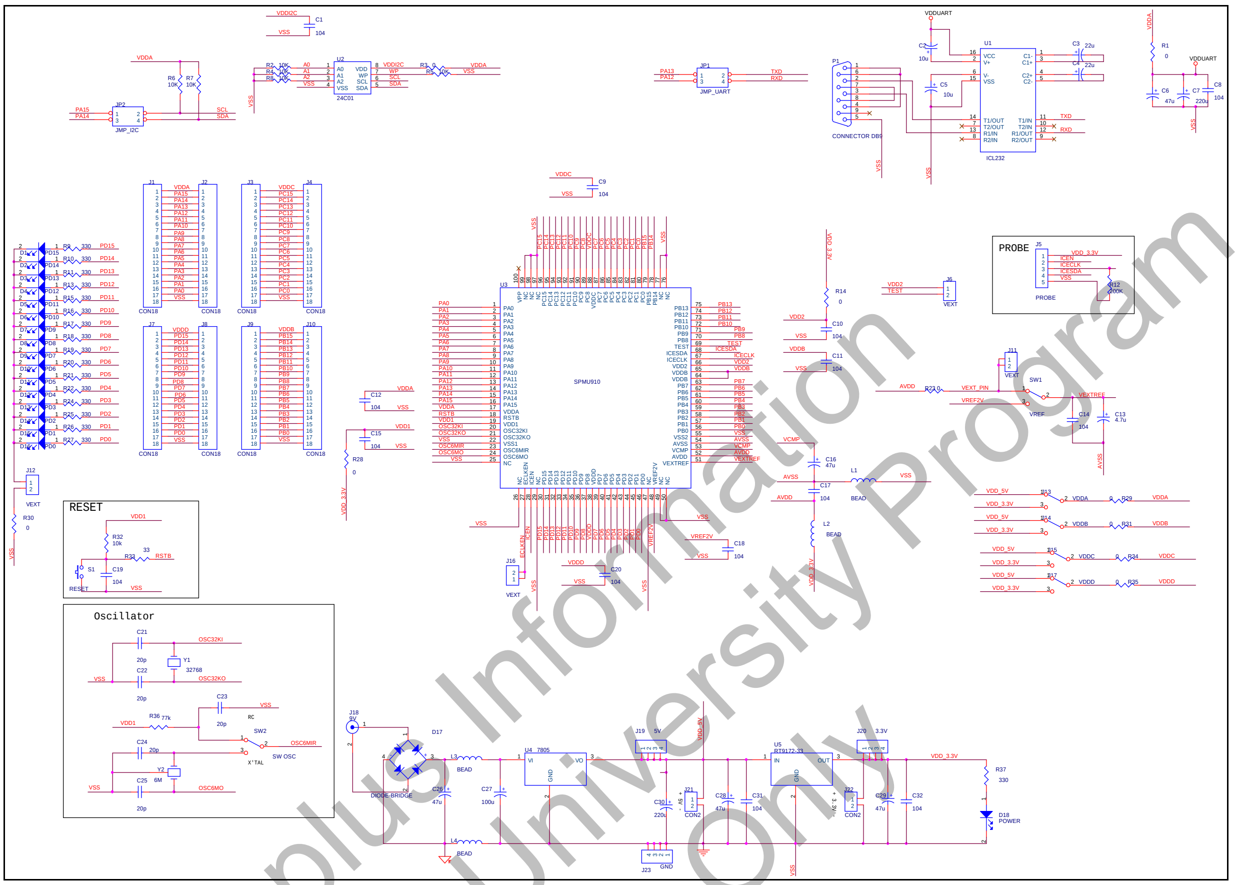
(VDD 3.3V, VSS: GND, VDDIO: 3.3V or 5V)

For Probe:

VDD
ICE
ICESCK
ICESDA
VSS

The pads above forms the minimal set necessary for program download.

24.4SPMC701FM0 EMU board v1.1 schematics



25 APPENDIX A. Reserved Words

The followings are reserved words for µ'nSP™ except the assembly language commands are not listed here.

1. Words with prefix of "__sn_" are reserved.
2. _BREAK, _FIQ, _RESET, _IRQ0, _IRQ1, _IRQ2, _IRQ3, _IRQ4, _IRQ5, _IRQ6, _IRQ7, __subf2, __blockcopy, __cmpf2, __common, __cvf2i1, __cvf2i2, __cvf2u1, __cvf2u2, __cvi1f2, __cvi2f2, __divf2, __divi1, __divi2, __divu1, __divu2, __lshiu1, __lshiu2, __modi1, __modi2, __modu1, __modu2, __mulf2, __muli2, __mulu1, __mulu2, __negf2, __rshi1, __rshi2, __rshu1, __rshu2, __addf2