



SUNPLUS

SPMC65X Family PROGRAMMING GUIDE

SPMC65X Family Programming Guide

V1.1 - May. 1, 2005



Technology for easy living

SPMC65X FAMILY PROGRAMMING GUIDE.....I**1 INTRODUCTION1-1**

1.1	DESCRIPTION	1-1
1.2	FEATURES.....	1-1
1.3	APPLICATIONS.....	1-4
1.4	DESIGN TIPS	1-4
1.5	RELATED APPLICATION NOTES AND LIBRARIES	1-4
1.6	REVISION HISTORY.....	1-4

2 ARCHITECTURE.....2-1

2.1	DESCRIPTION	2-1
2.2	BLOCK DIAGRAM.....	2-1
2.3	PIN ASSIGNMENT	2-2
2.3.1	<i>PLCC68</i>	2-2
2.3.2	<i>LQFP80</i>	2-3
2.4	PIN DESCRIPTION	2-4
2.4.1	<i>PLCC68</i>	2-5
2.4.2	<i>LQFP80</i>	2-8
2.5	DESIGN TIPS	2-12
2.6	RELATED APPLICATION NOTES AND LIBRARIES	2-12
2.7	REVISION HISTORY.....	2-12

3 CPU.....3-1

3.1	DESCRIPTION	3-1
3.2	BLOCK DIAGRAM.....	3-1
3.3	CPU REGISTERS	3-2
3.4	STATUS REGISTER (P).....	3-3
3.5	ADDRESSING MODES	3-6
3.5.1	<i>Immediate Addressing Mode</i>	3-6
3.5.2	<i>Absolute Addressing Mode</i>	3-7
3.5.3	<i>Absolute Indexed Addressing Mode</i>	3-7
3.5.4	<i>Zero Page Addressing Mode</i>	3-8
3.5.5	<i>Zero Page Indexed Addressing Mode</i>	3-8
3.5.6	<i>Implied Addressing Mode</i>	3-8

3.5.7	<i>Accumulator Addressing Mode</i>	3-8
3.5.8	<i>Indexed indirect Addressing Mode</i>	3-9
3.5.9	<i>Indirect Addressing Mode</i>	3-9
3.5.10	<i>Indirect Indexed Addressing Mode</i>	3-10
3.5.11	<i>Relative Addressing Mode</i>	3-10
3.6	DESIGN TIPS	3-12
3.7	RELATED APPLICATION NOTES AND LIBRARIES	3-12
3.8	REVISION HISTORY.....	3-12
4	MEMORY ORGANIZATION	4-1
4.1	DESCRIPTION	4-1
4.2	MEMORY MAP	4-1
4.3	PROGRAM MEMORY	4-2
4.4	DATA MEMORY	4-4
4.4.1	<i>Hardware Control Register</i>	4-5
4.4.2	<i>User RAM</i>	4-15
4.4.3	<i>Stack Area</i>	4-15
4.4.4	<i>Device Configuration Registers</i>	4-15
4.4.5	<i>User Information</i>	4-17
4.5	SPECIAL CONTROL REGISTER	4-17
4.6	DESIGN TIPS	4-18
4.7	RELATED APPLICATION NOTES AND LIBRARIES	4-18
4.8	REVISION HISTORY.....	4-18
5	RESET	5-1
5.1	DESCRIPTION	5-1
5.2	CONTROL REGISTER	5-2
5.2.1	<i>System Control Register (P_SYS_Ctrl, \$30)</i>	5-2
5.2.2	<i>Watchdog Timer Control Register (P_WDT_Ctrl, \$32)</i>	5-3
5.2.3	<i>LVR Option Register (P_LVR_Opt, \$36)</i>	5-4
5.3	OPERATION.....	5-5
5.3.1	<i>Power On Reset</i>	5-5
5.3.2	<i>External Reset</i>	5-5
5.3.3	<i>Low Voltage Reset</i>	5-6
5.3.4	<i>Watchdog Timer Reset</i>	5-7
5.3.5	<i>Illegal Address Reset</i>	5-8
5.4	DESIGN TIPS	5-9

5.5	RELATED APPLICATION NOTES AND LIBRARIES	5-11
5.6	REVISION HISTORY.....	5-11
6	INTERRUPTS.....	6-1
6.1	DESCRIPTION	6-1
6.2	CONTROL REGISTER	6-4
6.2.1	<i>Interrupt Flag Register 0 (P_INT_Flag0, \$0C) (R/W)</i>	6-4
6.2.2	<i>Interrupt Control Register 0 (P_INT_Ctrl0, \$0D) (R/W)</i>	6-5
6.2.3	<i>Interrupt Flag Register 1 (P_INT_Flag1, \$0E) (R/W)</i>	6-6
6.2.4	<i>Interrupt Control Register 1 (P_INT_Ctrl1, \$0F) (R/W)</i>	6-6
6.2.5	<i>Interrupt Flag Register 2 (P_INT_Flag2, \$26) (R/W)</i>	6-7
6.2.6	<i>Interrupt Control Register 2 (P_INT_Ctrl2, \$27) (R/W)</i>	6-8
6.2.7	<i>IRQ and Capture Option Register 0 (P IRQ_Opt0, \$33) (R/W)</i>	6-9
6.2.8	<i>IRQ and Capture Option Register 1 (P IRQ_Opt1, \$34) (R/W)</i>	6-9
6.3	OPERATION.....	6-11
6.4	DESIGN TIPS	6-12
6.5	RELATED APPLICATION NOTES AND LIBRARIES	6-18
6.6	REVISION HISTOR.....	6-18
7	CLOCK SOURCE.....	7-1
7.1	DESCRIPTION	7-1
7.2	OPERATION.....	7-1
7.3	DESIGN TIPS	7-3
7.4	RELATED APPLICATION NOTES AND LIBRARIES	7-3
7.5	REVISION HISTOY	7-4
8	I/O PORTS	8-6
8.1	DESCRIPTION	8-6
8.1.1	<i>Port A group</i>	8-8
8.1.2	<i>Port B Group</i>	8-8
8.1.3	<i>Port C Group</i>	8-9
8.1.4	<i>Port D Group</i>	8-9
8.1.5	<i>Port E Group</i>	8-10
8.1.6	<i>Port F Group</i>	8-10
8.1.7	<i>Slew Rate Control</i>	8-11
8.2	CONTROL REGISTER	8-12
8.2.1	<i>Port A Data Register (P IOA_Data, \$00)</i>	8-12
8.2.2	<i>Port A Direction Register (P IOA_Dir, \$04)</i>	8-12

8.2.3	<i>Port A Attribute Register (P_IOA_Attrib, \$08)</i>	8-12
8.2.4	<i>Port A Data Latch Buffer Register (P_IOA_Buf, \$59) (R/W)</i>	8-13
8.2.5	<i>Port B Data Register (P_IOB_Data, \$01)</i>	8-13
8.2.6	<i>Port B Direction Register (P_IOB_Dir, \$05)</i>	8-13
8.2.7	<i>Port B Attribute Register (P_IOB_Attrib, \$09)</i>	8-13
8.2.8	<i>Port B Data Latch Buffer Register (P_IOB_Buf, \$5A) (R/W)</i>	8-14
8.2.9	<i>Port C Data Register (P_IOC_Data, \$02)</i>	8-14
8.2.10	<i>Port C Direction Register (P_IOC_Dir, \$06)</i>	8-14
8.2.11	<i>Port C Attribute Register (P_IOC_Attrib, \$0A)</i>	8-14
8.2.12	<i>Port C Data Latch Buffer Register (P_IOC_Buf, \$5B) (R/W)</i>	8-15
8.2.13	<i>Port D Data Register (P_IOD_Data, \$03)</i>	8-15
8.2.14	<i>Port D Direction Register (P_IOD_Dir, \$07)</i>	8-15
8.2.15	<i>Port D Attribute Register (P_IOD_Attrib, \$0B)</i>	8-15
8.2.16	<i>Port D Data Latch Buffer Register (P_IOD_Buf, \$5C) (R/W)</i>	8-16
8.2.17	<i>Port E Data Register (P_IOE_Data, \$40) (R/W)</i>	8-16
8.2.18	<i>Port E Direction Register (P_IOE_Dir, \$42) (R/W)</i>	8-16
8.2.19	<i>Port E Attribute Register (P_IOE_Attrib, \$44) (R/W)</i>	8-16
8.2.20	<i>Port E Data Latch Buffer Register (P_IOE_Buf, \$5D) (R/W)</i>	8-17
8.2.21	<i>Port F Data Register (P_IOF_Data, \$41) (R/W)</i>	8-17
8.2.22	<i>Port F Direction Register (P_IOF_Dir, \$43) (R/W)</i>	8-17
8.2.23	<i>Port F Attribute Register (P_IOF_Attrib, \$45) (R/W)</i>	8-17
8.2.24	<i>Port F Data Latch Buffer Register (P_IOF_Buf, \$5E) (R/W)</i>	8-18
8.2.25	<i>Slew Rate Control Register (P_IO_Opt, \$35)</i>	8-18
8.3	OPERATION.....	8-18
8.4	DESIGN TIPS	8-20
8.5	RELATED APPLICATION NOTES AND LIBRARIES	8-21
8.6	REVISION HISTORY.....	8-22
9	TIMERS.....	9-23
9.1	DESCRIPTION	9-23
9.2	CONTROL REGISTER	9-24
9.2.1	<i>Timer0_1 Control Register 0 (P_TMR0_1_Ctrl0, \$11) (R/W)</i>	9-24
9.2.2	<i>Timer0_1 Control Register 1 (P_TMR0_1_Ctrl1, \$12) (R/W)</i>	9-24
9.2.3	<i>Timer0 Count Low Byte Register (P_TMR0_Count, \$13) (R)</i>	9-25
9.2.4	<i>Timer0 Preload Low Byte Register (P_TMR0_Preload, \$13) (W)</i>	9-25
9.2.5	<i>Timer0 Count High Byte Register (P_TMR0_CountHi, \$14) (R)</i>	9-25
9.2.6	<i>Timer0 Preload High Byte Register (P_TMR0_PreloadHi, \$14) (W)</i>	9-25

9.2.7	<i>Timer1 Count Low Byte Register (P_TMR1_Count, \$15) (R)</i>	9-26
9.2.8	<i>Timer1 Preload Low Byte Register (P_TMR1_Preload, \$15) (W)</i>	9-26
9.2.9	<i>Timer1 Count High Byte Register (P_TMR1_CountHi, \$16) (R)</i>	9-26
9.2.10	<i>Timer1 Preload High Byte Register (P_TMR1_PreloadHi, \$16) (W)</i>	9-26
9.2.11	<i>Timer2_3 Control Register 0 (P_TMR2_3_Ctrl0, \$18) (R/W)</i>	9-26
9.2.12	<i>Timer2_3 Control Register 1 (P_TMR2_3_Ctrl1, \$19) (R/W)</i>	9-27
9.2.13	<i>Timer2 Count Low Byte Register (P_TMR2_Count, \$1A) (R)</i>	9-28
9.2.14	<i>Timer2 Preload Low Byte Register (P_TMR2_Preload, \$1A) (W)</i>	9-28
9.2.15	<i>Timer2 Count High Byte Register (P_TMR2_CountHi, \$1B) (R)</i>	9-28
9.2.16	<i>Timer2 Preload High Byte Register (P_TMR2_PreloadHi, \$1B) (W)</i>	9-28
9.2.17	<i>Timer3 Count Low Byte Register (P_TMR3_Count, \$1C) (R)</i>	9-28
9.2.18	<i>Timer3 Preload Low Byte Register (P_TMR3_Preload, \$1C) (W)</i>	9-28
9.2.19	<i>Timer3 Count High Byte Register (P_TMR3_CountHi, \$1D) (R)</i>	9-29
9.2.20	<i>Timer3 Preload High Byte Register (P_TMR3_PreloadHi, \$1D) (W)</i>	9-29
9.2.21	<i>Timer4_5 Control Register 0 (P_TMR4_5_Ctrl0, \$1F) (R/W)</i>	9-29
9.2.22	<i>Timer4_5 Control Register 1 (P_TMR4_5_Ctrl1, \$20) (R/W)</i>	9-30
9.2.23	<i>Timer4 Count Low Byte Register (P_TMR4_Count, \$21) (R)</i>	9-30
9.2.24	<i>Timer4 Preload Low Byte Register (P_TMR4_Preload, \$21) (W)</i>	9-30
9.2.25	<i>Timer4 Count High Byte Register (P_TMR4_CountHi, \$22) (R)</i>	9-31
9.2.26	<i>Timer4 Preload High Byte Register (P_TMR4_PreloadHi, \$22) (W)</i>	9-31
9.2.27	<i>Timer5 Count Low Byte Register (P_TMR5_Count, \$23) (R)</i>	9-31
9.2.28	<i>Timer5 Preload Low Byte Register (P_TMR5_Preload, \$23) (W)</i>	9-31
9.2.29	<i>Timer5 Count High Byte Register (P_TMR5_CountHi, \$24) (R)</i>	9-31
9.2.30	<i>Timer5 Preload High Byte Register (P_TMR5_PreloadHi, \$24) (W)</i>	9-31
9.3	OPERATION	9-31
9.3.1	<i>8-bit Timer</i>	9-33
9.3.2	<i>16-bit Timer</i>	9-35
9.4	TIMER INTERRUPT	9-37
9.5	DESIGN TIPS	9-39
9.6	RELATED APPLICATION NOTES AND LIBRARIES	9-41
9.7	REVISION HISTORY	9-42

10 CAPTURE 10-43

10.1	DESCRIPTION	10-43
10.2	CONTROL REGISTER	10-44
10.2.1	<i>Timer0_1 Control Register 0 (P_TMR0_1_Ctrl0, \$11) (R/W)</i>	10-44
10.2.2	<i>Timer0_1 Control Register 1 (P_TMR0_1_Ctrl1, \$12) (R/W)</i>	10-44

10.2.3	<i>Timer0 Capture Width Low Byte Register (P_TMR0_Cap, \$13) (R/W)</i>	10-45
10.2.4	<i>Timer0 Capture Width High Byte Register (P_TMR0_CapHi, \$14) (R/W)</i>	10-45
10.2.5	<i>Timer0 Capture Cycle Byte Register (P_TMR0_CapCycle8, \$14) (R/W)</i>	10-46
10.2.6	<i>Timer1 Capture Width Low Byte Register (P_TMR1_Cap, \$15) (R/W)</i>	10-46
10.2.7	<i>Timer1 Capture Width High Byte Register (P_TMR1_CapHi, \$16) (R/W)</i>	10-46
10.2.8	<i>Timer1 Capture Cycle Byte Register (P_TMR1_CapCycle8, \$16) (R/W)</i>	10-46
10.2.9	<i>Timer2_3 Control Register 0 (P_TMR2_3_Ctrl0, \$18) (R/W)</i>	10-47
10.2.10	<i>Timer2_3 Control Register 1 (P_TMR2_3_Ctrl1, \$19) (R/W)</i>	10-47
10.2.11	<i>Timer2 Capture Width Low Byte Register (P_TMR2_Cap, \$1A) (R/W)</i>	10-48
10.2.12	<i>Timer2 Capture Width High Byte Register (P_TMR2_CapHi, \$1B) (R/W)</i>	10-48
10.2.13	<i>Timer2 Capture Cycle Byte Register (P_TMR2_CapCycle8, \$1B) (R/W)</i>	10-49
10.2.14	<i>Timer3 Capture Width Low Byte Register (P_TMR3_Cap, \$1C) (R/W)</i>	10-49
10.2.15	<i>Timer3 Capture Width High Byte Register (P_TMR3_CapHi, \$1D) (R/W)</i>	10-49
10.2.16	<i>Timer3 Capture Cycle Byte Register (P_TMR3_CapCycle8, \$1D) (R/W)</i>	10-49
10.2.17	<i>Timer4_5 Control Register 0 (P_TMR4_5_Ctrl0, \$1F) (R/W)</i>	10-50
10.2.18	<i>Timer4_5 Control Register 1 (P_TMR4_5_Ctrl1, \$20) (R/W)</i>	10-50
10.2.19	<i>Timer4 Capture Width Low Byte Register (P_TMR4_Cap, \$21) (R/W)</i>	10-51
10.2.20	<i>Timer4 Capture Width High Byte Register (P_TMR4_CapHi, \$22) (R/W)</i>	10-51
10.2.21	<i>Timer4 Capture Cycle Byte Register (P_TMR4_CapCycle8, \$22) (R/W)</i>	10-52
10.2.22	<i>Timer5 Capture Width Low Byte Register (P_TMR5_Cap, \$23) (R/W)</i>	10-52
10.2.23	<i>Timer5 Capture Width High Byte Register (P_TMR5_CapHi, \$24) (R/W)</i>	10-52
10.2.24	<i>Timer5 Capture Cycle Byte Register (P_TMR5_CapCycle8, \$24) (R/W)</i>	10-52
10.2.25	<i>Timer5 Capture Cycle Low Byte Register (P_TMR5_CapCycleLo, \$25) (R)</i>	10-53
10.2.26	<i>Timer5 Capture Cycle High Byte Register (P_TMR5_CapCycleHi, \$5F) (R)</i>	10-53
10.2.27	<i>Capture Control Register (P_CAP_Ctrl, \$58) (R/W)</i>	10-53
10.3	OPERATION	10-55
10.3.1	<i>8-bit Capture</i>	10-55
10.3.2	<i>16-bit Capture</i>	10-59
10.4	CAPTURE INTERRUPT	10-60
10.5	DESIGN TIPS	10-61
10.6	RELATED APPLICATION NOTES AND LIBRARIES	10-64
10.7	REVISION HISTORY	10-65
11	COMPARE	11-1
11.1	DESCRIPTION	11-1
11.2	CONTROL REGISTER	11-2
11.2.1	<i>Timer0_1 Control Register 0 (P_TMR0_1_Ctrl0, \$11) (R/W)</i>	11-2

11.2.2	<i>Timer0_1 Control Register 1 (P_TMR0_1_Ctrl1, \$12) (R/W)</i>	11-2
11.2.3	<i>Timer0 Compare Low Byte Register (P_TMR0_Comp, \$13) (R/W)</i>	11-3
11.2.4	<i>Timer0 Compare High Byte Register (P_TMR0_CompHi, \$14) (R/W)</i>	11-3
11.2.5	<i>Timer1 Compare Low Byte Register (P_TMR1_Comp, \$15) (R/W)</i>	11-4
11.2.6	<i>Timer1 Compare High Byte Register (P_TMR1_CompHi, \$16) (R/W)</i>	11-4
11.2.7	<i>Timer2_3 Control Register 0 (P_TMR2_3_Ctrl0, \$18) (R/W)</i>	11-4
11.2.8	<i>Timer2_3 Control Register 1 (P_TMR2_3_Ctrl1, \$19) (R/W)</i>	11-5
11.2.9	<i>Timer2 Compare Low Byte Register (P_TMR2_Comp, \$1A) (R/W)</i>	11-6
11.2.10	<i>Timer2 Compare High Byte Register (P_TMR2_CompHi, \$1B) (R/W)</i>	11-6
11.2.11	<i>Timer3 Compare Low Byte Register (P_TMR3_Comp, \$1C) (R/W)</i>	11-6
11.2.12	<i>Timer3 Compare High Byte Register (P_TMR3_CompHi, \$1D) (R/W)</i>	11-6
11.2.13	<i>Timer4_5 Control Register 0 (P_TMR4_5_Ctrl0, \$1F) (R/W)</i>	11-7
11.2.14	<i>Timer4_5 Control Register 1 (P_TMR4_5_Ctrl1, \$20) (R/W)</i>	11-7
11.2.15	<i>Timer4 Compare Low Byte Register (P_TMR4_Comp, \$21) (R/W)</i>	11-8
11.2.16	<i>Timer4 Compare High Byte Register (P_TMR4_CompHi, \$22) (R/W)</i>	11-8
11.2.17	<i>Timer5 Compare Low Byte Register (P_TMR5_Comp, \$23) (R/W)</i>	11-9
11.2.18	<i>Timer5 Compare High Byte Register (P_TMR5_CompHi, \$24) (R/W)</i>	11-9
11.3	OPERATION	11-9
11.3.1	<i>8-bit Compare</i>	11-9
11.3.2	<i>16-bit Compare</i>	11-10
11.4	COMPARE INTERRUPT	11-12
11.5	DESIGN TIPS	11-12
11.6	RELATED APPLICATION NOTES AND LIBRARIES	11-14
11.7	REVISION HISTORY	11-15
12	PWM	12-1
12.1	DESCRIPTION	12-1
12.2	CONTROL REGISTER	12-3
12.2.1	<i>Timer0_1 Control Register 0 (P_TMR0_1_Ctrl0, \$11) (R/W)</i>	12-3
12.2.2	<i>Timer0_1 Control Register 1 (P_TMR0_1_Ctrl1, \$12) (R/W)</i>	12-3
12.2.3	<i>Timer0 PWM Period Register (P_TMR0_PWMPeriod, \$13) (R/W)</i>	12-4
12.2.4	<i>Timer0 PWM Duty Register (P_TMR0_PWM Duty, \$14) (R/W)</i>	12-4
12.2.5	<i>Timer1 PWM Period Register (P_TMR1_PWMPeriod, \$15) (R/W)</i>	12-4
12.2.6	<i>Timer1 PWM Duty Period Register (P_TMR1_DutyPeriod, \$16) (R/W)</i>	12-5
12.2.7	<i>Timer1 PWM Duty Register (P_TMR1_PWM Duty, \$17) (R/W)</i>	12-5
12.2.8	<i>Timer2_3 Control Register 0 (P_TMR2_3_Ctrl0, \$18) (R/W)</i>	12-5
12.2.9	<i>Timer2_3 Control Register 1 (P_TMR2_3_Ctrl1, \$19) (R/W)</i>	12-6

12.2.10	<i>Timer2 PWM Period Register (P_TMR2_PWMPeriod, \$1A) (R/W)</i>	12-6
12.2.11	<i>Timer2 PWM Duty Register (P_TMR2_PWMduty, \$1B) (R/W)</i>	12-7
12.2.12	<i>Timer3 PWM Period Register (P_TMR3_PWMPeriod, \$1C) (R/W)</i>	12-7
12.2.13	<i>Timer3 PWM Duty Period Register (P_TMR3_DutyPeriod, \$1D) (R/W)</i>	12-7
12.2.14	<i>Timer3 PWM Duty Register (P_TMR3_PWMduty, \$1E) (R/W)</i>	12-7
12.2.15	<i>Timer4_5 Control Register 0 (P_TMR4_5_Ctrl0, \$1F) (R/W)</i>	12-8
12.2.16	<i>Timer4_5 Control Register 1 (P_TMR4_5_Ctrl1, \$20) (R/W)</i>	12-8
12.2.17	<i>Timer4 PWM Period Register (P_TMR4_PWMPeriod, \$21) (R/W)</i>	12-9
12.2.18	<i>Timer4 PWM Duty Register (P_TMR4_PWMduty, \$22) (R/W)</i>	12-9
12.2.19	<i>Timer5 PWM Period Low Byte Register (P_TMR5_PWMPeriodLo, \$23) (R/W)</i>	12-9
12.2.20	<i>Timer5 PWM Period High Byte Register (P_TMR5_PWMPeriodHi, \$24) (R/W)</i>	12-10
12.2.21	<i>Timer5 PWM Duty Low Byte Register (P_TMR5_PWMdutyLo, \$25) (R/W)</i>	12-10
12.2.22	<i>Timer5 PWM Duty High Byte Register (P_TMR5_PWMdutyHi, \$5F) (R/W)</i>	12-10
12.3	OPERATION	12-10
12.3.1	<i>8-bit PWM (Type I)</i>	12-10
12.3.2	<i>12-bit PWM (Type II)</i>	12-13
12.3.3	<i>16-bit PWM (Type III)</i>	12-17
12.4	PWM INTERRUPT	12-20
12.5	DESIGN TIPS	12-20
12.6	RELATED APPLICATION NOTES AND LIBRARIES	12-23
12.7	REVISION HISTORY	12-24

13 A/D CONVERTER 13-1

13.1	DESCRIPTION	13-1
13.2	CONTROL REGISTER	13-2
13.2.1	<i>ADC Control Register 0 (P_AD_Ctrl0, \$28) (R/W)</i>	13-2
13.2.2	<i>ADC Control Register 1 (P_AD_Ctrl1, \$29) (R/W)</i>	13-2
13.2.3	<i>ADC Control Register 2 (P_AD_Ctrl2, \$2A) (R/W)</i>	13-3
13.2.4	<i>ADC Conversion High Data Register (P_AD_DataHi, 2B)</i>	13-4
13.2.5	<i>ADC Conversion Low Data Register (P_AD_DataLo, \$2C) (R/W)</i>	13-4
13.3	OPERATION	13-4
13.4	ADC INTERRUPT	13-6
13.5	DESIGN TIPS	13-7
13.6	RELATED APPLICATION NOTES AND LIBRARIES	13-9
13.7	REVISION HISTORY	13-9

14 D/A CONVERTER 14-1

14.1	DESCRIPTION	14-1
14.2	CONTROL REGISTER	14-1
14.2.1	<i>DAC Control Register (P_DA_Ctrl, \$55) (R/W)</i>	14-1
14.2.2	<i>DAC Conversion Low Data Register (P_DA_DataLo, \$56) (R/W)</i>	14-1
14.2.3	<i>DAC Conversion High Data Register (P_DA_DataHi, \$57) (R/W)</i>	14-2
14.3	OPERATION	14-2
14.4	DAC INTERRUPT	14-2
14.5	DESIGN TIPS	14-2
14.6	RELATED APPLICATION NOTES AND LIBRARIES	14-5
14.7	REVISION HISTORY.....	14-5
15	COMPARATOR	15-1
15.1	DESCRIPTION	15-1
15.2	CONTROL REGISTER	15-1
15.2.1	<i>Comparator Control Register (P_CMP_Ctrl, \$2E) (R/W)</i>	15-1
15.3	OPERATION	15-2
15.4	COMPARATOR INTERRUPT	15-3
15.5	DESIGN TIPS	15-3
15.6	RELATED APPLICATION NOTES AND LIBRARIES	15-5
15.7	REVISION HISTORY.....	15-6
16	INTERVAL TIMER AND BUZZER.....	16-1
16.1	DESCRIPTION	16-1
16.2	CONTROL REGISTER	16-2
16.2.1	<i>Interval Timer and Buzzer Control Register (P_BUZ_Ctrl, \$2D) (R/W)</i>	16-2
16.3	OPERATION	16-3
16.3.1	<i>Interval Timer.</i>	16-3
16.3.2	<i>Buzzer.</i>	16-4
16.4	TIME BASE INTERVAL INTERRUPT	16-5
16.5	DESIGN TIPS	16-5
16.6	RELATED APPLICATION NOTES AND LIBRARIES	16-8
16.7	REVISION HISTORY.....	16-8
17	WATCH DOG TIMER.....	17-1
17.1	DESCRIPTION	17-1
17.2	CONTROL REGISTER	17-1
17.2.1	<i>Watch Dog Timer Control Register (P_WDT_Ctrl, \$32) (R/W)</i>	17-1
17.2.2	<i>Watch Dog Timer Clear Register (P_WDT_Clr, \$10) (W)</i>	17-2



17.3	OPERATION	17-3
17.4	WATCH DOG INTERRUPT	17-3
17.5	DESIGN TIPS	17-4
17.6	RELATED APPLICATION NOTES AND LIBRARIES	17-6
17.7	REVISION HISTORY	17-7

18 SPI 18-1

18.1	DESCRIPTION	18-1
18.2	APPLICATION CIRCUIT	18-2
18.3	CONTROL REGISTER	18-3
18.3.1	<i>SPI Control Register0 (P_SPI_Ctrl0, \$38)</i>	18-3
18.3.2	<i>SPI Control Register1 (P_SPI_Ctrl1, \$39)</i>	18-3
18.3.3	<i>SPI State Status Register (P_SPI_Status, \$3A)</i>	18-4
18.3.4	<i>SPI Transmission Buffer Register0 (P_SPI_TxData, \$3B)</i>	18-5
18.3.5	<i>SPI Receive Buffer Register0 (P_SPI_RxData, \$3C)</i>	18-5
18.4	OPERATION	18-5
18.4.1	<i>Master Mode</i>	18-5
18.4.2	<i>Slave Mode</i>	18-9
18.5	SPI INTERRUPT	18-11
18.6	DESIGN TIPS	18-11
18.7	RELATED APPLICATION NOTES AND LIBRARIES	18-15
18.8	REVISION HISTORY	18-16

19 UART 19-1

19.1	DESCRIPTION	19-1
19.2	APPLICATION CIRCUIT	19-2
19.3	CONTROL REGISTER	19-4
19.3.1	<i>UART Control Register (P_UART_Ctrl, \$46) (R/W)</i>	19-4
19.3.2	<i>UART Baud Rate Divider Register (P_UART_Baud, \$47) (R/W)</i>	19-5
19.3.3	<i>UART Status Register (P_UART_Status, \$48) (R/W)</i>	19-5
19.3.4	<i>UART Transmit or Receive Data Register (P_UART_Data, \$49) (R/W)</i>	19-6
19.4	OPERATION	19-6
19.5	UART INTERRUPT	19-8
19.6	DESIGN TIPS	19-8
19.7	RELATED APPLICATION NOTES AND LIBRARIES	19-10
19.8	REVISION HISTORY	19-11

20 IIC 20-1

20.1	DESCRIPTION	20-1
20.2	APPLICATION CIRCUIT	20-2
20.3	CONTROL REGISTER	20-2
20.3.1	<i>IIC Bus Control Register (P_IIC_Ctrl, \$4A) (R/W)</i>	20-2
20.3.2	<i>IIC Bus Status Register (P_IIC_Status, \$4B) (R/W)</i>	20-3
20.3.3	<i>IIC Bus Data Register (P_IIC_Data, \$4C) (R/W)</i>	20-4
20.3.4	<i>IIC Bus Address Register (P_IIC_Address, \$4D) (R/W)</i>	20-4
20.4	OPERATION	20-5
20.4.1	<i>Initializing and Terminating Data Transfer</i>	20-5
20.4.2	<i>Addressing IIC Devices</i>	20-6
20.4.3	<i>Transfer Acknowledge</i>	20-6
20.4.4	<i>General Call Address Support</i>	20-7
20.4.5	<i>Bus Arbitration</i>	20-7
20.4.6	<i>Slave mode</i>	20-7
20.4.7	<i>Master mode</i>	20-8
20.5	IIC INTERRUPT	20-9
20.6	DESIGN TIPS	20-9
20.7	RELATED APPLICATION NOTES AND LIBRARIES	20-14
20.8	REVISION HISTORY	20-14

21 POWER SAVING MODE 21-1

21.1	DESCRIPTION	21-1
21.2	CONTROL REGISTER	21-2
21.2.1	<i>Power Saving Mode Control Register (P_Mode_Ctrl, \$31) (R/W)</i>	21-2
21.2.2	<i>Watch Dog Timer Control Register (P_WDT_Ctrl, \$32) (R/W)</i>	21-3
21.3	OPERATION	21-3
21.3.1	<i>STOP mode</i>	21-3
21.3.2	<i>Release the STOP mode</i>	21-4
21.3.3	<i>Halt mode</i>	21-7
21.3.4	<i>Release the Halt mode</i>	21-8
21.3.5	<i>Minimizing Current Consumption</i>	21-9
21.4	POWER SAVING MODE INTERRUPT	21-10
21.5	DESIGN TIPS	21-10
21.6	RELATED APPLICATION NOTES AND LIBRARIES	21-15
21.7	REVISION HISTORY	21-16

22 DEVELOPMENT TOOL 22-1



22.1	DESCRIPTION	22-1
22.2	FORTIS IDE	22-1
22.2.1	<i>Key Feature</i>	22-1
22.2.2	<i>Installation</i>	22-2
22.2.3	<i>Interface</i>	22-7
22.2.4	<i>Quick Start</i>	22-11
22.2.5	<i>Create Project</i>	22-11
22.2.6	<i>Project Setting</i>	22-11
22.2.7	<i>Operation of Project File</i>	22-17
22.2.8	<i>Build Project</i>	22-18
22.2.9	<i>Text Editor</i>	22-19
22.2.10	<i>How To Use Debugger</i>	22-23
22.2.11	<i>Toolbar</i>	22-33
22.3	IN-CIRCUIT EMULATOR.....	22-34
22.3.1	<i>Hardware Architecture of Emulation Board</i>	22-34
22.3.2	<i>Operation Description</i>	22-36
22.3.3	<i>Real-time serial programming</i>	22-38
22.4	RELATED APPLICATION NOTES AND LIBRARIES	22-38
22.5	REVISION HISTORY	22-39
23	IN-CIRCUIT PROGRAMMER	23-40
23.1	DESCRIPTION	23-40
23.2	Q-WRITER.....	23-40
23.2.1	<i>General Description</i>	23-40
23.2.2	<i>Integrated Hardware Design</i>	23-40
23.2.3	<i>Installing and Starting</i>	23-40
23.2.4	<i>How to use Q-Writer</i>	23-43
23.2.5	<i>System Message</i>	23-51
23.3	OTP/MTP WRITER	23-53
23.3.1	<i>General Description</i>	23-53
23.3.2	<i>Hardware</i>	23-53
23.3.3	<i>Adapt-board</i>	23-53
23.3.4	<i>Software Tool</i>	23-53
23.3.5	<i>Function Description</i>	23-56
23.3.6	<i>Serial Number Function</i>	23-58
23.3.7	<i>Important Notice</i>	23-65
23.4	THIRD-PARTY PROGRAMMER	23-67

23.4.1	<i>Introduction</i>	23-67
23.4.2	<i>Feature & Benefits</i>	23-67
23.4.3	<i>Specifications</i>	23-68
23.5	RELATED APPLICATION NOTES	23-69
23.6	REVISION HISTORY	23-69
24	ASSEMBLY TOOL	24-1
24.1	DESCRIPTION	24-1
24.1.1	<i>Assembler (xasm8.exe)</i>	24-1
24.1.2	<i>Linker (xlink8.exe)</i>	24-1
24.1.3	<i>Library Maker (xlib8.exe)</i>	24-1
24.2	CODING FLOW	24-2
24.3	ASSEMBLER	24-3
24.3.1	<i>Filename extension</i>	24-3
24.3.2	<i>Operators</i>	24-3
24.3.3	<i>Assembly Language</i>	24-12
24.3.4	<i>Assembler Directives</i>	24-15
24.3.5	<i>Macros</i>	24-34
24.4	LINKER	24-36
24.4.1	<i>General Description</i>	24-36
24.4.2	<i>Filename Extensions</i>	24-36
24.4.3	<i>Load/Run-Time Address</i>	24-37
24.4.4	<i>Search Order</i>	24-37
24.4.5	<i>Address Relocation</i>	24-37
24.4.6	<i>Symbol Table Output Formats</i>	24-37
24.4.7	<i>Linker Features</i>	24-38
24.4.8	<i>Interactive mode</i>	24-38
24.4.9	<i>Interactive Mode Options</i>	24-39
24.4.10	<i>Command Line Mode</i>	24-40
24.4.11	<i>Linker Script File Mode</i>	24-41
24.4.12	<i>Offsets</i>	24-43
24.4.13	<i>Indirect Linking</i>	24-43
24.4.14	<i>Linker Script File Mode</i>	24-45
24.4.15	<i>Symbol Table</i>	24-45
24.4.16	<i>Executable Code File Format</i>	24-45
24.4.17	<i>Map File</i>	24-45
24.5	LIBRARY MAKER	24-47



24.6	ERROR MESSAGE.....	24-47
24.6.1	<i>Assembler Errors</i>	24-47
24.6.2	<i>Assembler Warning</i>	24-54
24.6.3	<i>Linker Errors</i>	24-56
24.6.4	<i>Library Maker Errors</i>	24-58
24.7	MOST COMMONLY MISTAKES.....	24-59
24.8	RELATED APPLICATION NOTES AND LIBRARIES	24-63
24.9	REVISION HISTORY.....	24-64
25	INSTRUCTION SET	25-1
25.1	DESCRIPTION	25-1
25.2	FORMAT OF ASSEMBLY LANGUAGE INSTRUCTION.....	25-1
25.3	INSTRUCTIONS	25-2
25.4	SUMMARY OF AVAILABLE INSTRUCTION SETS.....	25-20
25.5	REVISION HISTORY.....	25-23
26	INCLUDE FILE.....	26-1
26.1	DESCRIPTION	26-1
26.2	INCLUDE FILE	26-1
26.3	REVISION HISTORY.....	26-22



1 Introduction

1.1 Description

SPMC65X family are series of Sunplus'es micro-controllers. All of them include many chips with different features and can be emulated by a SPMC65X family's emulation system.

SPMC65 CPU core is a SUNPLUS new powerful 8-bit CPU derived from 6502 core. SUNPLUS has developed a new powewrful CPU called SPMC65 CPU base on 6502 architecture. Besides 6502 instructions, SPMC65 CPU also supported bit operations and is compatible with 6502. The SPMC65X family feature a SPMC65 CPU core, programmable general I/Os, varieties of ROM and RAM sizes, 8-bit/16-bit Timer(s), powerful CCP(s) (Capture/Compare/PWM) rather than common specification and watchdog to prevent system crash from abnormal operations. The advanced sub-micron CMOS process technology ensures the SPMC65X working at high performance, Industrial EMC capability and reliability.

In addition to the above features, some SPMC65X provides high sink current with slow output transition port pins, multi external interrupt pins, Low Voltage Reset (LVR) function, ADC, PWM, communication interface and multiple oscillator options. The SPMC65X is suitable in general-purpose controller, Industrial equipments, computer peripheral controller, home appliance...etc.

ECMC653 is an emulation chip of micro-controller. It equips with 8-bit SPMC65 CPU core, 928 bytes SRAM for data memory and 16k bytes SRAM for program memory that substitutes the role of ROM at the real chip. It also combines one time base interval timer, one watchdog timer, six 16-bit timers, and maximun nine channels of ADC. In order to reduce whole emulation board cost, this chip also equips with one EPROM programming series interface. In the debugging environment, it also equips with one PC trace that could help user to speed up the debugging program and find the implicit program error easily.

1.2 Features

The features introduced are based on ECMC653.

■ **Memories**

- 16K bytes SRAM for program memory
- 928 bytes SRAM for data memory

■ **SPMC65 CPU Core**

- 182 instructions
- Up to 8MHz clock operation.
- Support bit operation instruction.

n I/O Ports

- multifunction bi-directional I/O line
- Users can set the corresponding registers to select pull-up/down resistors, or outputs.

n Interrupt Management

- Interrupt option : NMI or IRQ for external interrupts.
- 6 channel of external interrupts
- internal interrupts

n Reset Management

- Enhanced reset system

n Clock Management

- 3 Clock sources : RC-oscillation, crystal input options and external clock input.
- Clock output capability

n Power Management

- 2 power saving modes : Halt、Stop

n 2 Analog Peripheral

- Maximun 9-channel of 10-bit resolution A/D converter, or 8-channel 10-bit resolution A/D converter with one top reference voltage.
- Low voltage reset with 4.0v or 2.5v option
- One channel of 10-bit D/A with 3.3mA output
- Two channel of comparator with 30mv hysteresis window.

n Three channels of 16-bit Timers (type I)

- Timers, Event counter mode
- Capture(8-bit with width/cycle measurement, 16-bit with width measurement)
- 16-bit compare mode
- 8-bit PWM output

n Two channels of 16-bit Timers (type II)

- Timers, Event counter mode
- Capture(8-bit with width/cycle measurement, 16-bit with width measurement)
- 16-bit compare output
- 12-bit PWM output

n One channel of 16-bit Timers (type III)

- Timers, Event counter mode
- Capture(16-bit with width/cycle measurement)
- 16-bit compare output
- 16-bit PWM output

n Time Base Interval Timer

- frequency: 1Hz to 62.5kHz @Fsys=8MHz

- **Buzzer output**
 - frequency: 1kHz to 2MHz @Fsys=8MHz
- **Configurable watchdog timer**
- **Serial Bus interface**
 - SPI bus
 - UART bus
 - IIC bus

1.3 Applications

- Small Home Appliance
 - Rice cooker, inductance cooker, microwave oven, refrigerator, washing machine, air condition, charger...
- General purpose MCU with I/O controller or A/D application
- Industrial equipments
- computer peripheral controller

1.4 Design Tips

None

1.5 Related Application Notes and Libraries

This section lists some application notes and libraries about various usage of SPMC65X family. The application notes may help user develop quickly and offer the concepts about the usage of this topic. Sunplus also supports some powerful libraries about general operation, mathematics, communication, etc. The purpose of library is to help user speed up development. The current application notes and libraries related to Introduction are listed below.

Application Note:

Title	Application Note Series Number
Coding method	AN_O0328.DOC

Library:

Title	Application Note Series Number
SPMC65X series software basics manual	AN_O0100.DOC
SPMC65X series software advances 1 manual (data processing operation)	AN_O0101.DOC
SPMC65X series software advances 2 manual (mathematical operation)	AN_O0102.DOC

1.6 Revision History

Revision	Date	Remark
V0.1	03/31/2005	First edition

2 ARCHITECTURE

2.1 Description

A block diagram of SPMC65X architecture is shown in Figure 2-1. SPMC65X architecture is based on Von Neumann architecture, which supports inseparable bus structure for program space and data space. SPMC65X architecture could be divided into three parts that are core, peripherals and special features. The core pertains to basic feature that are required to make device operation, such as CPU, Memory, etc. Peripherals are the features that add a differentiation from a MCU, such as Timer, Capture, etc. Special features are function to help system increase reliability or decrease cost, such as Low Voltage Reset, Power Saving Mode, etc.

2.2 Block Diagram

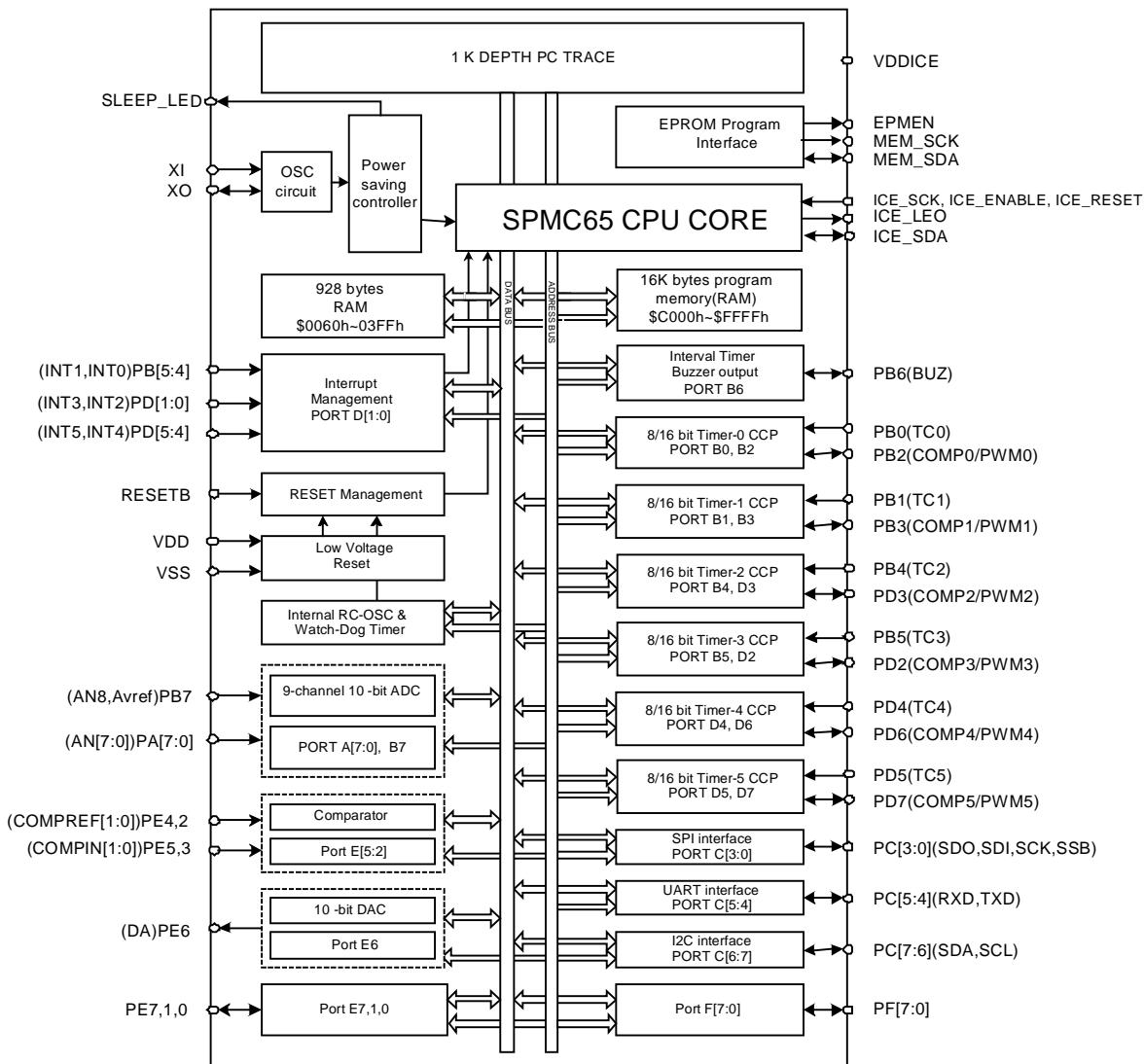
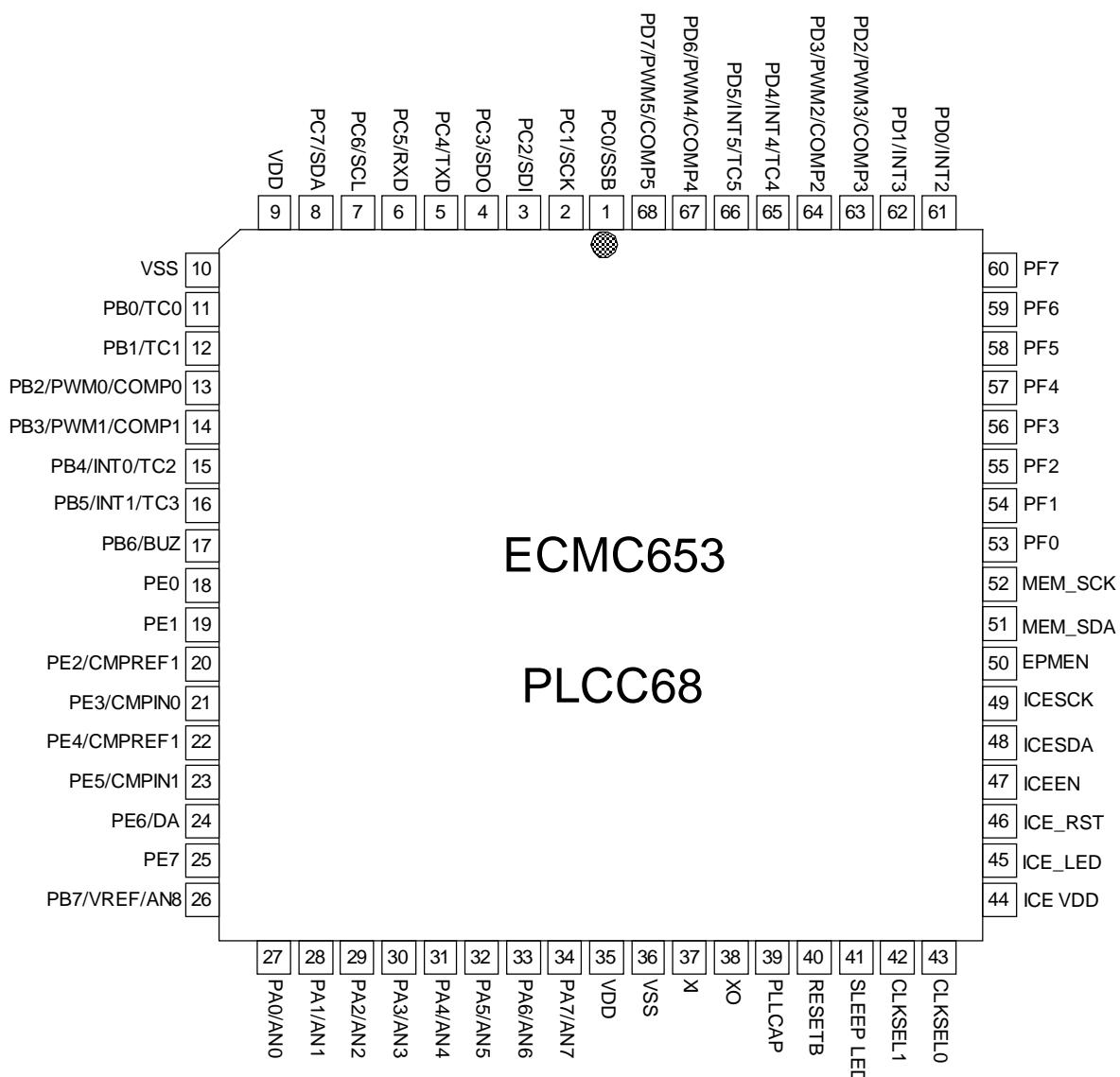


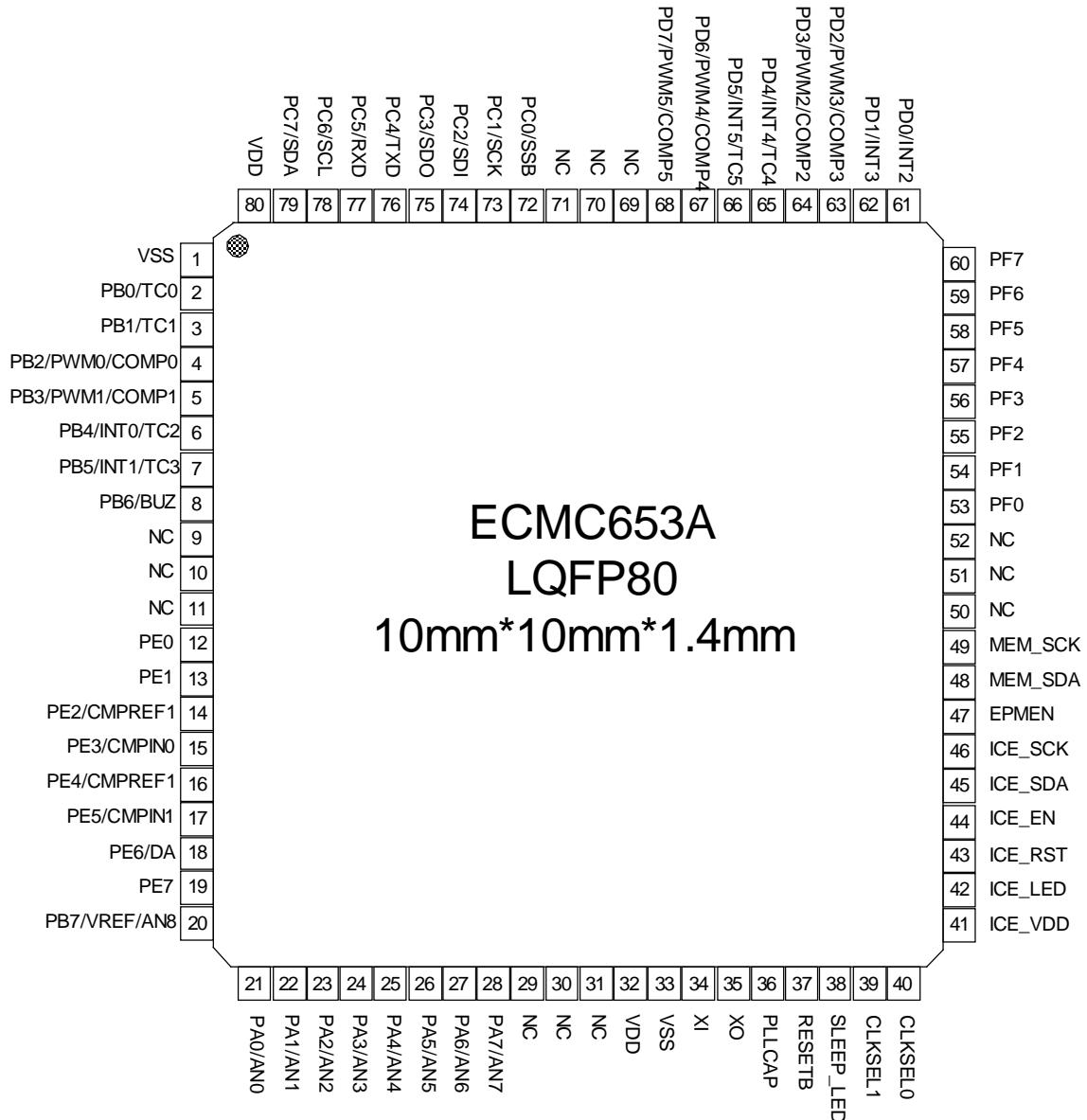
Figure 2-1 SPMC65X Block Diagram

2.3 Pin Assignment

2.3.1 PLCC68



2.3.2 LQFP80



2.4 Pin Description

Type : I = Input, O = Output, S = Supply

Input Level : A = Dedicated analog input

Input/Output Level : C = CMOS 0.3VDD/0.7VDD

Output Level : HS = 20mA high sink

Port and control configuration :

- Input : float = floating, WPD = weak pull-down, WPU = weak pull-up, Int = interrupt, ana = analog
- Output : OD = open drain, PP = push-pull

2.4.1 PLCC68

Pin No	Pin Name	Type	Level		Port / Control					Main Function (after reset)	Alternate Function	
			Input	Output	Input			Output				
					Float	WPD	WPU	Int	ana	OD	PP	
9,35	VDD	S										Main power supply
10,36	VSS	S										Ground
37	XI/R	I										<i>Crystal In or RC-Oscillation In Input or External Clock Input.</i> An external resistive pull-up is used to connect with internal OSC circuitry for generating the internal clock and the related time base in RC-Oscillation mode. It will be connected with external crystal for a crystal oscillation circuitry in crystal mode.
38	XO	O										<i>Crystal Output.</i> It will be connected with external crystal for a crystal oscillation circuitry in crystal mode.
39	NC											
34	PA7/AN7	I/O	C			X		X	X	Port A7	ADC analog input	
33	PA6/AN6	I/O	C			X		X	X	Port A6	ADC analog input	
32	PA5/AN5	I/O	C			X		X	X	Port A5	ADC analog input	
31	PA4/AN4	I/O	C			X		X	X	Port A4	ADC analog input	
30	PA3/AN3	I/O	C			X		X	X	Port A3	ADC analog input	
29	PA2/AN2	I/O	C			X		X	X	Port A2	ADC analog input	
28	PA1/AN1	I/O	C			X		X	X	Port A1	ADC analog input	
27	PA0/AN0	I/O	C			X		X	X	Port A0	ADC analog input	
26	PB7/VREF/AN8	I/O	C			X		X	X	Port B7	ADC analog input or ADC top reference voltage	
17	PB6/BUZ	I/O	C			X	X	X	X	Port B6	Buzzer output	
16	PB5/INT1/TC3	I/O	C			INT1	X	X	X	Port B5	External interrupt 1 input/ Capture3 event input to Timer3 or External event input using Timer3 as the counter.	
15	PB4/INT0/TC2	I/O	C			INT0	X	X	X	Port B4	External interrupt 0 input/ Capture2 event input to Timer2 or External event input using Timer2 as the counter.	



14	PB3/PWM1/COMP1	I/O	C			X	X	X	X	Port B3	Timer1 compare output / PWM output
13	PB2/PWM0/COMP0	I/O	C			X	X	X	X	Port B2	Timer0 compare output / PWM output
12	PB1/TC1	I/O	C			X	X	X	X	Port B1	Capture1 event input to Timer1 / External event input using Timer1 as the counter.
11	PB0/TC0	I/O	C			X	X	X	X	Port B0	Capture0 event input to Timer0 / External event input using Timer0 as the counter.
8	PC7/SDA	I/O	C			X	X	X	X	Port C7	IIC data input /output
7	PC6/SCL	I/O	C			X	X	X	X	Port C6	IIC clock input
6	PC5/RXD	I/O	C			X	X	X	X	Port C5	UART Receive signal
5	PC4/TXD	I/O	C			X	X	X	X	Port C4	UART Transfer signal
4	PC3/SDO	I/O	C			X	X	X	X	Port C3	SPI data output
3	PC2/SDI	I/O	C			X	X	X	X	Port C2	SPI data input
2	PC1/SCK	I/O	C			X	X	X	X	Port C1	SPI clock output / clock input
1	PC0/SSB	I/O	C			X	X	X	X	Port C0	SPI chip select
68	PD7/PWM5/COMP5	I/O	C			X	X	X	X	Port D7	Timer5 compare output / PWM output
67	PD6/PWM4/COMP4	I/O	C			X	X	X	X	Port D6	Timer4 compare output / PWM output
66	PD5/INT5/ TC5	I/O	C			INT5	X	X	X	Port D5	External interrupt 5 input/ Capture5 event input to Timer5 or External event input using Timer5 as the counter.
65	PD4/INT4/ TC4	I/O	C			INT4	X	X	X	Port D4	External interrupt 4 input/ Capture4 event input to Timer4 or External event input using Timer4 as the counter.
64	PD3/PWM2/COMP2	I/O	C			X	X	X	X	Port D3	Timer2 compare output / PWM output
63	PD2/PWM3/COMP3	I/O	C			X	X	X	X	Port D2	Timer3 compare output / PWM output
62	PD1/INT3	I/O	C			INT3	X	X	X	Port D1	External interrupt 3 input
61	PD0/INT2	I/O	C			INT2	X	X	X	Port D0	External interrupt 2 input
25	PE7	I/O	C			X	X	X	X	Port E7	
24	PE6/ DA	I/O	C			X	X	X	X	Port E6	D/A output
23	PE5/ CMPIN1	I/O	C			X	X	X	X	Port E5	Comparator1 input
22	PE4/ CMPREF1	I/O	C			X	X	X	X	Port E4	Comparator1 reference input
21	PE3/CMPIN0	I/O	C			X	X	X	X	Port E3	Comparator0 input

20	PE2/CMPREF0	I/O	C			X	X	X	X	Port E2	Comparator0 reference input
19	PE1	I/O	C			X	X	X	X	Port E1	
18	PE0	I/O	C			X	X	X	X	Port E0	
60	PF7	I/O	C			X	X	X	X	Port F7	
59	PF6	I/O	C			X	X	X	X	Port F6	
58	PF5	I/O	C			X	X	X	X	Port F5	
57	PF4	I/O	C			X	X	X	X	Port F4	
56	PF3	I/O	C			X	X	X	X	Port F3	
55	PF2	I/O	C			X	X	X	X	Port F2	
54	PF1	I/O	C			X	X	X	X	Port F1	
53	PF0	I/O	C			X	X	X	X	Port F0	
40	RESETB	I/O	C	X	X						External reset signal
50	EPMEN	O									EPROM mode output pin. Board designer use it to turn on 13/6v.
52	MEM_SCK	I	C								Accessing external EEPROM series clock input
51	MEM_SDA	I/O	C	C							Accessing external EEPROM series data input/output
44	ICE_VDD	S									ICE probe interface voltage, typical value is 3.3v.
49	ICE_SCK	I									ICE clock input.
48	ICE_SDA	I/O									ICE data input and output.
47	ICE_EN	I									ICE enable pin
46	ICE_RST	I									ICE reset pin
45	ICE_LED	O									ICE active LED indicator
41	SLEEP_LED	O									Sleep mode indicator
42	CLKSEL1	I									Clock Select Pin1
43	CLKSEL0	I									Clock Select Pin0

Note: CLKSEL[1:0]: 00:Crystal or resonator oscillator

01:RC oscillator

10:External clock

11:Reserve

Total: 68 pins

2.4.2 LQFP80

Pin No	Pin Name	Type	Level		Port / Control						Main Function (after reset)	Alternate Function	
			Input	Output	Input				Output				
					Float	WPD	WPU	Int	ana	OD	PP		
32,80	VDD	S										Main power supply	
1,33	VSS	S										Ground	
34	XI/R	I										<i>Crystal In or RC-Oscillation In Input or External Clock Input.</i> An external resistive pull-up is used to connect with internal OSC circuitry for generating the internal clock and the related time base in RC-Oscillation mode. It will be connected with external crystal for a crystal oscillation circuitry in crystal mode.	
35	XO	O										<i>Crystal Output.</i> It will be connected with external crystal for a crystal oscillation circuitry in crystal mode.	
36	NC												
28	PA7/AN7	I/O	C				X		X	X	Port A7	ADC analog input	
27	PA6/AN6	I/O	C				X		X	X	Port A6	ADC analog input	
26	PA5/AN5	I/O	C				X		X	X	Port A5	ADC analog input	
25	PA4/AN4	I/O	C				X		X	X	Port A4	ADC analog input	
24	PA3/AN3	I/O	C				X		X	X	Port A3	ADC analog input	
23	PA2/AN2	I/O	C				X		X	X	Port A2	ADC analog input	
22	PA1/AN1	I/O	C				X		X	X	Port A1	ADC analog input	
21	PA0/AN0	I/O	C				X		X	X	Port A0	ADC analog input	
20	PB7/VREF/AN8	I/O	C				X		X	X	Port B7	ADC analog input or ADC top reference voltage	
8	PB6/BUZ	I/O	C				X	X	X	X	Port B6	Buzzer output	
7	PB5/INT1/TC3	I/O	C				INT1	X	X	X	Port B5	External interrupt 1 input/ Capture3 event input to Timer3 or External event input using Timer3 as the counter.	

6	PB4/INT0/TC2	I/O	C			INT0	X	X	X	Port B4	External interrupt 0 input/ Capture2 event input to Timer2 or External event input using Timer2 as the counter.
5	PB3/PWM1/COMP1	I/O	C			X	X	X	X	Port B3	Timer1 compare output / PWM output
4	PB2/PWM0/COMP0	I/O	C			X	X	X	X	Port B2	Timer0 compare output / PWM output
3	PB1/TC1	I/O	C			X	X	X	X	Port B1	Capture1 event input to Timer1 / External event input using Timer1 as the counter.
2	PB0/TC0	I/O	C			X	X	X	X	Port B0	Capture0 event input to Timer0 / External event input using Timer0 as the counter.
79	PC7/SDA	I/O	C			X	X	X	X	Port C7	IIC bus data input /output
78	PC6/SCL	I/O	C			X	X	X	X	Port C6	IIC bus clock input
77	PC5/RXD	I/O	C			X	X	X	X	Port C5	UART Receive signal
76	PC4/TXD	I/O	C			X	X	X	X	Port C4	UART Transfer signal
75	PC3/SDO	I/O	C			X	X	X	X	Port C3	SPI data output
74	PC2/SDI	I/O	C			X	X	X	X	Port C2	SPI data input
73	PC1/SCK	I/O	C			X	X	X	X	Port C1	SPI clock output / clock input
72	PC0/SSB	I/O	C			X	X	X	X	Port C0	SPI chip select
68	PD7/PWM5/COMP5	I/O	C			X	X	X	X	Port D7	Timer5 compare output / PWM output
67	PD6/PWM4/COMP4	I/O	C			X	X	X	X	Port D6	Timer4 compare output / PWM output
66	PD5/INT5/ TC5	I/O	C			INT5	X	X	X	Port D5	External interrupt 5 input/ Capture5 event input to Timer5 or External event input using Timer5 as the counter.
65	PD4/INT4/ TC4	I/O	C			INT4	X	X	X	Port D4	External interrupt 4 input/ Capture4 event input to Timer4 or External event input using Timer4 as the counter.



64	PD3/PWM2/COMP2	I/O	C			X	X	X	X	Port D3	Timer2 compare output / PWM output
63	PD2/PWM3/COMP3	I/O	C			X	X	X	X	Port D2	Timer3 compare output / PWM output
62	PD1/INT3	I/O	C			INT3	X	X	X	Port D1	External interrupt 3 input
61	PD0/INT2	I/O	C			INT2	X	X	X	Port D0	External interrupt 2 input
19	PE7	I/O	C			X	X	X	X	Port E7	
18	PE6/ DA	I/O	C			X	X	X	X	Port E6	D/A output
17	PE5/ CMPIN1	I/O	C			X	X	X	X	Port E5	Comparator1 input
16	PE4/ CMPREF1	I/O	C			X	X	X	X	Port E4	Comparator1 reference input
15	PE3/CMPIN0	I/O	C			X	X	X	X	Port E3	Comparator0 input
14	PE2/CMPREF0	I/O	C			X	X	X	X	Port E2	Comparator0 reference input
13	PE1	I/O	C			X	X	X	X	Port E1	
12	PE0	I/O	C			X	X	X	X	Port E0	
60	PF7	I/O	C			X	X	X	X	Port F7	
59	PF6	I/O	C			X	X	X	X	Port F6	
58	PF5	I/O	C			X	X	X	X	Port F5	
57	PF4	I/O	C			X	X	X	X	Port F4	
56	PF3	I/O	C			X	X	X	X	Port F3	
55	PF2	I/O	C			X	X	X	X	Port F2	
54	PF1	I/O	C			X	X	X	X	Port F1	
53	PF0	I/O	C			X	X	X	X	Port F0	
37	RESETB	I/O	C	X	X						External reset signal
47	EPMEN	O									EPROM mode output pin. Board designer use it to turn on 13/6v.
49	MEM_SCK	I	C								Accessing external EPROM series clock input
48	MEM_SDA	I/O	C	C							Accessing external EPROM series data input/output
41	ICE_VDD	S									ICE probe interface voltage, typical value is 3.3v.
46	ICE_SCK	I									ICE clock input.
45	ICE_SDA	I/O									ICE data input and output.
44	ICE_EN	I									ICE enable pin

43	ICE_RST	I									ICE reset pin
42	ICE_LED	O									ICE active LED indicator
38	SLEEP_LED	O									Sleep mode indicator
39	CLKSEL1	I									Clock Select Pin1
40	CLKSEL0	I									Clock Select Pin0

Note: CLKSEL[1:0]: 00:Crystal or resonator oscillator

01:RC oscillator

10:External clock

11:Reserve

Total: 80 pins

2.5 Design Tips

None

2.6 Related Application Notes and Libraries

This section lists some application notes and libraries about various usage of SPMC65X family. The application notes may help user develop quickly and offer the concepts about the usage of this topic. Sunplus also supports some powerful libraries about general operation, mathematics, communication, etc. The purpose of library is to help user speed up development. The current application notes and libraries related to Architecture are listed below.

Application Note:

Title	Application Note Series Number
--------------	---------------------------------------

No associated application notes at this time.

Library:

Title	Application Note Series Number
SPMC65X series software basics manual	AN_O0100.DOC
SPMC65X series software advances 1 manual (data processing operation)	AN_O0101.DOC
SPMC65X series software advances 2 manual (mathematical operation)	AN_O0102.DOC

2.7 Revision History

Revision	Date	Remark
V0.1	03/31/2005	First edition

3 CPU

3.1 Description

The microprocessors of SPMC65X family are high performance processors equipped with 6 internal registers : Accumulator, Program Counter, X Register, Y Register, Stack Pointer and Processor Status Register. They also support 182 instructions. SPMC65X family are fully static CMOS design. The oscillation frequency (F_{sys}) could be varied up to 8.0MHz depends on the application needs. Figure 3-1 shows CPU Block Diagram.

3.2 Block Diagram

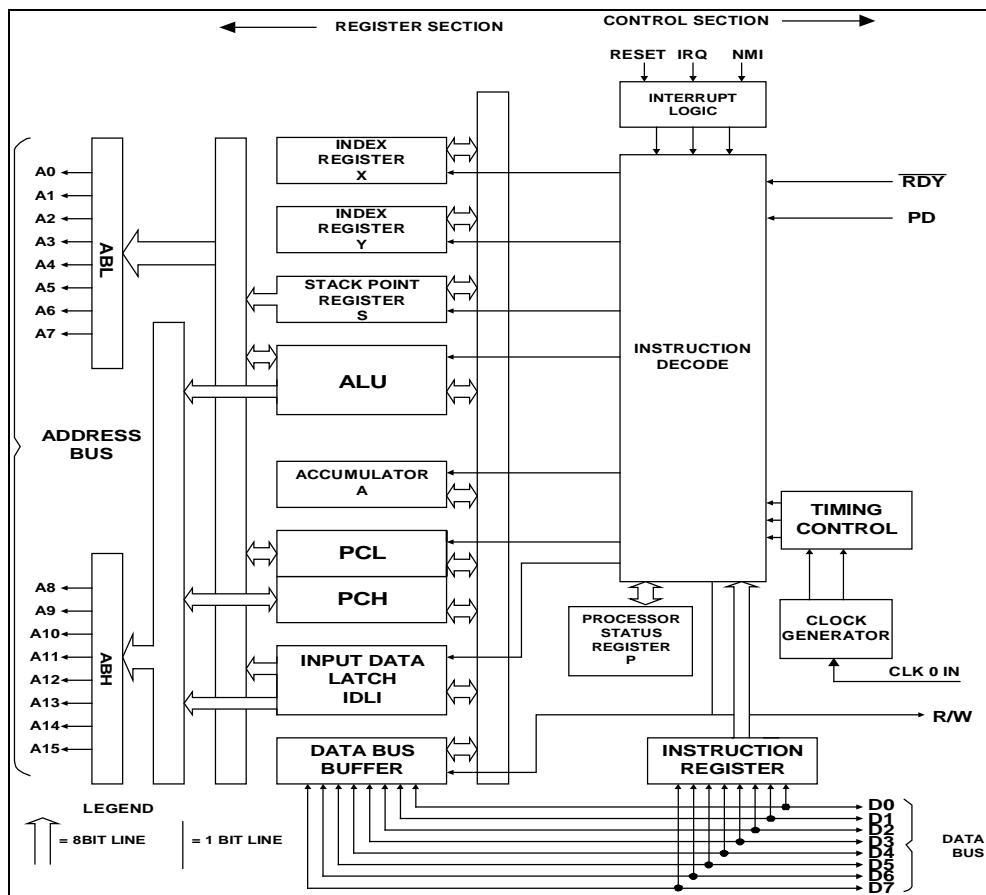


Figure 3-1 CPU Block Diagram

3.3 CPU Registers

The SPMC65X family have six registers that are the Program Counter (PC), an Accumulator (A), two index registers (X, Y), the Stack Pointer (SP), and the Status register (P). The program Counter consists of a 16-bit register. Others are 8-bit registers. All of CPU's registers shown in Figure 3-2 are described as following.

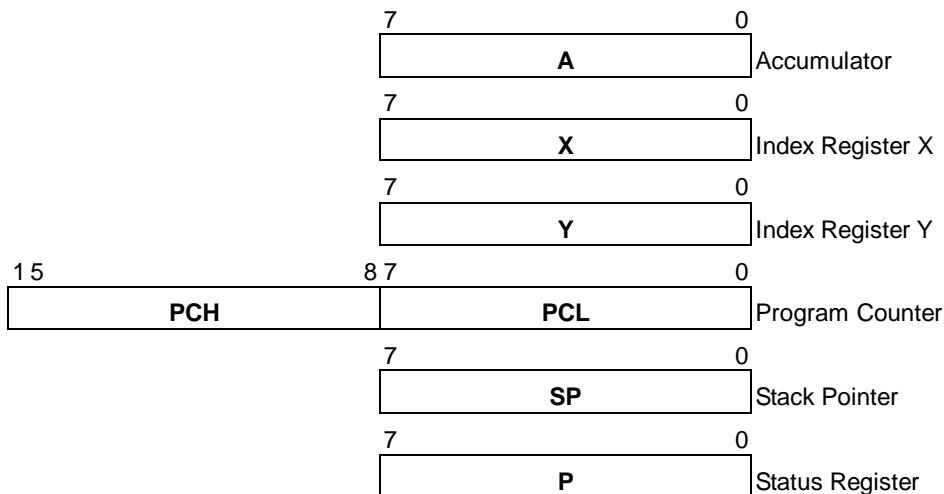


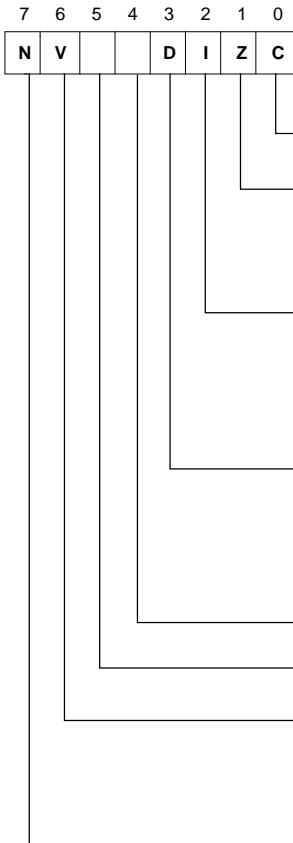
Figure 3-2 System Registers

The registers are described as shown in Table 3-1.

Register	Size	Description
Accumulator (A)	8 Bit	Accumulator is the only register that can be used for arithmetic or logic operation such as ADD, SUB, AND, OR and EOR and store the result in it.
Index Register X	8 Bit	X is an index register that can be used as a memory buffer, a offset, or a counter.
Index Register Y	8 Bit	Y is an index register that can be used as a memory buffer, a offset, or a counter.
Program Counter (PC)	16 Bit	PC is a 16-bit register. Program Counter points to an address location where an instruction is held and waits to be executed by CPU next. When CPU fetches one instruction to execute, PC is incremented to the next location in memory from which the next instruction to be executed will be taken unless a branch is occurred that will lead PC points to the specified address location.
Stack Pointer (SP)	8 Bit	Stack Pointer is an 8-bit register. Normally, SP is used to indicate the location in the stack to be accessed.(Push or Pop).
Status Register (P)	8 Bit	Status Register usually offers information on result of previous instruction executed.

Table 3-1 CPU Registers

3.4 Status Register (P)



Carry Flag, If Carry flag is set, C=1

Zero Flag

If arithmetic or logic operation results to zero, Z is set to 1;
Otherwise, Z=0

Interrupt Disable Flag

If interrupt disable flag is set, I=1, CPU will ignore interrupt signal.
If interrupt disable flag is clear, I=0, CPU will accept interrupt signal.

This is Decimal Mode Flag

If D=1, CPU will work in Decimal Mode.
Otherwise, in Binary Mode.

Not Used

Not Used

Overflow flag

If Overflow is set, V=1
If Overflow is clear, V=0

Negative Flag

If arithmetic or logic operation results to negative, N= 1
Otherwise, N=0.

I Note: Not all instructions affect Status Register. A detailed instruction description will be discussed in Chapter 25 instruction manual.

X, Y register: In Address mode, X, Y registers can be used as index registers or buffer registers. These register contents are added to the specified address, which become the actual address. Some operations such as increment, decrement, comparison and data transfer function can be used in X and Y registers.

Accumulator: The Accumulation is the 8-bit general-purpose register, which can be operated with transfer, temporary saving, condition judgment, etc.

Stack point: SPMC65X family have an 8-bit-wide register used to indicate the location in the stack to be accessed (Push or Pop) when a subroutine call or interrupt occurs.

When subroutine call is executed or an interrupt occurs is accept, the value of stack point is updated automatically.

However, if the value of stack point is used in excess of stack area permitted by the data memory allocating configuration, the Illegal address reset would be occurred.

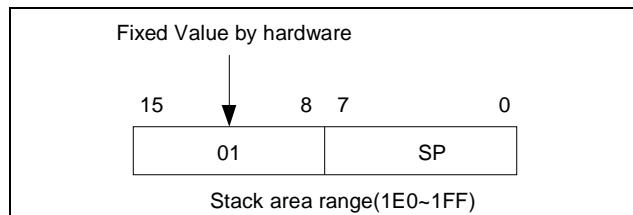


Figure 3-3 Stack Point Register

[Example 3-1]: Initialized stack point value

Idx	#C_STACK_BOTTOM	; Initial stack pointer at \$FF
txs		;Transfer to stack point

Program counter (PC): The Program Counter is a 16-bit wide register. It consists of two 8-bit registers which registers are PCH and PCL. This register indicates the address of next instruction to be executed. In Reset state, the content of program counter is stored with FFFCH.

Status register (P): The 8-bit status register contains the interrupt mask and 6 flags representative of the result of the instruction just executed. This register can also be handled by the PHP and PLP instructions. These bits can be individually controlled by specific instructions. The detailed descriptions are shown in following description.

.. Negative flag bit

This flag indicated the bit7 status of the result of a data or arithmetic operation. User can use this bit to do some operations such as branch condition.

.. Overflow flag bit

This flag indicated whether the overflow has occurred in arithmetic operation. When the result of an addition or subtraction is over +127 or less -128, this overflow bit is set to '1'.

.. Decimal mode flag

This flag indicated what mode is operated by arithmetic operation. SPMC65X family has two operation modes that are binary mode and decimal mode for arithmetic operation. User can use the instruction to alternate them.

.. Interrupt disable flag

This bit can enable or disable all interrupt except NMI interrupt source. If this bit is set to '1', CPU will ignore interrupt signal. Contrary, if this bit is set to '0', CPU will accept interrupt signal.

.. Zero flag

This flag indicated the result of a data or arithmetic operation. If the result is equal to zero, the Zero flag is set to '1'. Contrary, this bit is set to '0' by other value.

.. Carry flag

This bit is set to '1 if the result of addition operation generates a carry, or if the result of subtraction doesn't generate a borrowing. In addition, some Shift Instructions or Rotate Instructions also change this bit.

3.5 Addressing Modes

The SPMC65X family's CPUs use eleven addressing modes:

- Immediate Addressing Mode
- Absolute Addressing Mode
- Absolute Indexed Addressing Mode
- Zero Page Addressing mode
- Zero Page Indexed Addressing Mode
- Implied Addressing Mode
- Accumulator Addressing Mode
- Indexed Indirect Addressing Mode
- Indirect Addressing Mode
- Indirect Indexed Addressing Mode
- Relative Addressing Mode

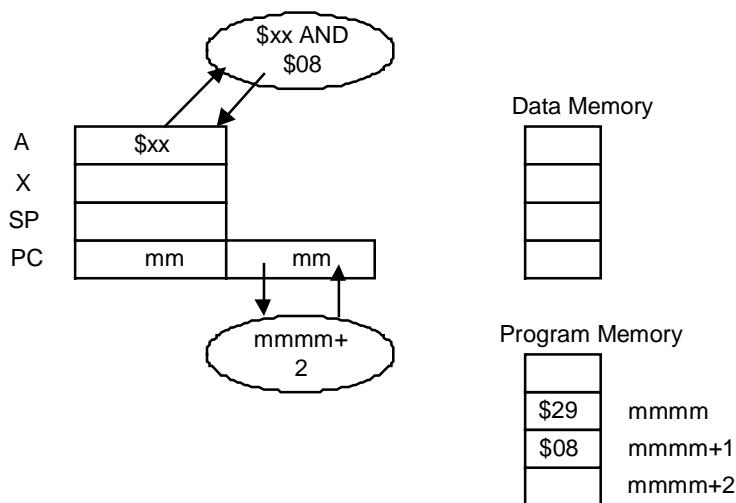
3.5.1 Immediate Addressing Mode

There is one byte in an immediate addressing mode.

Operation: **OP-code #dd**

[Example 3-2]: immediate addressing mode

AND	#\$08
-----	-------



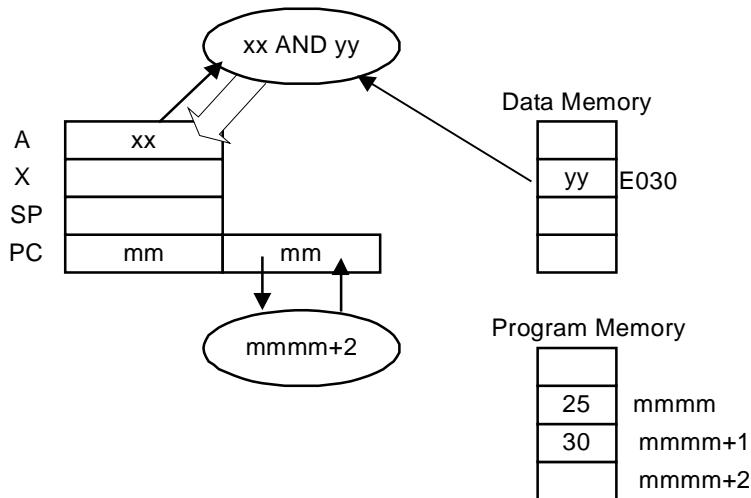
3.5.2 Absolute Addressing Mode

The absolute addressing mode uses two bytes (adr 16) to specify a memory address. The adr 16 may be the address of a byte of data or the beginning address for the next instruction.

Operation: **OP-code** **Adr16**

[Example 3-3]: absolute addressing mode

AND	\$E030
-----	--------



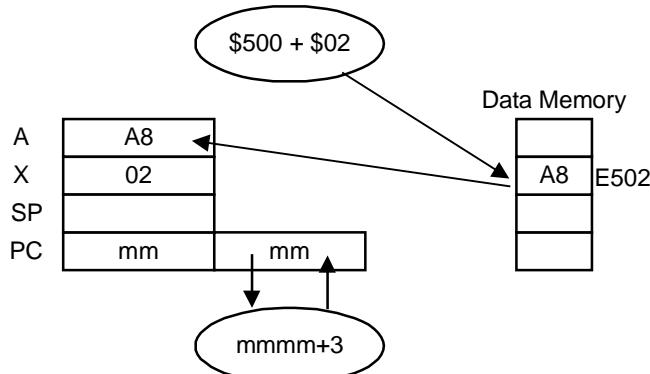
3.5.3 Absolute Indexed Addressing Mode

The absolute indexed addressing mode uses two-part (adr 16 and X) to specify a memory address.

Operation: **OP-code** **Adr 16, X**

[Example 3-4]: Absolute indexed addressing mode

LDA	\$E502,X
-----	----------



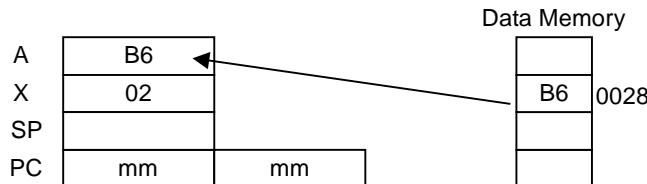
3.5.4 Zero Page Addressing Mode

The zero page addressing mode uses the low-order byte of the address in page zero (adr 08) to specify a memory address.

Operation: **OP-Code** **Adr 08**

[Example 3-5]: Zero page addressing mode

LDA	\$28
-----	------



3.5.5 Zero Page Indexed Addressing Mode

The zero page indexed addressing mode uses two-part (adr 08 and X) to specify a memory address.

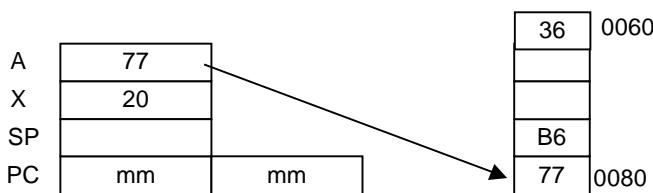
Operation: **OP-Code** **Adr 08, X**

[Example 3-6]: Zero page indexed addressing mode

LDX	#\$20
LDA	#\$77
STA	\$60,X

$\$20 + \$60 = \$80$

Data Memory



3.5.6 Implied Addressing Mode

The implied addressing mode does not have any address.

Operation: **OP-Code**

[Example 3-7]: Implied addressing mode

TAX	; To transfer data from accumulator to register X.
CLC	; To clear carry

3.5.7 Accumulator Addressing Mode

The accumulator addressing mode does not have any address. The instruction operates on the data in the

accumulator.

Operation: **OP-Code**

[Example 3-8]: Accumulator addressing mode

ROL A ; Rotate Left with Carry
ROR A ; Rotate Right with Carry

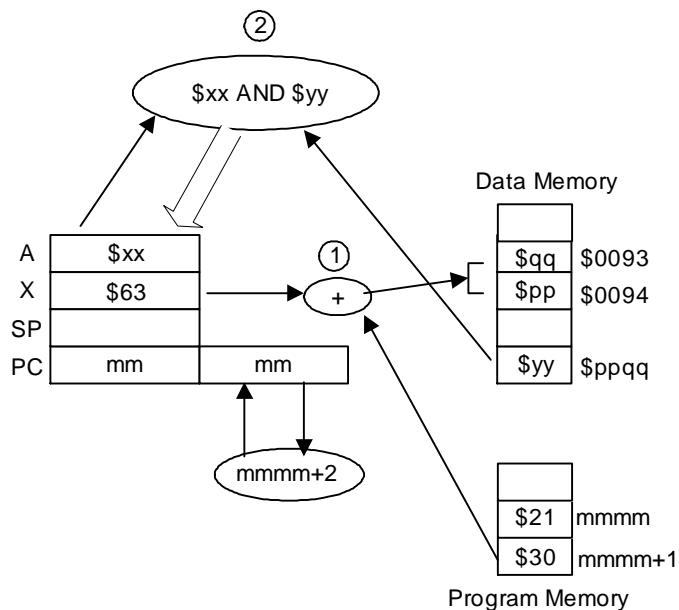
3.5.8 Indexed indirect Addressing Mode

The pre-indexed indirect addressing mode uses “(adr 08 and X)” to specify a memory address. Only register X can be used in this mode. The pre-indexed indirect address is a zero-page indexed direct address. Thus, the valid address must be on page zero.

Operation : **OP-Code** **(Adr 08, X)**

[Example 3-9]: Indexed indirect addressing mode

AND (\$30, X)



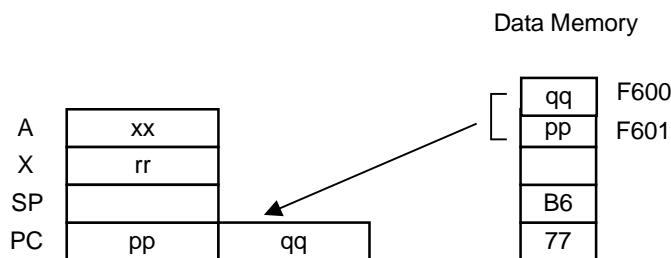
3.5.9 Indirect Addressing Mode

Index addressing mode can only use JMP instruction.

Operation: **OP-Code** **JMP** **(Adr 16)**

[Example 3-10]: Indirect addressing mode

JMP (\$F600)



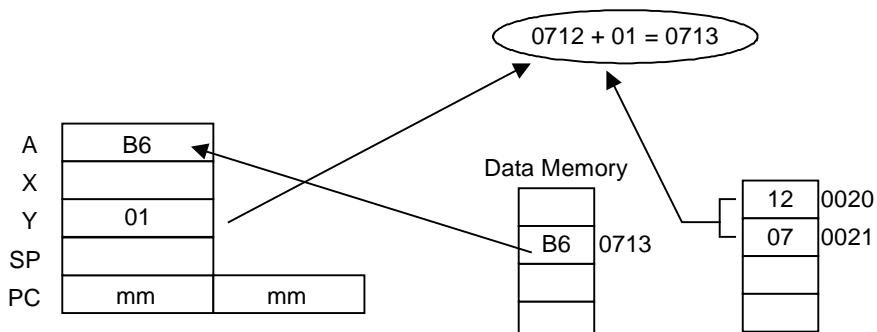
3.5.10 Indirect Indexed Addressing Mode

Indirect Indexed addressing mode can only be applied for Y index register.

Operation: **OP-Code** **(aa), Y**

[Example 3-11]: Indirect indexed addressing mode

LDA	(\$20), Y
-----	-----------



3.5.11 Relative Addressing Mode

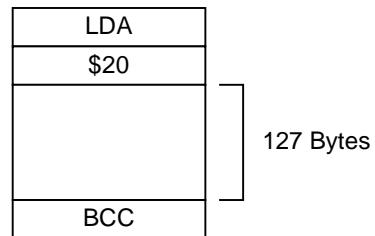
The relative addressing mode uses (adr 08) to specify a memory address. The relative addressing mode only uses with the branch instructions. The maximum branch forward is 127 bytes and the maximum branch backward is 128 bytes.

Operation: **OP-Code** **Adr 08**

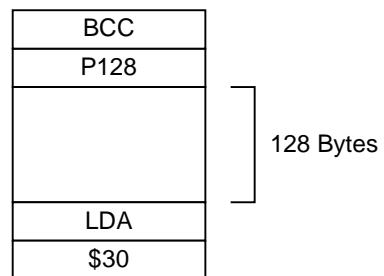
[Example 3-12]: Relative addressing mode

BCC	M127
BCC	P128

M127: LDA #\$20
.
.
.
BCC M127



BCC P128
.
.
.
P128: LDA #\$30



3.6 Design Tips

None

3.7 Related Application Notes and Libraries

This section lists some application notes and libraries about various usage of SPMC65X family. The application notes may help user develop quickly and offer the concepts about the usage of this topic. Sunplus also supports some powerful libraries about general operation, mathematics, communication, etc. The purpose of library is to help user speed up development. The current application notes and libraries related to CPU are listed below.

Application Note:

Title	Application Note Series Number
No associated application notes at this time.	

Library:

Title	Application Note Series Number
SPMC65X series software basics manual	AN_O0100.DOC
SPMC65X series software advances 1 manual (data processing operation)	AN_O0101.DOC
SPMC65X series software advances 2 manual (mathematical operation)	AN_O0102.DOC

3.8 Revision History

Revision	Date	Remark
V0.1	03/31/2005	First edition

4 Memory Organization

4.1 Description

The SPMC65X family have separate address spaces for Program memory and Data memory.

Program memory can only be read, not written to. It can be up to maximum 16K bytes of Program memory for SPMC65X family.

Data memory can be read and written by control registers. User RAM is maximum 928 bytes including stack area. All the on-chip peripherals of the SPMC65X family devices are mapped into data memory space (0~5FH). Access to these registers are made by the CPU instructions addressing their data memory locations. Figure 4-1 shows memory map of SPMC65X. Note that accessing “Reserved” memory locations may cause illegal address reset.

4.2 Memory Map

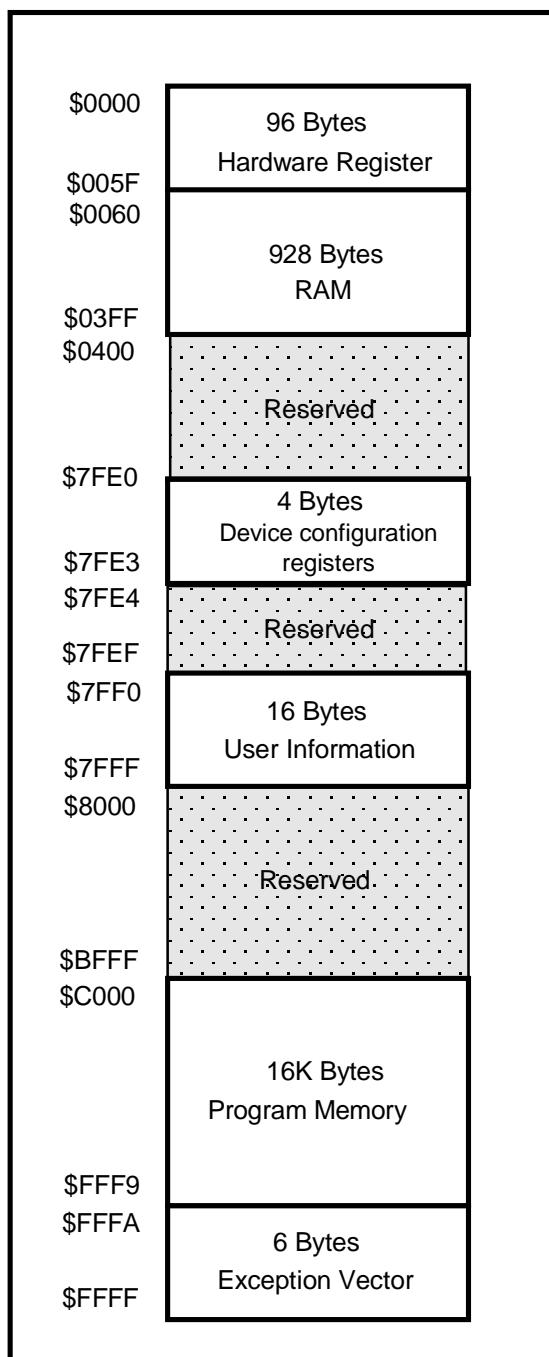


Figure 4-1 SPMC65X Memory Map

4.3 Program Memory

A 16-bit program counter is capable of addressing up to 64K bytes, but the SPMC65X family have maximum 16K bytes of program memory space only physically implemented. Figure 4-2 shows a map of Program Memory. The program memory is constructed from OTP ROM, which can be written by programmer and be protected by security bit setting. When the security bit is set to '0' by a programmer, the 16k of OTP ROM can't be read by user. Contrary, if security bit is set to '1', the 16k of OTP ROM can be readable. In other words, the program is not

secure for protection. device configuration registers and user information can be readable even security. After reset, the CPU begins execution from reset vector which is stored in address \$FFFC and \$FFFD as shown in Figure 4-2. The interrupt causes the CPU to jump to specific location, where it commences the execution of the service routine. The interrupt service location spaces 2-byte interval: \$FFFE and \$FFFF for IRQ vector. The address of NMI, RESET and IRQ are located from \$FFFA to \$FFFF. The exception vectors should be specified in the program to have proper operation. Please see the Figure 4-2.

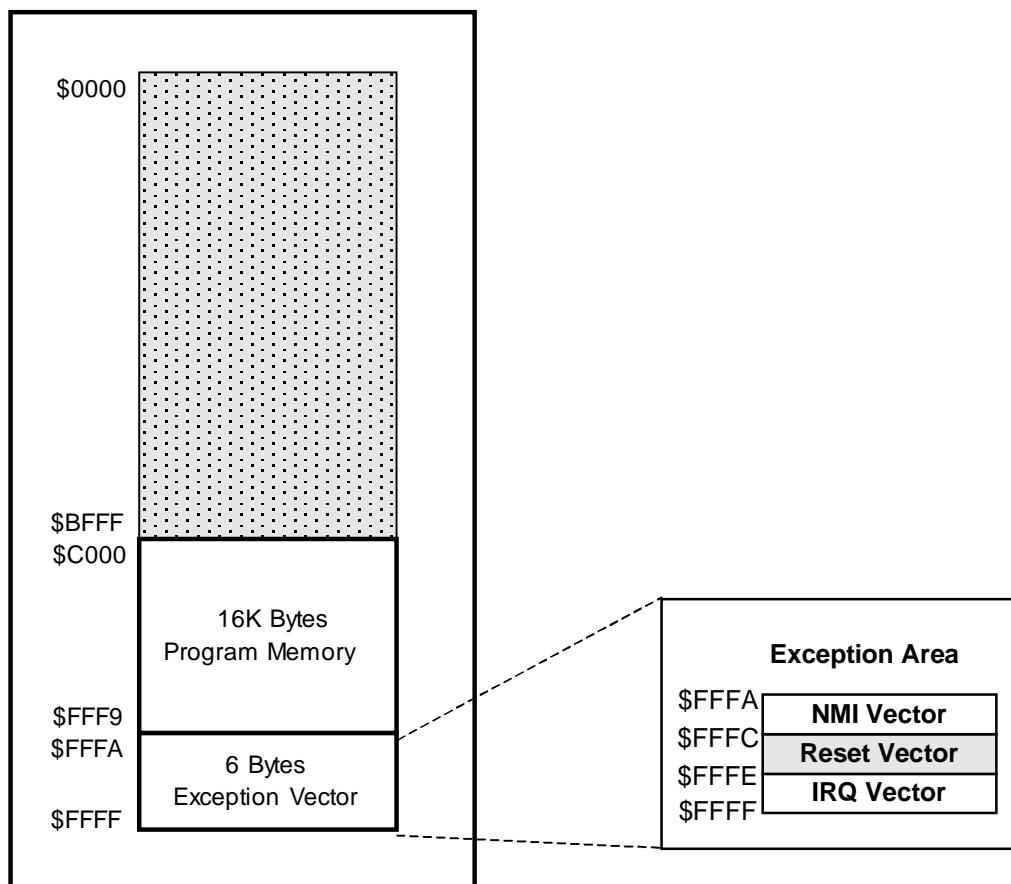


Figure 4-2 SPMC65X Memory Map

[Example 4-1]: Interrupt Vector Table in software

VECTOR:	.SECTION
DW	V_NMI
DW	V_Reset
DW	V_IRQ

4.4 Data Memory

Data Memory is divided into four groups, control registers, user RAM area (including stack area), Device configuration registers and user information. Figure 4-3 shows a map of Data Memory. The first 96 (0-\$5F) data memory locations are used for on-chip peripherals. Addresses \$0060~\$03FF are user RAM area including stack area.

Stack pointer are started from \$01E0~\$01FF with downward direction. Once the Stack is over the 01E0h, CPU reset will be generated. The 4 bytes register located in \$7FE0 ~ \$7FE3 are used to define the device configurations, which help programmer to configure this chip usage condition. SPMC65X family also has the user information registers located in \$7FF0 ~ \$7FFF and can be programmed by user for series number or version control setting..

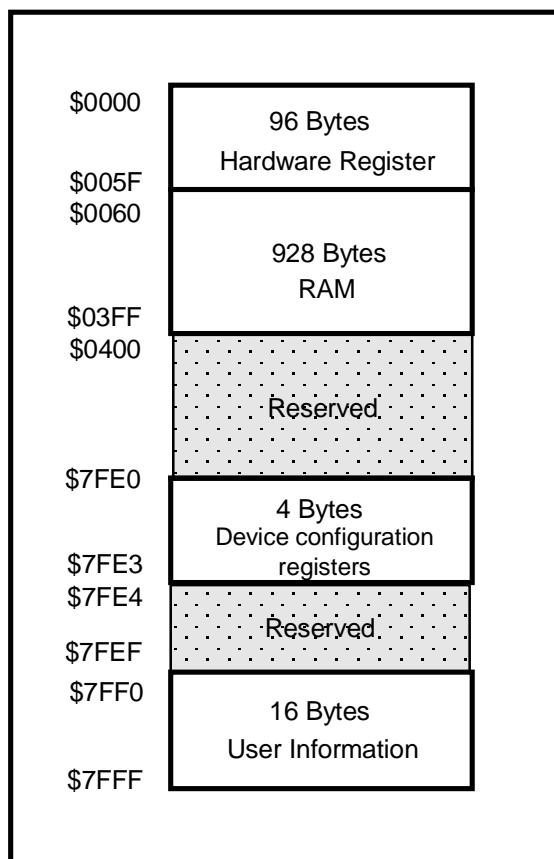


Figure 4-3 SPMC65X Data Memory Map

4.4.1 Hardware Control Register

The hardware control registers are used by CPU and peripheral function blocks for controlling the desired operations of devices. SPMC65X family contain 96 byte registers that are in address range of 0000h to 005fh. The functions of hardware control registers are control and status bits for interrupt system, the timer or counter, analog to digital converters and I/O ports. Section 4.4.1.1 Summary shows the summary of hardware control registers. For detailed descriptions of hardware control registers, refer to them in each peripheral section.

n 4.4.1.1 Summary

Addr.	\$0000	\$0001	\$0002	\$0003	\$0004	\$0005	\$0006	\$0007	\$0008	\$0009	\$000A	\$000B	\$000C	\$000D	\$000E	\$000F
	I/O PortA~D Port Data & Configuration															Interrupt Configuration & Flag
Addr.	\$0010	\$0011	\$0012	\$0013	\$0014	\$0015	\$0016	\$0017	\$0018	\$0019	\$001A	\$001B	\$001C	\$001D	\$001E	\$001F
	WTD Clear	Timer/PWM/Capture 0~1 Configuration & Data														
Addr.	\$0020	\$0021	\$0022	\$0023	\$0024	\$0025	\$0026	\$0027	\$0028	\$0029	\$002A	\$002B	\$002C	\$002D	\$002E	\$002F
	Timer/PWM/Capture 4~5 Configuration & Data								Interrupt Configuration & Flag	ADC Configuration & Data				Buzzer	Comp 0~1 Config	
Addr.	\$0030	\$0031	\$0032	\$0033	\$0034	\$0035	\$0036	\$0037	\$0038	\$0039	\$003A	\$003B	\$003C	\$003D	\$003E	\$003F
	System & Watchdog Configuration									SPI Configuration & Data						
Addr.	\$0040	\$0041	\$0042	\$0043	\$0044	\$0045	\$0046	\$0047	\$0048	\$0049	\$004A	\$004B	\$004C	\$004D	\$004E	\$004F
	I/O PortE~F Port Data & Configuration								UART Configuration & Data	IIC Configuration & Data						
Addr.	\$0050	\$0051	\$0052	\$0053	\$0054	\$0055	\$0056	\$0057	\$0058	\$0059	\$005A	\$005B	\$005C	\$005D	\$005E	\$005F
									DAC Configuration & Data	Cap Ctrl	I/O PortA~F Port Data Latch Buffer				Timer5 Data	

n 4.4.1.2 Control Registers list
\$0000~\$000F:

Address	Function	Power-on Reset value	R/W	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
\$0000	Port A IO (P_IOA_Data)	00h	R	Port A I/O pins							
\$0001	Port B IO (P_IOB_Data)	00h	R	Port B I/O pins							
\$0002	Port C IO (P_IOC_Data)	00h	R	Port C I/O pins							
\$0003	Port D IO (P_IOD_Data)	00h	R	Port D I/O pins							
\$0004	Port A Direction (P_IOA_Dir)	00h	R	Port A Data Direction register. 0=IN, 1=OUT.							
\$0005	Port B Direction (P_IOB_Dir)	00h	R	Port B Data Direction register. 0=IN, 1=OUT.							
\$0006	Port C Direction (P_IOC_Dir)	00h	R	Port C Data Direction register. 0=IN, 1=OUT.							
\$0007	Port D Direction (P_IOD_Dir)	00h	R	Port D Data Direction register. 0=IN, 1=OUT.							
\$0008	Port A Attribute (P_IOA_Attrib)	00h	R	Port A Attribute register							
\$0009	Port B Attribute (P_IOB_Attrib)	00h	R	Port B Attribute register							
\$000A	Port C Attribute (P_IOC_Attrib)	00h	R	Port C Attribute register							
\$000B	Port D Attribute (P_IOD_Attrib)	00h	R	Port D Attribute register							
\$000C	Interrupt Flag 0 (P_INT_Flag0)	00h	R	ADIF	WDIF	IRQ5IF/ CAP5IF	IRQ4IF/ CAP4IF	IRQ3IF	IRQ2IF	IRQ1IF/ CAP3IF	IRQ0IF/ CAP2IF
						Write '1' to clear corresponding interrupt flag(s)					
\$000D	Interrupt Ctrl 0 (P_INT_Ctrl0)	00h	R	ADIE	WDIE	IRQ5IE/ CAP5IE	IRQ4IE/ CAP4IE	IRQ3IE	IRQ2IE	IRQ1IE/ CAP3IE	IRQ0IE/ CAP2IE
						IRQ5IE/ CAP5IE	IRQ4IE/ CAP4IE	IRQ3IE	IRQ2IE	IRQ1IE/ CAP3IE	IRQ0IE/ CAP2IE
\$000E	Interrupt Flag 1 (P_INT_Flag1)	00h	R	CAP1IF	CAP0IF	T5OIF	T4OIF	T3OIF	T2OIF	T1OIF	T0OIF
						Write '1' to clear corresponding interrupt flag(s)					
\$000F	Interrupt Ctrl 1 (P_INT_Ctrl1)	00h	R	CAP1IE	CAP0IE	T5OIE	T4OIE	T3OIE	T2OIE	T1OIE	T0OIE
						CAP1IE	CAP0IE	T5OIE	T4OIE	T3OIE	T2OIE

\$0011~\$001F : Timer/PWM Configuration & Data

Address	Function	Power-on Reset value	Reset value	R/W	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0						
\$0010	Watchdog clear (P_WDT_Clr)		00h	R	0	0	0	0	0	0	0	0						
				W	Write #\$55 to clear Watchdog timer (C_WDT_Clr = \$55)													
\$0011	Timer0-1 Ctrl0 (P_TMR0_1_Ctrl0)		00h	R	0	Timer1 Function Selection			0	Timer0 Function Selection								
				W	-	Timer1 Function Selection			-	Timer0 Function Selection								
\$0012	Timer0-1 Ctrl1 (P_TMR0_1_Ctrl1)		00h	R	0	Timer1 Pre-scale Selection			0	Timer0 Pre-scale Selection								
				W	-	Timer1 Pre-Scale Selection			-	Timer0 Pre-scale Selection								
\$0013	Timer0 Counter Low (P_TMR0_Count)		00h	R	Timer0 8-bit or 16-bit Low Byte Counter Register													
	Timer0 Preload Low (P_TMR0_Preload)			W	Timer0 8-bit or 16-bit Low Byte Preload Register													
	Compare0 Low Byte (P_TMR0_Comp)		00h	R	Timer0 8-bit or 16-bit Low Byte Counter Value													
				W	Timer0 8-bit or 16-bit Low Byte Compare Value													
	Capture0 Low Byte (P_TMR0_Cap)		00h	R	Timer0 8-bit or 16-bit Low Byte Capture Width Value													
				W	Timer0 8-bit or 16-bit Low Byte Preload Value													
	PWM0 Period (P_TMR0_PWMPeriod)		00h	R	Timer0 8-bit PWM Period Value													
				W	Timer0 8-bit PWM Period Value													
\$0014	Timer0 Counter High (P_TMR0_CountHi)		00h	R	Timer0 16-bit High Byte Counter Register													
	Timer0 Preload High (P_TMR0_PreloadHi)			W	Timer0 16-bit High Byte Preload Register													
	Compare0 High Byte (P_TMR0_CompHi)		00h	R	Timer0 16-bit High Byte Counter Value													
				W	Timer0 16-bit High Byte Compare Value													
	Capture0 High Byte (P_TMR0_CapHi)		00h	R	Timer0 16-bit High Byte Capture Width Value													
				W	Timer0 16-bit High Byte Preload Value													
	Capture0 Cycle (P_TMR0_CapCycle8)		00h	R	Timer0 8-bit Capture Cycle Value													
				W	Timer0 8-bit Capture Preload Value													
	PWM0 Duty (P_TMR0_PWMDuty)		00h	R	Timer0 8-bit PWM Duty Value													
				W	Timer0 8-bit PWM Duty Value													
\$0015	Timer1 Counter Low (P_TMR1_Count)		00h	R	Timer1 8-bit or 16-bit Low Byte Counter Register													
	Timer1 Preload Low (P_TMR1_Preload)			W	Timer1 8-bit or 16-bit Low Byte Preload Register													
	Compare1 Low Byte (P_TMR1_Comp)		00h	R	Timer1 8-bit or 16-bit Low Byte Counter Value													
				W	Timer1 8-bit or 16-bit Low Byte Compare Value													
	Capture1 Low Byte (P_TMR1_Cap)		00h	R	Timer1 8-bit or 16-bit Low Byte Capture Width Value													
				W	Timer1 8-bit or 16-bit Low Byte Preload Value													
	PWM1 Low Period		00h	R	Timer1 12-bit PWM Low Bye Period Value													

			W	Timer1 12-bit PWM Low Bye Period Value				
\$0016	Timer1 Counter High (P_TMR1_CountHi)		00h	R	Timer1 16-bit High Byte Counter Register			
	Timer1 Preload High (P_TMR1_PreloadHi)			W	Timer1 16-bit High Byte Preload Register			
	Compare1 High Byte (P_TMR1_CompHi)		00h	R	Timer1 16-bit High Byte Counter Value			
				W	Timer1 16-bit High Byte Compare Value			
	Capture1 High Byte (P_TMR1_CapHi)		00h	R	Timer1 16-bit High Byte Capture Width Value			
				W	Timer1 16-bit High Byte Preload Value			
	Capture1 Cycle (P_TMR1_CapCycle8)		00h	R	Timer1 8-bit Capture Cycle Value			
				W	Timer1 8-bit Capture Preload Value			
\$0017	PWM1 Duty Period (P_TMR1_PWMPeriod)		00h	R	Timer1 12-bit PWM Duty High Byte Value	Timer1 12-bit PWM Period High Byte Value		
				W	Timer1 12-bit PWM Duty High Byte Value	Timer1 12-bit PWM Period High Byte Value		
\$0018	Timer2-3 Ctrl0 (P_TMR2_3_Ctrl0)		00h	R	0	Timer3 Function Selection	0	Timer2 Function Selection
				W	-	Timer3 Function Selection	-	Timer2 Function Selection
\$0019	Timer2-3 Ctrl1 (P_TMR2_3_Ctrl1)		00h	R	0	Timer3 Pre-scale Selection	0	Timer2 Pre-scale Selection
				W	-	Timer3 Pre-scale Selection	-	Timer2 Pre-scale Selection
\$001A	Timer2 Counter Low (P_TMR2_Count)		00h	R	Timer2 8-bit or 16-bit Low Byte Counter Register			
	Timer2 Preload Low (P_TMR2_Preload)			W	Timer2 8-bit or 16-bit Low Byte Preload Register			
	Compare2 Low Byte (P_TMR2_Comp)		00h	R	Timer2 8-bit or 16-bit Low Byte Counter Value			
				W	Timer2 8-bit or 16-bit Low Byte Compare Value			
	Capture2 Low Byte (P_TMR2_Cap)		00h	R	Timer2 8-bit or 16-bit Low Byte Capture Width Value			
				W	Timer2 8-bit or 16-bit Low Byte Preload Value			
\$001B	PWM2 Period (P_TMR2_PWMPeriod)		00h	R	Timer2 8-bit PWM Period Value			
				W	Timer2 8-bit PWM Period Value			
	Timer2 Counter High (P_TMR2_CountHi)		00h	R	Timer2 16-bit High Byte Counter Register			
				W	Timer2 16-bit High Byte Preload Register			
	Compare2 High Byte (P_TMR2_CompHi)		00h	R	Timer2 16-bit High Byte Counter Value			
				W	Timer2 16-bit High Byte Compare Value			
\$001C	Capture2 High Byte (P_TMR2_CapHi)		00h	R	Timer2 16-bit High Byte Capture Width Value			
				W	Timer2 16-bit High Byte Preload Value			
	Capture2 Cycle (P_TMR2_CapCycle8)		00h	R	Timer2 8-bit Capture Cycle Value			
				W	Timer2 8-bit Capture Preload Value			
	PWM2 Duty (P_TMR2_PWMPeriod)		00h	R	Timer2 PWM Duty Value			
				W	Timer2 PWM Duty Value			
\$001C	Timer3 Counter Low (P_TMR3_Count)		00h	R	Timer3 8-bit or 16-bit Low Byte Counter Register			

	Timer3 Preload Low (P_TMR3_Preload)			W	Timer3 8-bit or 16-bit Low Byte Preload Register			
	Compare3 Low Byte (P_TMR3_Comp)		00h	R	Timer3 8-bit or 16-bit Low Byte Counter Value			
				W	Timer3 8-bit or 16-bit Low Byte Compare Value			
\$001D	Capture3 Low Byte (P_TMR3_Cap)		00h	R	Timer3 8-bit or 16-bit Low Byte Capture Width Value			
				W	Timer3 8-bit or 16-bit Low Byte Preload Value			
	PWM3 Low Period (P_TMR3_PWMPeriod)		00h	R	Timer3 12-bit Low Byte PWM Period Value			
				W	Timer3 12-bit Low Byte PWM Period Value			
\$001E	Timer3 Counter High (P_TMR3_CountHi)		00h	R	Timer3 16-bit High Byte Counter Register			
	Timer3 Preload High (P_TMR3_PreloadHi)			W	Timer3 16-bit High Byte Preload Register			
	Compare3 High Byte (P_TMR3_CompHi)		00h	R	Timer3 16-bit High Byte Counter Value			
				W	Timer3 16-bit High Byte Compare Value			
	Capture3 High Byte (P_TMR3_CapHi)		00h	R	Timer3 16-bit High Byte Capture Width Value			
				W	Timer3 16-bit High Byte Preload Value			
	Capture3 Cycle (P_TMR3_CapCycle8)		00h	R	Timer3 8-bit Capture Cycle Value			
				W	Timer3 8-bit Capture Preload Value			
\$001F	PWM3 Duty Period (P_TMR3_DutyPeriod)		00h	R	Timer3 12-bit High Byte PWM Duty Value	Timer3 12-bit High Byte PWM Period Value		
				W	Timer3 12-bit High Byte PWM Duty Value	Timer3 12-bit High Byte PWM Period Value		
	Timer4-5 Ctrl0 (P_TMR4_5_Ctrl0)		00h	R	0	Timer5 Function Selection	0	Timer4 Function Selection
				W	-	Timer5 Function Selection	-	Timer4 Function Selection

\$0020~002F:Timer 4~5 and ADC control

Address	Function	Reset value	Power-on Reset value	R/W	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0							
\$0020	Timer4-5 Ctrl1 (P_TMR4_5_Ctrl1)		00h	R	0	Timer5 Pre-scale Selection			0	Timer4 Pre-scale Selection									
						W	-	Timer5 Pre-scale Selection			-	Timer4 Pre-scale Selection							
\$0021	Timer4 Counter Low (P_TMR4_Count)		00h	R	Timer4 8-bit or 16-bit Low Byte Counter Register														
	Timer4 Preload Low (P_TMR4_Preload)			W	Timer4 8-bit or 16-bit Low Byte Preload Register														
	Compare4 Low Byte (P_TMR4_Comp)		00h	R	Timer4 8-bit or 16-bit Low Byte Counter Value														
					Timer4 8-bit or 16-bit Low Byte Compare Value														
	Capture4 Low Byte (P_TMR4_Cap)		00h	R	Timer4 8-bit or 16-bit Low Byte Capture Width Value														
					Timer4 8-bit or 16-bit Low Byte Preload Value														
\$0022	PWM4 Period (P_TMR4_PWMPeriod)		00h	R	Timer4 8-bit PWM Period Value														
					Timer4 8-bit PWM Period Value														
	Timer4 Counter High (P_TMR4_CountHi)		00h	R	Timer4 16-bit High Byte Counter Register														
					Timer4 16-bit High Byte Preload Register														
	Compare4 High Byte (P_TMR4_CompHi)		00h	R	Timer4 16-bit High Byte Counter Value														
					Timer4 16-bit High Byte Compare Value														
	Capture4 High Byte (P_TMR4_CapHi)		00h	R	Timer4 16-bit High Byte Capture Width Value														
					Timer4 16-bit High Byte Preload Value														
\$0023	Capture4 Cycle (P_TMR4_CapCycle8)		00h	R	Timer4 8-bit Capture Cycle Value														
					Timer4 8-bit Capture Preload Value														
	PWM4 Duty (P_TMR4_PWMDuty)		00h	R	Timer4 8-bit PWM Duty Value														
					Timer4 8-bit PWM Duty Value														
	Timer5 Counter Low (P_TMR5_Count)		00h	R	Timer5 8-bit or 16-bit Low Byte Counter Register														
					Timer5 8-bit or 16-bit Low Byte Preload Register														
\$0024	Compare5 Low Byte (P_TMR5_Comp)		00h	R	Timer5 8-bit or 16-bit Low Byte Counter Value														
					Timer5 8-bit or 16-bit Low Byte Compare Value														
	Capture5 Low Byte (P_TMR5_Cap)		00h	R	Timer5 8-bit or 16-bit Low Byte Capture Width Value														
					Timer5 8-bit or 16-bit Low Byte Preload Value														
	PWM5 Low Period (P_TMR5_PWMPeriodLo)		00h	R	Timer5 16-bit Low Bye PWM Period Value														
					Timer5 16-bit Low Bye PWM Period Value														
	Timer5 Counter High (P_TMR5_CountHi)		00h	R	Timer5 16-bit High Byte Counter Register														
	Timer5 Preload High (P_TMR5_PreloadHi)			W	Timer5 16-bit High Byte Preload Register														
	Compare5 High Byte		00h	R	Timer5 16-bit High Byte Counter Value														

	(P_TMR5_CompHi)			W	Timer5 16-bit High Byte Compare Value											
\$0025	Capture5 High Byte (P_TMR5_CapHi)		00h	R	Timer5 16-bit High Byte Capture Width Value											
					Timer5 16-bit High Byte Preload Value											
\$0026	Capture5 Cycle (P_TMR5_CapCycle8)		00h	R	Timer5 8-bit Capture Cycle Value											
					Timer5 8-bit Capture Preload Value											
\$0027	PWM5 High Period (P_TMR5_PWMPeriodHi)		00h	R	Timer5 16-bit High Bye PWM Period Value											
					Timer5 16-bit High Bye PWM Period Value											
\$0028	Interrupt Flag 2 (P_INT_Flag2)		00h	R	-	-	ITVALIF	IICIF	UARTIF	SPIIIF	CMP1IF	CMP0IF				
					W	-	'1': clear	Rd only	Rd only	Rd only	'1': clear	'1': clear				
\$0029	A/D Input Channel Ctrl 1 (P_AD_Ctrl1)		00h	R	ADEN	ADVRT	0	0	AD Clock Selection							
					W	ADEN	ADVRT	-	-	AD Clock Selection						
\$002A	A/D Control 2 (P_AD_Ctrl2)		00h	R	ADCE	ADC Channel Selection				0	0	Pin8				
					W	ADCE	ADC Channel Selection				-	-	Pin8			
\$002B	A/D Data Hi (P_AD_DataHi)		00h	R	10-bit A/D High Byte Value											
					W	-										
\$002C	A/D Data Lo (P_AD_DataLo)		00h	R	10-bit A/D Low Byte Value	0	0	0	0	0	0	0				
					W	-										
\$002D	Buzzer Control (P_BUZ_Ctrl)		00h	R	Time Base Clock Selection				Buzzer Clock Selection							
					W	Time Base Clock Selection				Buzzer Clock Selection						
\$002E	Comparator Control (P_CMP_Ctrl)		00h	R	CMP1EN	CMP1RS	CMP1LOG	CMP1OUT	CMP0EN	CMP0RS	CMP0LOG	CMP0OUT				
					W	CMP1EN	CMP1RS	CMP1LOG	CMP1OUT	CMP0EN	CMP0RS	CMP0LOG	CMP0OUT			
\$002F					Reserved											

\$0030~\$003F : System control and SPI

Address	Function	Power-on reset value	Reset value	R/W	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0		
\$0030	System Ctrl (P_SYS_Ctrl)	80h	C0h	R	POR	ERST	LVR	-	WDR	IAR	0	-		
				W	Write '1' to clear corresponding reset flag(s)									
\$0031	Mode Ctrl (P_Mode_Ctrl)		00h	R	0	0	0	0	0	0	0	0		
				W	Write #\$5A to enter STOP mode. (C_MODE_STOP = \$5A) Write #\$A5 to enter HALT mode. (C_MODE_HALT = \$A5) Write #\$66 to reset all internal modules, except CPU. (C_MODE_Reset = \$66)									
\$0032	Watchdog Ctrl (P_WDT_Ctrl)		F2h	R	SCKEN	Watchdog Clock Selection			0	0	0	0		
				W	SCKEN	Watchdog Clock Selection			-	-	-	-		
\$0033	IRQ Option0 (P_IRQ_Opt0)		00h	R					IRQ5ES / CAP5ES	IRQM5	IRQ4ES / CAP4ES	IRQM4		
				W					IRQ5ES / CAP5ES	IRQM5	IRQ4S / CAP4ES	IRQM4		
\$0034	IRQ Option1 (P_IRQ_Opt1)		00h	R	IRQ3ES	IRQM3	IRQ2ES	IRQM2	IRQ1ES / CAP3ES	IRQM1	IRQ0ES / CAP2ES	IRQM0		
				W	IRQ3ES	IRQM3	IRQ2ES	IRQM2	IRQ1ES / CAP3ES	IRQM1	IRQ0ES / CAP2ES	IRQM0		
\$0035	Slew Rate Ctrl (P_IO_Opt)		00h	R	0	0	0	0	0	0	0	SLOWE		
				W	-	-	-	-	-	-	-	SLOWE		
\$0036	LVR Option (P_LVR_Opt)		00h	R	0	0	0	0	0	0	0	LVRV40		
				W	-	-	-	-	-	-	-	LVRV40		
\$0037					Reserved									
\$0038	SPI Control 0 (P_SPI_Ctrl0)		00h	R	SPIEN	MOD	SCKPHA	SCKPOL	SMS	SPI CPU Clock Selection				
				W	SPIEN	MOD	SCKPHA	SCKPOL	SMS	SPI CPU Clock Selection				
\$0039	SPI Control 1 (P_SPI_Ctrl1)		02h	R	SMSEN	SWRST	-	-	-	-	SPI Sampling Clock Selection			
				W	SMSEN	SWRST	-	-	-	-	SPI Sampling Clock Selection			
\$003A	SPI TX/RX Status (P_SPI_Status)		00h	R	SPIIF	SPIIEN	TXBF	0	0	-	-	BUFFull		
				W	'1': clear	SPIIEN	-	-	-	-	-	-		
\$003B	SPI TX DATA (P_SPI_TxData)		00h	R	0	0	0	0	0	0	0	0		
				W	SPI Transmit Data out Buffer									
\$003C	SPI RX DATA (P_SPI_RxData)		00h	R		SPI Receive Data in Buffer								
				W	-	-	-	-	-	-	-	-	-	
\$003D					Reserved									
\$003E					Reserved									
\$003F					Reserved									

\$0040~\$004F : Port E/F and UART/IIC

Address	Function	Reset value	R/W	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
\$0040	Port E IO Data (P_IOE_Data)	00h	R	Port E I/O pins							
				Port E Data Latch (\$005D)							
\$0041	Port F IO Data (P_IOF_Data)	00h	R	Port F I/O pins							
				Port F Data Latch (\$005E)							
\$0042	Port E Direction (P_IOE_Dir)	00h	R	Port E Data Direction register. 0=IN, 1=OUT.							
				Port E Data Direction register. 0=IN, 1=OUT.							
\$0043	Port F Direction (P_IOF_Dir)	00h	R	Port F Data Direction register. 0=IN, 1=OUT.							
				Port F Data Direction register. 0=IN, 1=OUT.							
\$0044	Port E Attribute (P_IOE_Attrib)	00h	R	Port E Attribute register							
				Port E Attribute register							
\$0045	Port F Attribute (P_IOF_Attrib)	00h	R	Port F Attribute register							
				Port F Attribute register							
\$0046	UART Control (P_UART_Ctrl)	00h	R	RXIE	TXIE	RXEN	TXEN	SOFTRST	STOPSEL	PSEL	PEN
				W	RXIE	TXIE	RXEN	TXEN	SOFTRST	STOPSEL	PSEL
\$0047	UART Baud Rate Divider (P_UART_Baud)	00h	R	UART Baud Rate Divider							
				UART Baud Rate Divider							
\$0048	UART STATUS (P_UART_Status)	00h	R	RXIF	TXIF	BUSY	0	0	OERR	PERR	FERR
				W	-	-	-	-	OERR	PERR	FERR
\$0049	UART DATA (P_UART_Data)	00h	R	UART Received Data							
				UART Transmitted Data							
\$004A	IIC Bus Control (P_IIC_Ctrl)	00h	R	IICEN		TXRXEN	INTEN	ACKEN	IIC Clock Selection		
				W	IICEN	TXRXEN	INTEN	ACKEN	IIC Clock Selection		
\$004B	IIC Bus Status (P_IIC_Status)	00h	R	MXT	TXRXSEL	BUSY/SIGGEN	IICIF	ARBITRAT	AAS	AZERO	LRB
				W	MXT	TXRXSEL	BUSY/SIGGEN	IICIF	ARBITRAT	AAS	AZERO
\$004C	IIC Bus DATA (P_IIC_Data)	00h	R	IIC Series Data Register							
				IIC Series Data Register							
\$004D	IIC Bus Address (P_IIC_Address)	00h	R	IIC Series Address Register							
				IIC Series Address Register							
\$004E									Reserved		
\$004F									Reserved		

\$0058~\$005F : DAC and Port Data Register

Address	Function	Power-on Reset value	R/W	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0			
\$0050~54				Reserved										
\$0055	DAC Control (P_DA_Ctrl)		00h	R	DAEN	0	0	0	0	0	0			
				W	DAEN	-	-	-	-	-	-			
\$0056	DAC DATA Low (P_DA_DataLo)		00h	R	10-bit D/A Low Byte Value		0	0	0	0	0			
				W	10-bit D/A Low Byte Value		-	-	-	-	-			
\$0057	DAC DATA High (P_DA_DataHi)		00h	R	10-bit D/A High Byte Value									
				W	10-bit D/A High Byte Value									
\$0058	Capture Control (P_CAP_Ctrl)		00h	R	CAPOPT	CAPIP4	CAPIP3	CAPIP2	CAPIP1	CAPIP0	CAP1ES			
				W	CAPOPT	CAPIP4	CAPIP3	CAPIP2	CAPIP1	CAPIP0	CAP1ES			
\$0059	Port A Latch Buffer (P_IOA_Buf)		00h	R	Port A Data Latch									
				W	Port A Data Latch									
\$005A	Port B Latch Buffer (P_IOB_Buf)		00h	R	Port B Data Latch									
				W	Port B Data Latch									
\$005B	Port C Latch Buffer (P_IOC_Buf)		00h	R	Port C Data Latch									
				W	Port C Data Latch									
\$005C	Port D Latch Buffer (P_IOD_Buf)		00h	R	Port D Data Latch									
				W	Port D Data Latch									
\$005D	Port E Latch Buffer (P_IOE_Buf)		00h	R	Port E Data Latch									
				W	Port E Data Latch									
\$005E	Port F Latch Buffer (P_IOF_Buf)		00h	R	Port F Data Latch									
				W	Port F Data Latch									
\$005F	Capture5 High Cycle (P_TMR5_CapCycleHi)		00h	R	Timer5 16-bit High Byte Capture Cycle Value									
				W	-	-	-	-	-	-	-			
\$005F	PWM5 High Duty (P_TMR5_PWM_DutyHi)		00h	R	Timer5 16-bit High Byte PWM Duty Value									
				W	Timer5 16-bit High Byte PWM Duty Value									

4.4.2 User RAM

The data memory areas (RAM) are located in \$0060 ~ \$03FF. It in SPMC65X family can be used for stack, read and written memory.

4.4.3 Stack Area

32 bytes of stack in \$01E0~\$01FF and stack pointer are started from \$01FF to \$01E0 with downward direction. Once the Stack is over the limiter, CPU reset will be generated.

The stack provides the area where the return address is saved before a jump is performed during the processing routine at the execution of a subroutine call instruction or acceptance of an interrupt. When returning from processing routine, executing the subroutine return instruction [RTS] restores the contents of the program counter from the stack; executing the interrupt return instruction [RTI] restores the contents of the program counter and flags.

The save / restore locations in the stack are determined by the stack pointed (SP). The SP is automatically decreased after the saving, and increased before the restoring. It means the value of the SP indicates the stack location number for the next save. Figure 4-4 shows detailed stack operations.

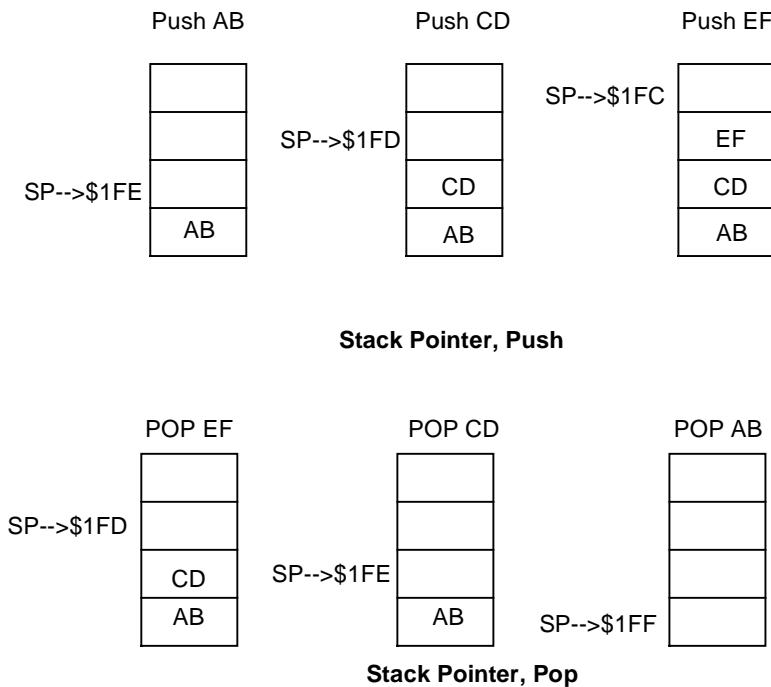


Figure 4-4 SPMC65X Stack Operations

4.4.4 Device Configuration Registers

The device configuration registers are used to set the operation conditions. The OTP Writer configures them during the programming cycle. These device configuration Registers are mapped in program memory locations starting at \$7FE0.

The 4 byte registers located from \$7FE0 to \$7FE3 are used to define operation conditions, such as low voltage reset option, watchdog reset option, source of NMI and the type of clock source, which help user customize certain aspects of the device to the needs of the application.

When using Sunplus Fortis IDE, device configuration registers can be set in [ProjectàSettingàMask Option],Figure 4-5.

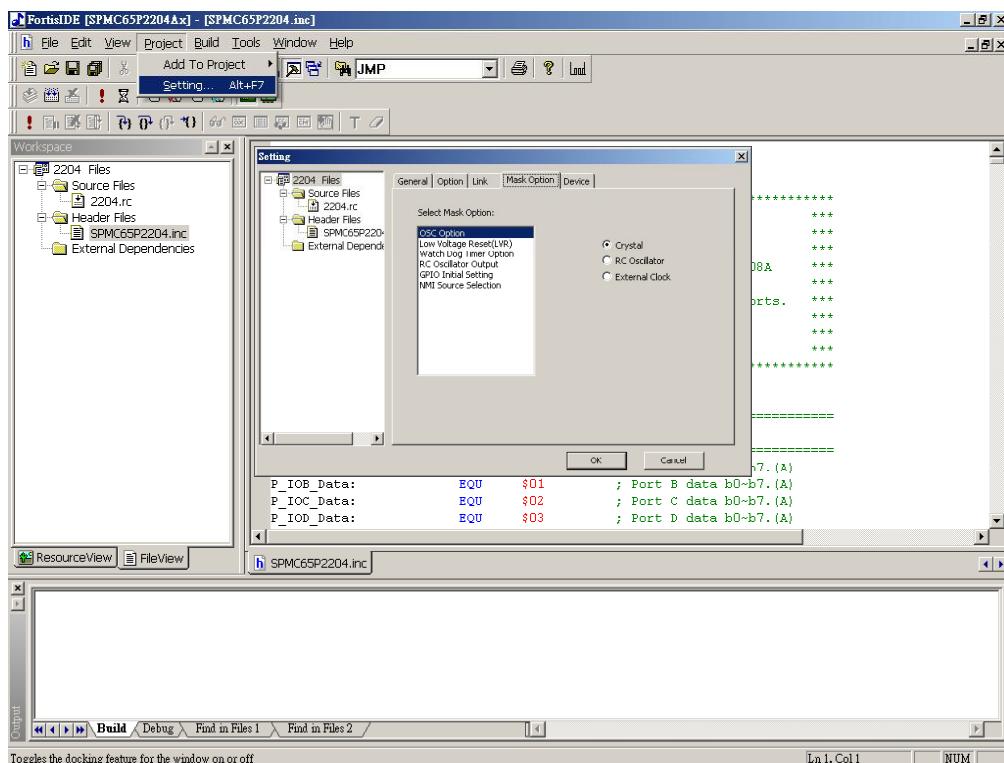


Figure 4-5 Device Configuration Registers

- In [OSC Option] item, there are three types of system clock resources including [Crystal], [RC Oscillator] and [External Clock]. User can select one of them from Fortis IDE. Please refer to Chapter 7
- In [Low Voltage Reset (LVR)] item, user can choose [Enabled] or [Disabled]. If [Enabled], the system will start the [LVR] function. Please refer to Chapter 5.
- In [Watch Dog Timer Option] item, user can choose [Enabled] or [Disabled]. Please refer to Chapter 5.
- In [RC Oscillator Output] item, user can choose [Clock Output] or [No Output]. Please refer to Chapter 7
- In [GPIO Initial Setting] item, user can choose [All float] or [All pull low]. Please refer to Chapter 8
- In [NMI Source Selection] item, user can choose [Disabled], [PB4(INT0)], [PB5(INT1)], [PD0(INT2)], [PD1(INT3)], [PD4(INT4)] or [PD5(INT5)]. Please refer to Chapter 6.

4.4.5 User Information

SPMC65X family has the user information registers located in \$7FF0 ~ \$7FFF and can be programmed by user for series number or version control setting. Sunplus defines \$7FE0~\$7FE3 as the serial number and the remaining address as the free definition location which can record product information. The detailed user information register setting are discussed in chapter 23.

4.5 Special Control Register

The registers located between \$30 and \$36 are related with system operation, so the SPMC65X family provides significant registers with powerful writing method. The processes of double writing are formed with two consecutive writing cycles to the fixed address, which are between \$30 and \$36. The only way to modify the content of control registers located between \$30 and \$36 is by double writing access with certain address. The purposes of double writing are to keep the content of the modification in the control registers right and to protect the data bus or address bus from noise interference.

[Example 4-2]: Clear Reset flag

```
lde      #$FF          ; Clear Reset flag
sta      P_SYS_Ctrl
sta      P_SYS_Ctrl
```

[Example 4-3]: Reset all I/O

```
lde      #C_MODE_Reset    ; reset all I/O
sta      P_MODE_Ctrl
sta      P_MODE_Ctrl
```

[Example 4-4]: WatchDog INT is 1.5Hz

```
lde      #C_WDT_Div_16384   ; WDI= Fslow(25KHz)/16384= 1.5Hz
sta      P_WDT_Ctrl
sta      P_WDT_Ctrl
```

[Example 4-5]: INT0 is rising edge trigger, other are falling

```
lde      #C IRQOpt1_IRQ0ES   ; INT0 is rising edge trigger, other are falling
sta      P_IRQ_Opt1
sta      P_IRQ_Opt1
```

[Example 4-6]: Set LVR as 4.0V

```
lde      # C_LVR_V40        ; set LVR= 4.0V
sta      P_LVR_Opt
sta      P_LVR_Opt
```

4.6 Design Tips

None

4.7 Related Application Notes and Libraries

This section lists some application notes and libraries about various usage of SPMC65X family. The application notes may help user develop quickly and offer the concepts about the usage of this topic. Sunplus also supports some powerful libraries about general operation, mathematics, communication, etc. The purpose of library is to help user speed up development. The current application notes and libraries related to Memory Organization are listed below.

Application Note:

Title	Application Note Series Number
No associated application notes at this time.	

Library:

Title	Application Note Series Number
SPMC65X series software basics manual	AN_O0100.DOC
SPMC65X series software advances 1 manual (data processing operation)	AN_O0101.DOC
SPMC65X series software advances 2 manual (mathematical operation)	AN_O0102.DOC

4.8 Revision History

Revision	Date	Remark
V0.1	03/31/2005	First edition



5 Reset

5.1 Description

There are five types of reset resources for the system, Power On Reset (POR), External Reset (RESET), Low Voltage Reset (LVR), Watchdog Timer Reset (WDTR), and Illegal Address Reset (IAR). These reset sources can be concluded as external events and internal events. The external event comes from the power line, or external trigger event. The internal events are come from the program exceptions or internal software reset event.

There are five reset resources for the system:

1. Power On Reset (POR)
2. External Reset (RESET)
3. Low Voltage Reset (LVR)
4. Watchdog Timer Reset (WDR)
5. Illegal Address Reset (IAR)

These reset sources can be concluded as external events and internal events. The external events come from the power line, or external trigger event. The internal events come from the program exceptions or internal software reset event.

The reset sequence is shown in Figure 5-1 :

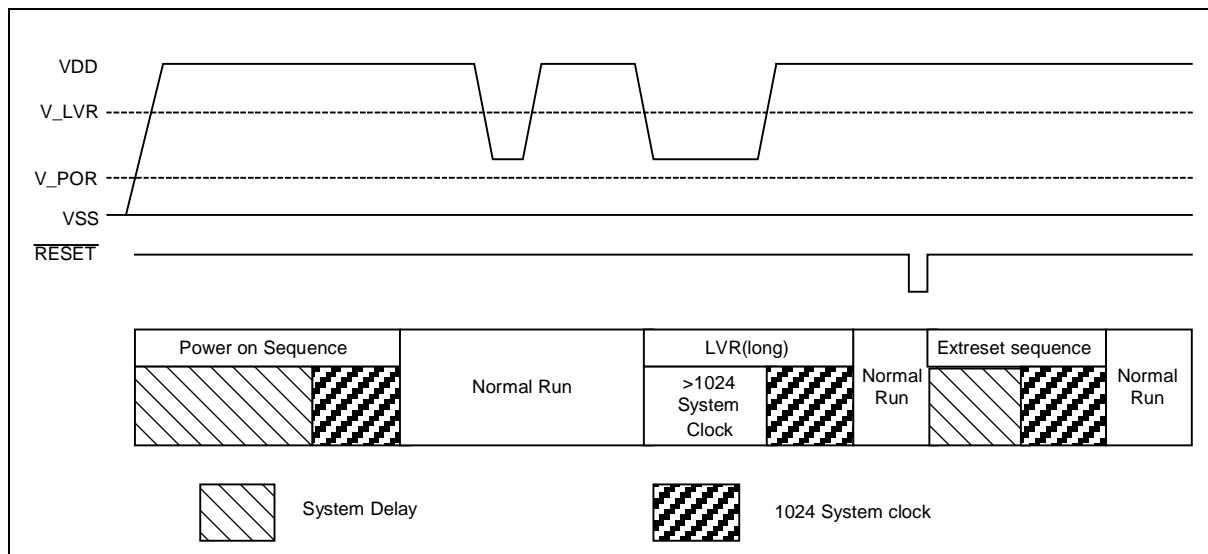


Figure 5-1 Reset Sequence

The responses for these reset events are defined as system reset and CPU reset. System reset will reset all of the system including CPU. CPU reset will reset CPU to restart the program only. Some of reset events will generate system, and rest of them will generate CPU reset. In order to get more robust system design, we propose the responses for different reset sources are listed as Table 5-1.

Table 5-1 Reset Sources List

Resources	Int./Ext.	Reset Functions	Note
Power on Reset (POR)	External	Reset All including CPU	
External Reset	External	Reset All including CPU	
Low Voltage Reset	External	Reset All including CPU	Option Control. T is expected as 1024 system clock. (Case of >T)
WDT Reset (WDR)	Internal	Reset CPU and WDT counting stage only	Option Control.
Illegal Address Reset (IAR)	Internal	Reset CPU and WDT counting stage only	

5.2 Control Register

5.2.1 System Control Register (P_SYS_Ctrl, \$30)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
POR	ERST	LVR	-	WDR	IAR	-	-
R/W	R/W	R/W	-	R/W	R/W	-	-
1	0	0	0	0	0	0	0

Bit 7 **POR:** Power ON Reset flag

0 = No POR

1 = POR is occurred

It is used to indicate the reset cycle is generated by power on reset. The POR will reset the whole internal blocks.

Bit 6 **ERST:** External Reset flag

0 = No ERST

1 = ERST is occurred

It is used to indicate the reset cycle is generated by external reset input,RESETB. The external reset will reset the whole internal blocks, except this register.

Bit 5 **LVR:** Low Voltage Reset flag

0 = No LVR

1 = LVR is occurred

It is used to indicate the reset cycle is generated by low voltage reset with period in case of the LVR being enabled by option. The LVR will reset the whole internal blocks, except this register.

Bit 4 Reserved

Bit 3 **WDR:** Watchdog Timer Reset flag

0 = No WDR

1 = WDR is occurred

It is used to indicate the reset cycle is generated by WDT overflow reset in case of the WDT being enabled by option. The WDT reset will reset CPU, but not whole internal blocks.

Bit 2 **IAR:** Illegal Address Reset flag

0 = No IAR

1 = IAR is occurred

It is used to indicate the reset cycle is generated by IAR in case of the exception being detected. The IAR will reset CPU, not whole internal blocks.

Bit [1:0] Reserved

Note: 1.This flag is cleared by writing the corresponding bit by “1”.

2.This byte must be double-write.

5.2.2 Watchdog Timer Control Register (P_WDT_Ctrl, \$32)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
WDTES	WDS2	WDS1	WDS0	-	-	-	-
R/W	R/W	R/W	R/W				
1	1	1	1	0	0	0	0

Bit 7 **WDTES**: Watchdog enable bit in stop mode

0 = Disable in stop mode

1 = Enable in stop mode

Bit [6:4] **WDS [2:0]**: Select bits of Watchdog interrupt rate

000 = $f_{slow}/128$

001 = $f_{slow}/256$

010 = $f_{slow}/512$

011 = $f_{slow}/1024$

100 = $f_{slow}/2048$

101 = $f_{slow}/4096$

110 = $f_{slow}/8192$

111 = $f_{slow}/16384$

f_{slow} : internal build-in RC oscillator, typical frequency is 25kHz.

WDS[2:0]	WDR Clock (Hz)	WDI Clock (Hz)
000	195/8	195
001	97/8	97
010	48/8	48
011	24/8	24
100	12/8	12
101	6/8	6
110	3/8	3
111	1.5/8	1.5

Note: All of above bits need write twice to set corresponding ones.

5.2.3 LVR Option Register (P_LVR_Opt, \$36)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
-	-	-	-	-	-	-	LVRV40
-	-	-	-	-	-	-	R/W
0	0	0	0	0	0	0	0

Bit [7:1] Reserved

Bit 0 **LVRV40**: LVR level select

0 = The LVR level is 2.5V

1 = The LVR level is 4.0V

Note: 1. This select bit can only be set once and will be cleared only when Power-on reset occurred.

2. All of above bits need write twice to set corresponding ones.

5.3 Operation

There are five types of reset resources for the system. They are Power On Reset (POR), External Reset (RESET), Low Voltage Reset (LVR), Watchdog Timer Reset (WDR), and Illegal Address Reset (IAR). When a reset occurs, user could judge from the system control register (P_SYS_Ctrl). Please refer to [Example 5-1]. The detailed operations of reset sources are introduced as following sections.

[Example 5-1]:Save the value of P_SYS_Ctrl register and then clear it

```
G_MWorkReg1 DS      1
    lda      P_SYS_Ctrl           ; Read out reset flag
    sta      G_MWorkReg1
    lda      #$FF                 ; Clear Reset flag
    sta      P_SYS_Ctrl
    sta      P_SYS_Ctrl
```

5.3.1 Power On Reset

A power-on-reset (POR) is generated when VDD rising is detected. When VDD raised to acceptable level (1.45V), the power on reset circuit starts working. In general, the POR will reset whole chip and the registers. To take advantage of the POR, just tie the RESETB pin directly (or through the resistor) to VDD. This will eliminate external RC components usually needed to create a power-on reset delay. After a reset time, all registers will be initialized at designed value. Figure 5-2 shows the timing of power on reset sequence.

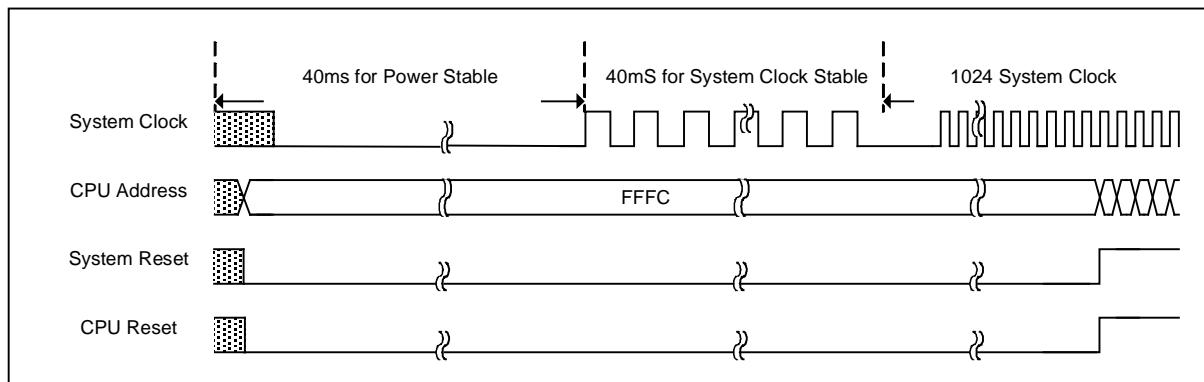


Figure 5-2 Power-On Reset Sequence

5.3.2 External Reset

The SPMC65X family provide an external pin to force the system returning to the initial status. The RESETB pin is used to connect an RC circuit.

In Figure 5-3, The diode D helps discharge the capacitor quickly when VDD powers down. R1 < 40KΩ is recommended to make sure that voltage drop across R does not violate the device's electrical specification. R4=

33Ω to 1KΩ will limit any current flowing into RESETB to prevent ESD.

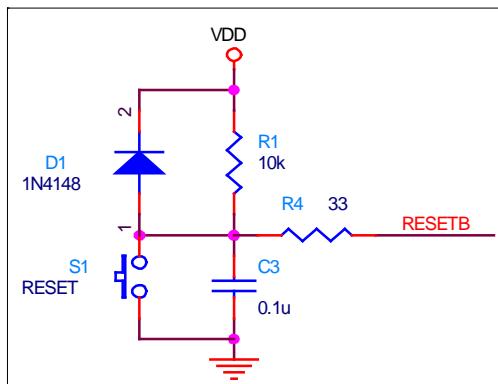


Figure 5-3 Reset Circuit

This pin is a low active signal. When the RESETB 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 PWRT delay when power on reset and then complete the whole reset function. See Figure 5-1 for timing information. Figure 5-4 shows the timing of external reset sequence.

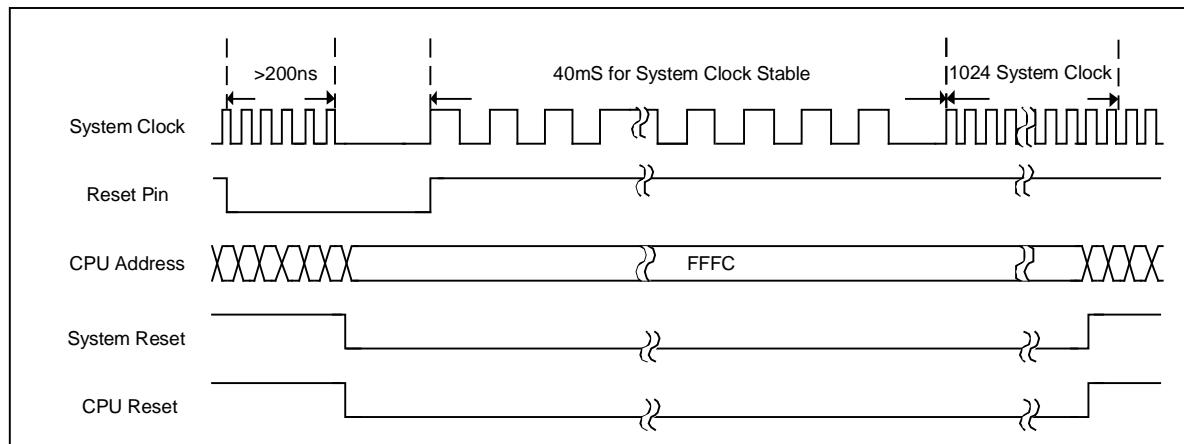


Figure 5-4 External Reset Sequence

5.3.3 Low Voltage Reset

The on-chip Low Voltage Reset (LVR) circuitry makes the device entering reset state when the MCU voltage falls below the specific LVR trigger voltage. This function ensures that the MCU does not continue to work at the invalid operating voltage range.

The Low Voltage Reset bit can be disabled or enabled by setting in device configuration registers shown in Figure 5-5. During power transition cycle, the LVR will monitor power level. Once the power being lower than the specific level and if the power remained low with a specific period, 1024 cycles of system clock, the system reset will be

reset. These reset cycles of LVR will be extended with additional 1024 cycles of system clock.

In general, the P_SYS_Ctrl will not be cleared by LVR. In addition, the LVR event will be ignored while in POR reset cycle or in RESETB reset sequence. Figure 5-6 shows the timing of LVR reset sequence.

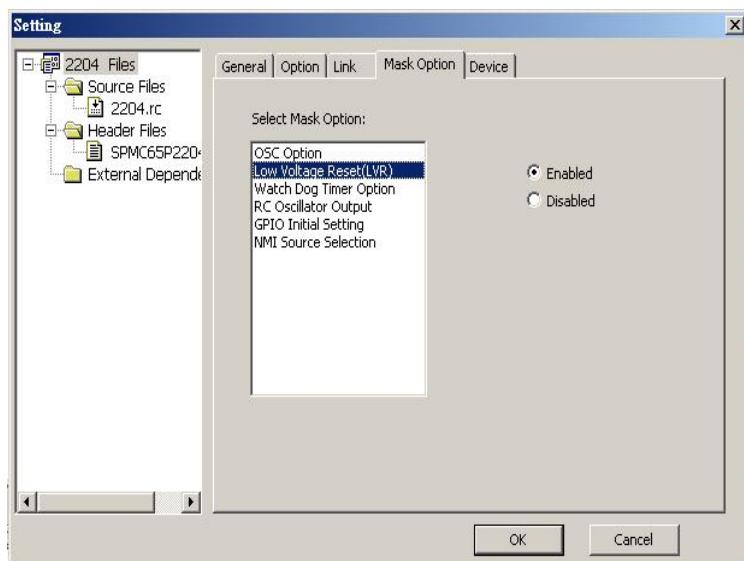


Figure 5-5 LVR setting window

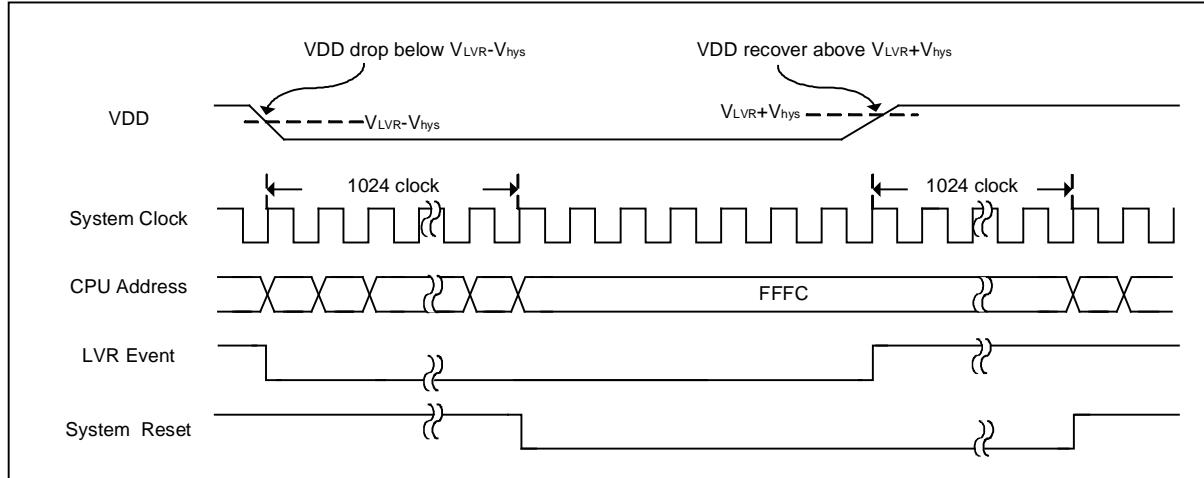


Figure 5-6 LVR Reset Sequence

【Example 5-2】: Enable low voltage reset on condition that operation voltage is bellow 4V

lda	#C_LVR_V40	; set LVR= 4.0V
sta	P_LVR_Opt	
sta	P_LVR_Opt	

5.3.4 Watchdog Timer Reset

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 Watchdog Timer bit can be disabled or enabled by setting in device configuration registers shown in Figure 5-7. The internal reset of WDT will be generated by a time-out of the WDT automatically when watchdog is enabled. This reset signal will be the time base of WDT uses the internal watchdog RC oscillator circuit and typical frequency is 25kHz. It can be operated by eight kinds of clock rate --- P_WDT_Ctrl[6:4]. This time out generates reset if the WDT register is not cleared. An internal reset is generated to restart the CPU. Writing '55h' to P_WDT_Clr does preventing a WDT time-out reset within a specific time. The CPU reset will clear the WDT counting stage to restart the WDT, but the WDT reset flag will not be cleared. Figure 5-8 shows the timing of watch dog reset sequence. The detailed operation is described in Chapter 17.

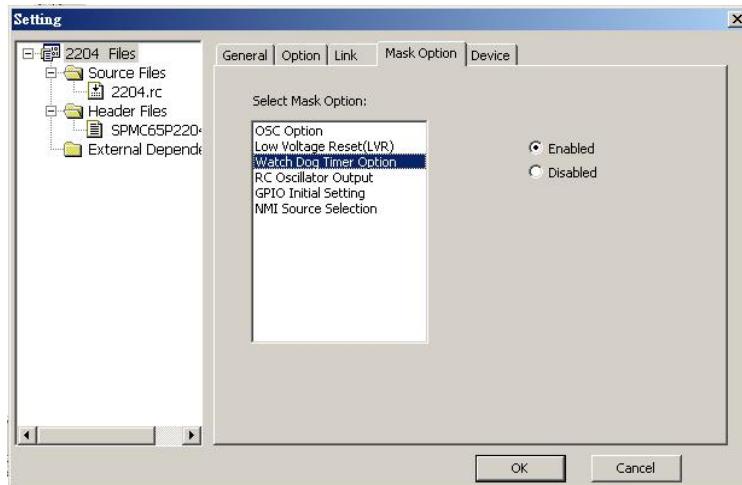


Figure 5-7 WDT setting window

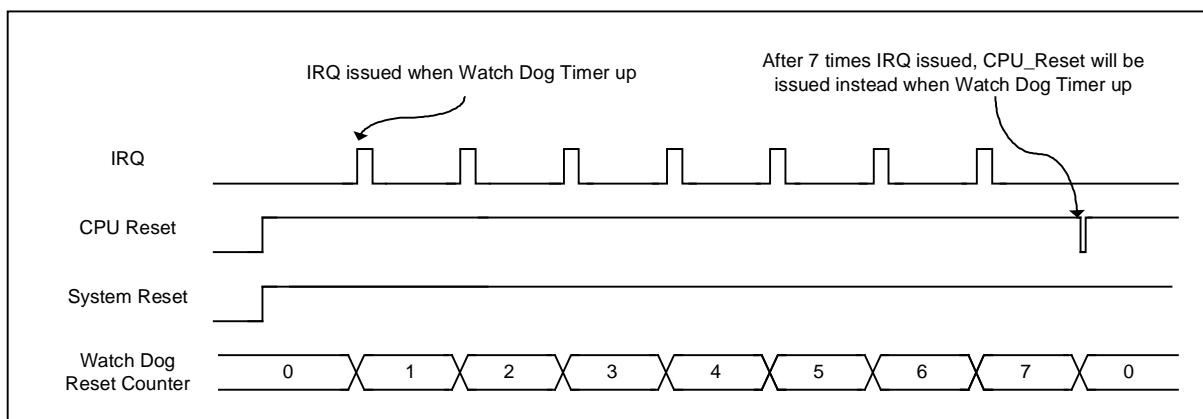


Figure 5-8 Watch-Dog Reset Sequence

5.3.5 Illegal Address Reset

The SPMC65X family provide an illegal address reset for preventing system entering into illegal address. When system goes into illegal address, CPU will be reset. The internal reset of IAR is generated when an instruction

op-code fetch occurs from an address neither in the working area nor in the stack area. The IAR will generate the reset signal that will reset the CPU. Figure 5-9 shows the timing of IAR reset sequence.

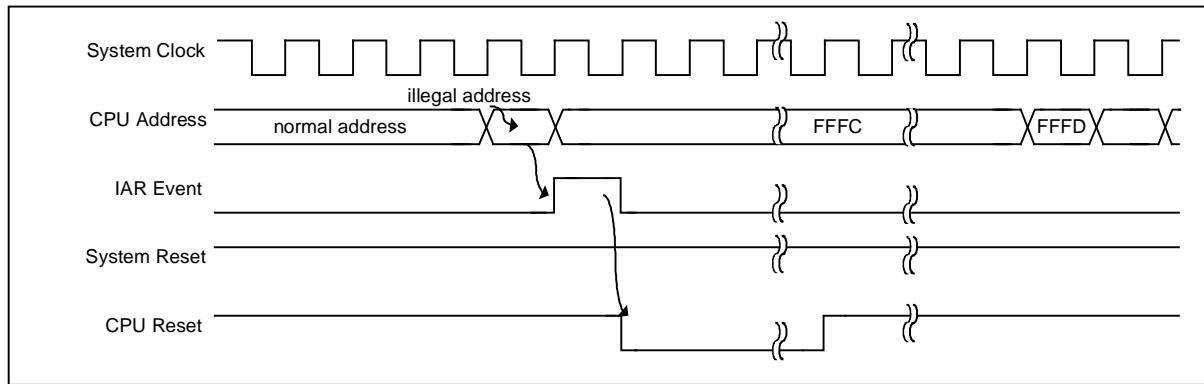


Figure 5-9 IAR Reset Sequence

5.4 Design Tips

[Example 5-3] : Shows how to initialize LVR function based on SPMC65P2404A. Please refer to an example of Fortis IDE.

- Set LVR as 4.0V.

```
.SYNTAX 6502 ; process standard 6502 addressing syntax
.LINKLIST ; generate linklist information
.SYMBOLS ; generate symbolic debug information

.INCLUDE      SPMC65P2404A.inc

.PAGE0          ; define values in the range from $00 to $FF
;
.DATA           ; define data storage section
;
.CODE
; ****
;*
;* Power on Reset Process - Main Program *
;*
; ****

V_Reset:
    sei          ; Disable interrupt
    ldx      #C_STACK_BOTTOM ; Initial stack pointer at $00FF
    txs          ; Transfer to stack point
;
```

```
;  
    jsr      F_InitIOPort           ; initial GPIO port  
;  
    lda      P_SYS_Ctrl  
    sta      P_IOA_Data           ; IOA = reset flag  
;  
    set      P_SYS_Ctrl, CB_SCR_LVR1  
    set      P_SYS_Ctrl, CB_SCR_LVR1 ; clear LVR reset flag  
;  
    set      P_LVR_Opt,CB_LVR_V40  
    set      P_LVR_Opt,CB_LVR_V40 ; LVR set to 4.0V  
L_Main:  
    nop  
    jmp      L_Main  
;  
; ****  
; *  
; *      Interrupt Vector Table  
; *  
; *  
; ****  
  
VECTOR:     .SECTION  
    DW      V_NMI                 ; may download program emulated either  
    DW      V_Reset               ; in internal memory or external memory  
    DW      V_IRQ                 ; dw define two bytes interrupt vector  
;  
; ****  
; *  
; *      End Of Interrupt Vector Table  
; *  
; *  
; ****  
  
.END          ; end of program
```

5.5 Related Application Notes and Libraries

This section lists some application notes and libraries about various usage of SPMC65X family. The application notes may help user develop quickly and offer the concepts about the usage of this topic. Sunplus also supports some powerful libraries about general operation, mathematics, communication, etc. The purpose of library is to help user speed up development. The current application notes and libraries related to Reset are listed below.

Application Note:

Title	Application Note Series Number
No associated application notes at this time.	

Library:

Title	Application Note Series Number
SPMC65X series software basics manual	AN_O0100.DOC
SPMC65X series software advances 1 manual (data processing operation)	AN_O0101.DOC
SPMC65X series software advances 2 manual (mathematical operation)	AN_O0102.DOC

5.6 Revision History

Revision	Date	Remark
V0.1	03/31/2005	First edition

6 Interrupts

6.1 Description

There are seven kinds of interrupts: timer overflow interrupt, capture interrupt, external input interrupt, analog interrupt, communication interrupt, watchdog interrupt, and interval timer interrupt, which are summarized in below table. Each interrupt has individual status (occurred or not) and control (enable or not) registers. Besides setting corresponding interrupt control registers, user must still use 'CLI' instruction to enable interrupt. In general, once an interrupt event occurs, the corresponding flag bit will be set. If the related interrupt control bit is set to enable interrupt, an interrupt request signal will be generated and will be dealt with by CPU for service. The interrupt flag bits must be cleared in the interrupt service routine to prevent program from deadlock in interrupt service routine.

The interrupt sources are listed in Table 6-1. Notice * signs, IRQ0 and Capture2, IRQ1 and Capture3, IRQ4 and Capture4, IRQ5 and Capture5 interrupts use the same interrupt status and control register.

Source		Interrupt status register	Interrupt control register	Source		Interrupt status register	Interrupt control register
Timer Overflow	Timer0	T0OIF (\$0E.0)	T0OIE (\$0F.0)	External Input	IRQ0*	IRQ0IF (\$0C.0)	IRQ0IE (\$0D.0)
	Timer1	T1OIF (\$0E.1)	T1OIE (\$0F.1)		IRQ1*	IRQ1IF (\$0C.1)	IRQ1IE (\$0D.1)
	Timer2	T2OIF (\$0E.2)	T2OIE (\$0F.2)		IRQ2	IRQ2IF (\$0C.2)	IRQ2IE (\$0D.2)
	Timer3	T3OIF (\$0E.3)	T3OIE (\$0F.3)		IRQ3	IRQ3IF (\$0C.3)	IRQ3IE (\$0D.3)
	Timer4	T4OIF (\$0E.4)	T4OIE (\$0F.4)		IRQ4*	IRQ4IF (\$0C.4)	IRQ4IE (\$0D.4)
	Timer5	T5OIF (\$0E.5)	T5OIE (\$0F.5)		IRQ5*	IRQ5IF (\$0C.5)	IRQ5IE (\$0D.5)
Capture	Capture0	CAP0IF (\$0E.6)	CAP0IE (\$0F.6)	Analog	ADC	ADIF (\$0C.7)	ADIE (\$0D.7)
	Capture1	CAP1IF (\$0E.7)	CAP1IE (\$0F.7)		Comparator0	CMP0IF (\$26.0)	CMP0IE (\$27.0)
	Capture2*	CAP2IF (\$0C.0)	CAP2IE (\$0D.0)		Comparator1	CMP1IF (\$26.1)	CMP1IE (\$27.1)
	Capture3*	CAP3IF (\$0C.1)	CAP3IE (\$0D.1)	Communication	SPI	SPIIF (\$26.2)(R) SPIIF (\$3A.7)(R/W)	SPIIE (\$3A.6)
	Capture4*	CAP4IF (\$0C.4)	CAP4IE (\$0D.4)		UART	UARTIF (\$26.3)(R) RXIF (\$48.7)(R) TXIF (\$48.6)(R)	RXIE (\$46.7) TXIE (\$46.6)
	Capture5*	CAP5IF (\$0C.5)	CAP5IE (\$0D.5)		IIC	IICIF(\$26.4) (R) IICIF(\$4B.4)(R/W)	IICIE (\$4A.4)
Watch Dog		WDIF (\$0C.6)	WDIE (\$0D.6)	Interval Timer		ITVALIF (\$26.5)	ITVALIE (\$27.5)

Table 6-1 All of the interrupt resources

External interrupts are coming from INT0~5—the same as IRQ0~5. These INT signals are combined with the edge/level registers and status/control registers to generate the maskable interrupt events to CPU. For all INT channels, each channel has individual interrupt control or status bits. Once an external interrupt is occurred, the flag will be set and stays at set unless user's program clears the flag. The interrupt request signal will be generated in case that the interrupt is enabled. Each INT has a control register used to set the trigger mode of the interrupt event. The trigger mode can be selected as either edge trigger mode or level trigger mode except NMI interrupt. That is, trigger method of NMI interrupt can only be selected from falling edge or rising edge. In other words, level trigger is illegal to NMI interrupt use. When the interrupt channel is enabled with edge trigger mode, an active transition edge on the external interrupt inputs will generate the interrupt. If the channel is enabled with level trigger mode, the active level of the external interrupt inputs will set the interrupt event until the active level condition is removed. In order prevent the external interrupt from being abnormally setting, these selection register including P_IRQ_Opt0 or P_IRQ_Opt1 need to write twice for proper activation.

In SPMC65X family's devices, there are maximum 6 external interrupts. But the amount is different from different body. For example, SPMC65P2404 has only 4 external interrupts, INT0~INT3. Only one of the six external interrupts can be configured as the source of non-maskable interrupt (NMI). As this setting is set to the EEPROM of the real chip, it will never be changed. The method of using "NMI" is showed in Figure 6-1. Its illustration is based on SPMC65P2404. If a general external interrupt, choose the "Disabled" items. If NMI, please select a corresponding interrupt.

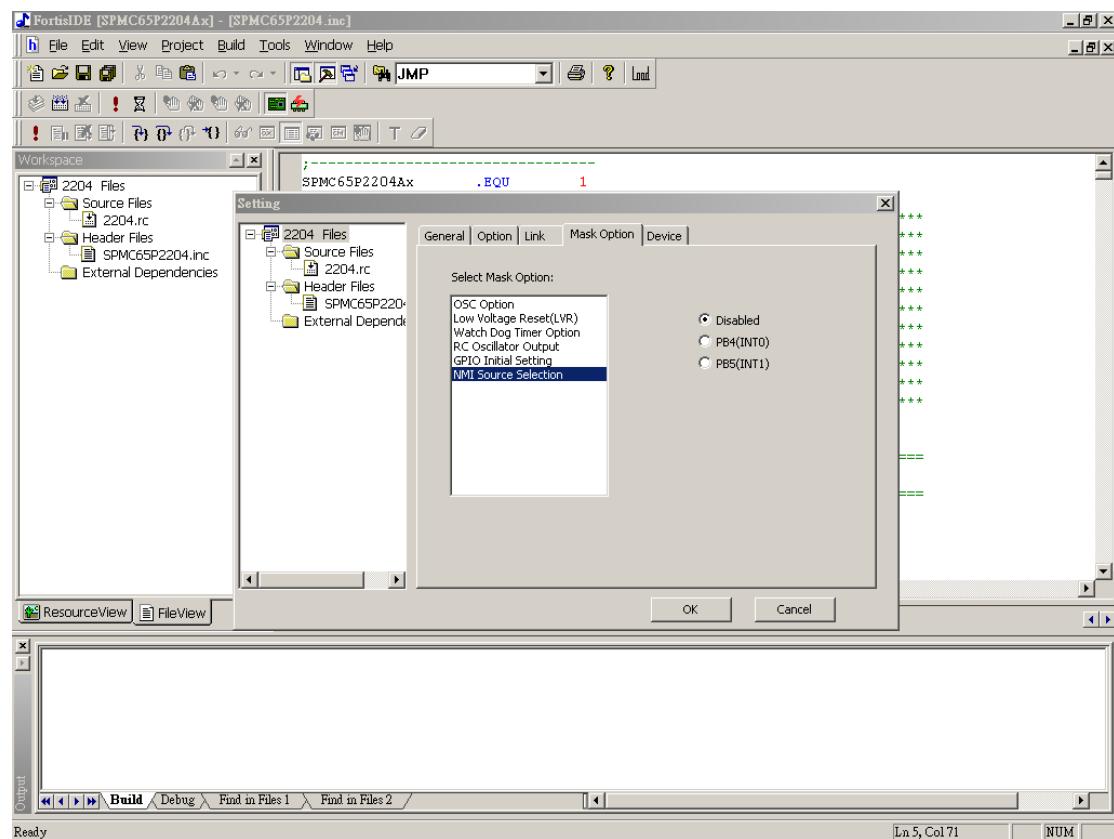


Figure 6-1 NMI Source Selection—based on SPMC65P2404

The Timer Interrupt, Real-Time Interrupt, Watchdog Interrupt, ADC Interrupt, Capture Interrupt, comparator interrupt, and communication interrupt will be described in detail in corresponding section.

6.2 Control Register

6.2.1 Interrupt Flag Register 0 (P_INT_Flag0, \$0C) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
ADIF	WDIF	IRQ5IF/ CAP5IF	IRQ4IF/ CAP4IF	IRQ3IF	IRQ2IF	IRQ1IF/ CAP3IF	IRQ0IF/ CAP2IF
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit 7 **ADIF:** ADC Interrupt flag

0 = No event

1 = Event has occurred

Bit 6 **WDIF:** Watchdog Interrupt flag

0 = No event

1 = Event has occurred

Bit 5 **IRQ5F/CAP5F:** IRQ5/CAP5 Interrupt flag

0 = No event

1 = Event has occurred

Bit 4 **IRQ4F/CAP4F:** IRQ4/CAP4 Interrupt flag

0 = No event

1 = Event has occurred

Bit 3 **IRQ3IF:** IRQ3 Interrupt flag

0 = No event

1 = Event has occurred

Bit 2 **IRQ2IF:** IRQ2 Interrupt flag

0 = No event

1 = Event has occurred

Bit 1 **IRQ1IF/CAP3IF:** IRQ1/CAP3 Interrupt flag

0 = No event

1 = Event has occurred

Bit 0 **IRQ0IF/CAP2IF:** IRQ0/CAP2 Interrupt flag

0 = No event

1 = Event has occurred

Note:

1. This flag is cleared by writing the corresponding bit by “1”.

2. IRQ5, 4 status bits are shared with CAP5, 4 status bits.

3. IRQ1, 0 status bits are shared with CAP3, 2 status bits.

6.2.2 Interrupt Control Register 0 (P_INT_Ctrl0, \$0D) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
ADIE	WDIE	IRQ5E/ CAP5E	IRQ4E/ CAP4E	IRQ3IE	IRQ2IE	IRQ1IE/ CAP3IE	IRQ0IE/ CAP2IE
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit 7 **ADIE:** AD Converter Interrupt enable bit

0 = Interrupt disable

1 = Interrupt enable

Bit 6 **WDIE:** WDT Interrupt enable bit

0 = Interrupt disable

1 = Interrupt enable

Bit 5 **IRQ5IE/CAP5IE:** IRQ5/CAP5 Interrupt enable bit

0 = Interrupt disable

1 = Interrupt enable

Bit 4 **IRQ4IE/CAP4IE:** IRQ4/CAP4 Interrupt enable bit

0 = Interrupt disable

1 = Interrupt enable

Bit 3 **IRQ3IE:** IRQ3 Interrupt enable bit

0 = Interrupt disable

1 = Interrupt enable

Bit 2 **IRQ2IE:** IRQ2 Interrupt enable bit

0 = Interrupt disable

1 = Interrupt enable

Bit 1 **IRQ1IE/ CAP3IE:** IRQ1/CAP3 Interrupt enable bit

0 = Interrupt disable

1 = Interrupt enable

Bit 0 **IRQ0IE/ CAP2IE:** IRQ0/CAP2 Interrupt enable bit

0 = Interrupt disable

1 = Interrupt enable

Note: IRQ5, 4 control bits are shared with CAP5, 4 control bits.

Note: IRQ1, 0 control bits are shared with CAP3, 2 control bits.

6.2.3 Interrupt Flag Register 1 (P_INT_Flag1, \$0E) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
CAP1IF	CAP0IF	T5OIF	T4OIF	T3OIF	T2OIF	T1OIF	T0OIF
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit 7 **CAP1IF:** Capture 1 Interrupt flag

0 = No event

1 = Event has occurred

Bit 6 **CAP0IF:** Capture 0 Interrupt flag

0 = No event

1 = Event has occurred

Bit 5 **T5OIF:** Timer5 Overflow Interrupt flag

0 = No event

1 = Event has occurred

Bit 4 **T4OIF:** Timer4 Overflow Interrupt flag

0 = No event

1 = Event has occurred

Bit 3 **T3OIF:** Timer3 Overflow Interrupt flag

0 = No event

1 = Event has occurred

Bit 2 **T2OIF:** Timer2 Overflow Interrupt flag

0 = No event

1 = Event has occurred

Bit 1 **T1OIF:** Timer1 Overflow Interrupt flag

0 = No event

1 = Event has occurred

Bit 0 **T0OIF:** Timer0 Overflow Interrupt flag

0 = No event

1 = Event has occurred

Note: This flag is cleared by writing the corresponding bit by "1".

6.2.4 Interrupt Control Register 1 (P_INT_Ctrl1, \$0F) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
CAP1IE	CAP0IE	T5OIE	T4OIE	T3OIE	T2OIE	T1OIE	T0OIE
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit 7 **CAP1IE:** CAP1 Interrupt enable bit

	0 = Interrupt disable
	1 = Interrupt enable
Bit 6	CAP0IE: CAP0 Interrupt enable bit
	0 = Interrupt disable
	1 = Interrupt enable
Bit 5	T5OIE: Timer5 Overflow Interrupt enable bit
	0 = Interrupt disable
	1 = Interrupt enable
Bit 4	T4OIE: Timer4 Overflow Interrupt enable bit
	0 = Interrupt disable
	1 = Interrupt enable
Bit 3	T3OIE: Timer3 Overflow Interrupt enable bit
	0 = Interrupt disable
	1 = Interrupt enable
Bit 2	T2OIE: Timer2 Overflow Interrupt enable bit
	0 = Interrupt disable
	1 = Interrupt enable
Bit 1	T1OIE: Timer1 Overflow Interrupt enable bit
	0 = Interrupt disable
	1 = Interrupt enable
Bit 0	T0OIE: Timer0 Overflow Interrupt enable bit
	0 = Interrupt disable
	1 = Interrupt enable

6.2.5 Interrupt Flag Register 2 (P_INT_Flag2, \$26) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
-	-	ITVALIF	IIC_IF	UARTIF	SPIIF	CMP1IF	CMP0IF
-	-	R/W	R	R	R	R/W	R/W
0	0	0	0	0	0	0	0

Bit [7:6] Reserved

Bit 5 **ITVALIF:** Interval Timer Overflow Interrupt flag

0 = No event

1 = Event has occurred

Bit 4 **IIC_IF:** IIC interrupt flag

0 = No event

1 = Event has occurred

Bit 3	UARTIF: UART Interrupt flag
	0 = No event
	1 = Event has occurred
Bit 2	SPIIF: SPI Interrupt flag
	0 = No event
	1 = Event has occurred
Bit 1	CMP1IF: Comparator 1 Interrupt flag
	0 = No event
	1 = Event has occurred
Bit 0	CMP0IF: Comparator 0 Interrupt flag
	0 = No event
	1 = Event has occurred

Note: This flag is cleared by writing the corresponding bit by “1”, but SPIIF cannot be clear. It can be cleared by SPIIF bit in the P_SPI_Status.

6.2.6 Interrupt Control Register 2 (P_INT_Ctrl2, \$27) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
-	-	ITVALIE	-	-	-	CMP1IE	CMP0IE
-	-	R/W	-	-	-	R/W	R/W
0	0	0	0	0	0	0	0

Bit [7:6]	Reserved
Bit 5	ITVALIE: Interval Timer Overflow Interrupt enable bit
	0 = Interrupt disable
	1 = Interrupt enable
Bit [4:2]	Reserved
Bit 1	CMP1IE: Comparator 1 interrupt enable bit
	0 = Interrupt disable
	1 = Interrupt enable
Bit 0	CMP0IE: Comparator 0 interrupt enable bit
	0 = Interrupt disable
	1 = Interrupt enable

6.2.7 IRQ and Capture Option Register 0 (P_IRQ_Opt0, \$33) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
-	-	-	-	IRQ5ES/ CAP5ES	IRQM5	IRQ4ES/ CAP4ES	IRQM4
-	-	-	-	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit [7:4] Reserved

Bit 3 **IRQ5ES/CAP5ES:** Polarity control of INT5 or Capture 5 bit

IRQ5 **CAP5**

IRQM5=1: (level trigger) IRQM5=0: (edge trigger) 1= Falling edge clear counter

1=High level trigger 1=Rising edge trigger 0=Rising edge clear counter

0=Low level trigger 0=Falling edge trigger

Bit 2 **IRQM5:** INT5 trigger mode selection bit

1= Level trigger

0= Edge trigger

Bit 1 **IRQ4ES/CAP4ES:** Polarity control of INT4 or Capture 4 bit

IRQ4 **CAP4**

IRQM4=1: (level trigger) IRQM4=0: (edge trigger) 1= Falling edge clear counter

1=High level trigger 1=Rising edge trigger 0=Rising edge clear counter

0=Low level trigger 0=Falling edge trigger

Bit 0 **IRQM4:** INT4 trigger mode selection bit

1= Level trigger

0= Edge trigger

Note: All of above bits need write twice to set corresponding ones.

6.2.8 IRQ and Capture Option Register 1 (P_IRQ_Opt1, \$34) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
IRQ3ES	IRQM3	IRQ2ES	IRQM2	IRQ1ES/ CAP3ES	IRQM1	IRQ0ES/ CAP2ES	IRQM0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit 7 **IRQ3ES:** Polarity Control of IRQ3

IRQ3

IRQM3=1: (level trigger) IRQM3=0: (edge trigger)

1=High level trigger 1=Rising edge trigger

0=Low level trigger 0=Falling edge trigger

Bit 6 **IRQM3:** IRQ3 function select bit

0 = Edge trigger

1 = Level trigger

Bit 5 **IRQ2ES:** Polarity Control of IRQ2

IRQ2

IRQM2=1: (level trigger) IRQM2=0: (edge trigger)

1=High level trigger 1= Rising edge trigger

0=Low level trigger 0= Falling edge trigger

Bit 4 **IRQM2:** IRQ2 function select bit

0 = Edge trigger

1 = Level trigger

Bit 3 **IRQ1ES/CAP3ES:** Polarity Control of IRQ1/CAP3ES

IRQ1

CAP3

IRQM1=1: (level trigger) IRQM1=0: (edge trigger) 1= Falling edge clear counter

1=High level trigger 1= Rising edge trigger 0= Rising edge clear counter

0=Low level trigger 0= Falling edge trigger

Bit 2 **IRQM1:** IRQ1 function select bit

0 = Edge trigger

1 = Level trigger

Bit 1 **IRQ0ES/CAP2ES:** Polarity Control of IRQ0//CAP2ES

IRQ0

CAP2

IRQM0=1: (level trigger) IRQM0=0: (edge trigger) 1= Falling edge clear counter

1=High level trigger 1= Rising edge trigger 0= Rising edge clear counter

0=Low level trigger 0= Falling edge trigger

Bit 0 **IRQM0:** IRQ0 trigger mode select bit

0 = Edge trigger

1 = Level trigger

Note: All of above bits need write twice to set corresponding ones.

6.3 Operation

1. Select NMI or a general external interrupt resource from Fortis IDE's mask option.
2. Interrupt Disable flag (I-flag) is set to "1" by using "SEI" instruction.
3. Interrupt control register is enabled by writing the corresponding bit by "1".
4. Using 'CLI' instruction to enable all interrupt function.
5. Now the interrupt will be accepted.

During an interrupt, only the return Program Counter (PC) and Status Register (P) are automatically saved on the stack. Typically, users may wish to save key registers during an interrupt, (i.e. A register, X register, Y register). This will have to be implemented in software.

Once in the interrupt service, the source of the interrupt can be determined by polling the interrupt flag bits. The interrupt flag bit must be cleared in software before re-enabling interrupts to avoid recursive interrupt. The interrupt service routine is terminated upon execution of an interrupt return instruction [RTI].

[Example 6-1]: Enable IRQ1 interrupt

lda	#\$FF	
sta	P_INT_Flag0	; clear INT request flag
lda	#C_INT_IRQ1E	; enable IRQ1 INT (INT1).
sta	P_INT_Ctrl0	
cli		; enable INT

6.4 Design Tips

[Example 6-2] : Shows how to enable IRQ0 interrupt using falling edge trigger based on SPMC65P2404A. Please refer to an example of Fortis IDE.

- In Fortis IDE, set [Fortisà Projectà Settingà Mask optionà NMI Source Selection] as “Disabled”.
- Set PB4 as input high for INT0
- IRQ0 is falling edge trigger
- Enable IRQ0 interrupt

```
.SYNTAX 6502 ; process standard 6502 addressing syntax
.LINKLIST ; generate linklist information
.SYMBOLS ; generate symbolic debug information

.INCLUDE      spmc65p2404a.inc

        .PAGE0 ; define values in the range from $00 to $FF
;
        .DATA ; define data storage section
;
        .CODE
; ****
;*
;* Power on Reset Process - Main Program *
;*
; ****

V_Reset:
        sei ; Disable interrupt
        ldx #C_STACK_BOTTOM ; Initial stack pointer at $00FF
        txs ; Transfer to stack point
;
        lda #$FF ; Clear Reset flag
        sta P_SYS_Ctrl
        sta P_SYS_Ctrl
;
        jsr F_InitIOPort ; initial GPIO port
;
        lda #$00
        sta P_IOB_Attrib
;
        lda #00010000B
        sta P_IOB_Data
```

```

    lda      #11101111B
    sta      P_IOB_Dir           ; set PB4 as input high for INT0
;
    lda      #$00
    sta      P_IRQ_Opt1
    sta      P_IRQ_Opt1           ; IRQ0 is falling edge trigger
;
    lda      #$FF                ; clear INT request flag
    sta      P_INT_Flag0
    sta      P_INT_Flag1
    set      P_INT_Ctrl0, CB_INT_IRQ0E ; enable IRQ0 interrupt
    cli      ; enable INT
;
L_Main:
    nop
    jmp      L_Main
;
;
; *****
; *          *
; *      IRQ Interrupt Service Routine      *
; *          *
; *****
;
V_IRQ:
    pha      ; push A register
    txa      ; transfer X to A
    pha      ; push A register (ie. push X)
;
; Timer 1 overflow interrupt ?
;
    lda      P_INT_Flag0
    and      #C_INT_T0OIF
    beq      L_exit_irq
;
    set      P_INT_Flag0, CB_INT_T0OIF
;
    lda      P_IOA_Data
    eor      #$FF
    sta      P_IOA_Data           ; toggle P_IOA_Data
;
L_exit_irq:
    pla      ; pop A register
    tax      ; transfer A to X
    pla      ; pop A register (ie. pop X)
;

```

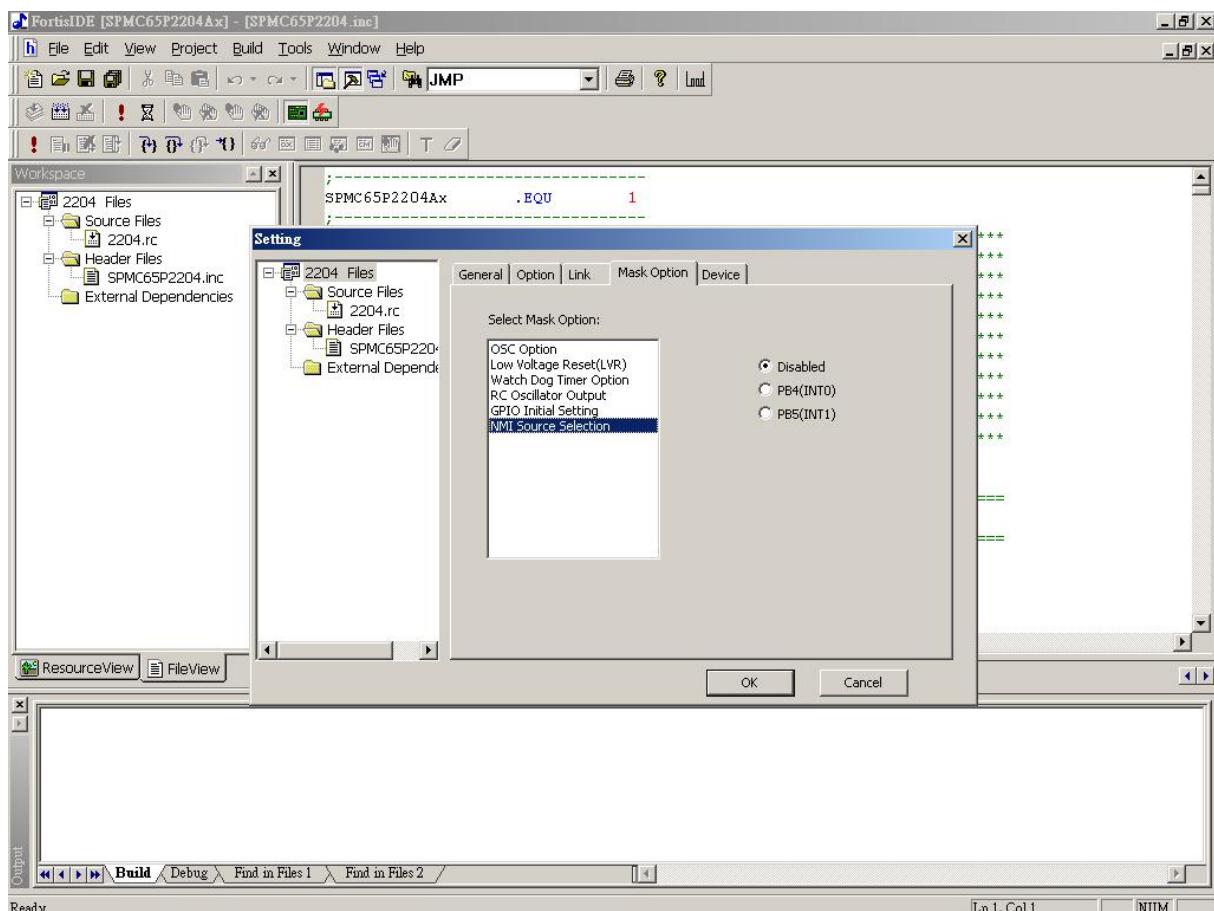
```
rti

; ****
; *          *
; *      Interrupt Vector Table      *
; *          *
; ****
; VECTOR:      .SECTION
    DW      V_NMI           ; may download program emulated either
    DW      V_Reset          ; in internal memory or external memory
    DW      V_IRQ            ; dw define two bytes interrupt vector
;
; ****
; *          *
; *      End Of Interrupt Vector Table      *
; *          *
; ****
.END           ; end of program
```

[Example 6-3] : Based on SPMC65P2404A, please refer to an example of Fortis IDE.

Enable NMI interrupt using falling edge trigger.

- In Fortis IDE, set [Fortisà Projectà Settingà Mask optionà NMI Source Selection] as “PB4(INT0)”.
- Set PB4 as input low for INT0
- IRQ0 is falling edge trigger
- Enable IRQ0 interrupt



```

.SYNTAX 6502          ; process standard 6502 addressing syntax
LINKLIST               ; generate linklist information
.SYMBOLS               ; generate symbolic debug information

.INCLUDE    spmc65p2404a.inc

.PAGE0                 ; define values in the range from $00 to $FF
;
.DATA                  ; define data storage section
;
.CODE
;=====
;=                      =
;= Power on Reset Process - Main Program   =
;=                      =
;=====

V_Reset:
    sei                  ; Disable interrupt
    ldx      #C_STACK_BOTTOM ; Initial stack pointer at $00FF
    txs                  ; Transfer to stack point

```

```

;
    lda      #$FF           ; Clear Reset flag
    sta      P_SYS_Ctrl
    sta      P_SYS_Ctrl
;
    jsr      F_InitIOPort   ; initial GPIO port
;
    lda      #$00
    sta      P_IOB_Attrib
    sta      P_IOB_Data
;
    lda      #11101111B
    sta      P_IOB_Dir       ; set PB4 as input low for INT0
;
    lda      #C_IRQOpt1 IRQ0ES
    sta      P_IRQ_Opt1
    sta      P_IRQ_Opt1       ; IRQ0 is rising edge trigger
;
    lda      #$FF           ; clear INT request flag
    sta      P_INT_Flag0
    sta      P_INT_Flag1
    set      P_INT_Ctrl0, CB_INT_IRQ0E ; enable IRQ0 interrupt
    cli      ; enable INT
;
L_Main:
    nop
    jmp      L_Main
;
;=====
;=          =
;= NMI Interrupt Service Routine      =
;=          =
;=====

V_NMI:
    pha      ; push A register
    txa      ; transfer X to A
    pha      ; push A register (ie. push X)
;
; IRQ0 interrupt ?
;
    lda      P_INT_Flag0
    and      #C_INT_IRQ0F
    beq      L_exit_nmi
;

```

```
set      P_INT_Flag0, CB_INT_IRQ0F
;
lda      P_IOA_Data
eor      #$FF
sta      P_IOA_Data          ; toggle P_IOA_Data
;
L_exit_nmi:
pla      ; pop A register
tax      ; transfer A to X
pla      ; pop A register (ie. pop X)
;
rti

;=====
;=          =
;=      Interrupt Vector Table      =
;=          =
;=====

VECTOR:    .SECTION
DW        V_NMI           ; may download program emulated either
DW        V_Reset          ; in internal memory or external memory
DW        V_IRQ            ; dw define two bytes interrupt vector
;
;=====
;=          =
;=      End Of Interrupt Vector Table      =
;=          =
;=====

.END       ; end of program
```

6.5 Related Application Notes and Libraries

This section lists some application notes and libraries about various usage of SPMC65X family. The application notes may help user develop quickly and offer the concepts about the usage of this topic. Sunplus also supports some powerful libraries about general operation, mathematics, communication, etc. The purpose of library is to help user speed up development. The current application notes and libraries related to Interrupt are listed below.

Application Note:

Title	Application Note Series Number
Infrared Remote control	AN_O0320.DOC
RF communication	AN_O0321.DOC
Zero cross detection	AN_O0322.DOC

Library:

Title	Application Note Series Number
SPMC65X series software basics manual	AN_O0100.DOC
SPMC65X series software advances 1 manual (data processing operation)	AN_O0101.DOC
SPMC65X series software advances 2 manual (mathematical operation)	AN_O0102.DOC

6.6 Revision History

Revision	Date	Remark
V0.1	03/31/2005	First edition

7 Clock Source

7.1 Description

The SPMC65X family support three types of clock resources: Crystal Resonator, RC oscillator, and external clock input. These oscillator options are made available to allow a single device type flexibility to fit application with different oscillator requirements. The system clock (F_{sys}) required for SPMC65X to execute instructions and for the peripherals to function is derived from an extenr clock (F_{osc}) generated in one of three type oscillator modes listed below. Note that the maximum crystal frequency or external clock frequency is equal to 16Mhz and half of F_{osc} is F_{sys} . F_{osc} means input crystal frequency or external clock frequency.

7.2 Operation

Different types of clock sources can be selected by setting device configuration registers. When using Sunplus Fortis IDE, device configuration registers can be set in [Projectà Settingà Mask Optionà OSC Option] Figure 7-1.

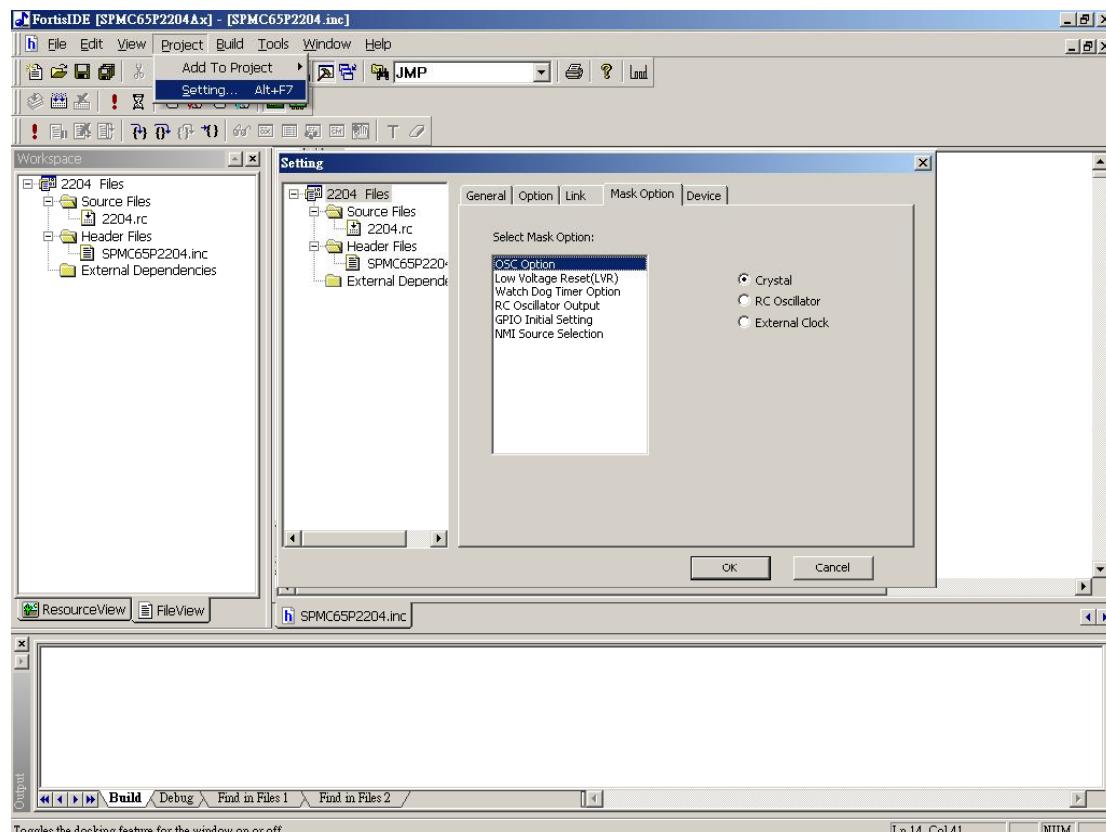


Figure 7-1 Clock Source Option Window

The design of application circuit should follow the vendors' specifications or recommendations. The diagram,

Figure 7-2, listed below represents typical X'TAL/ROSC circuits for most applications:

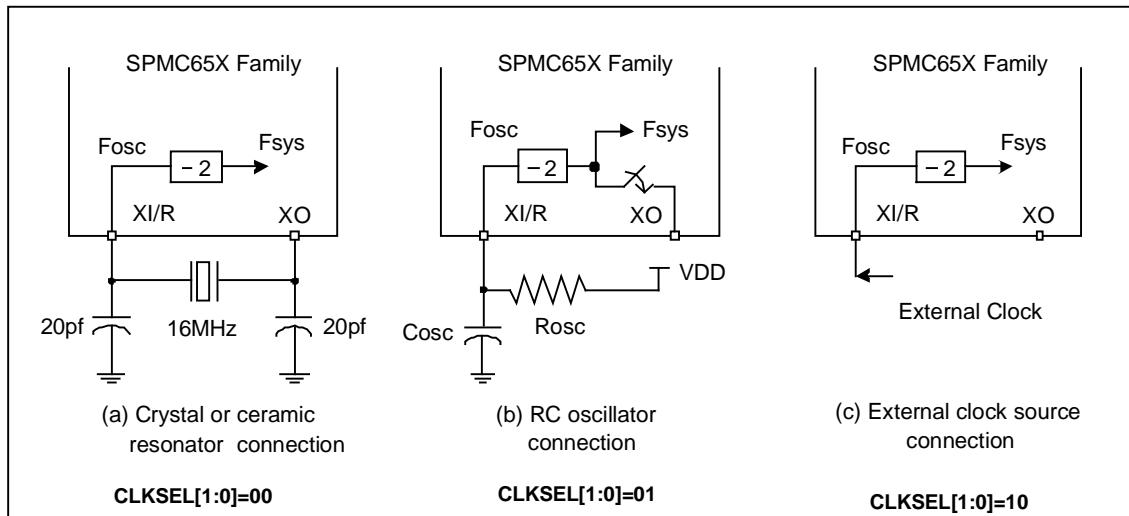


Figure 7-2 Three Types of Clock Sources

The clock source modes are:

1. Crystal or Resonator or Oscillator
2. RC Oscillator
3. External Clock

In Crystal or Resonator or Oscillator mode:

- Using Fortis IDE, in Figure 7-1, select [Crystal] in [OSC option].
- Connect a Crystal, Resonator or Oscillator and two 20pF capacities with "XI" and "XO" pins, showed as Figure 7-2 (a).
- If choosing frequency is F_{osc} , the system frequency F_{sys} is equal to $F_{osc}/2$. For example, if $F_{osc} = 16MHz$, $F_{sys} = 8MHz$.

In R-Oscillator mode:

- Using Fortis IDE, in Figure 7-1, select [RC Oscillator] in [OSC option].
- Connect a resistance and a capacity with "XI" pin, showed as Figure 7-2 (b).
- The operation clock can be outputted at "XO" pin when device configuration registers is set in [Mask Optionà RC Oscillator Output] as [Clock Output], Figure 7-3.
- The relation between the value of R-Oscillator registers and system frequency (F_{sys}) are listed in Table 7-1.

In External Clock mode:

- Using Fortis IDE, in Figure 7-1, select [External Clock] in [OSC option].
- Connect an External Clock source with “XI” pin, showed as Figure 7-2 (c).
- If choosing frequency is F_{OSC} , the system frequency F_{SYS} is equal to $F_{OSC}/2$.

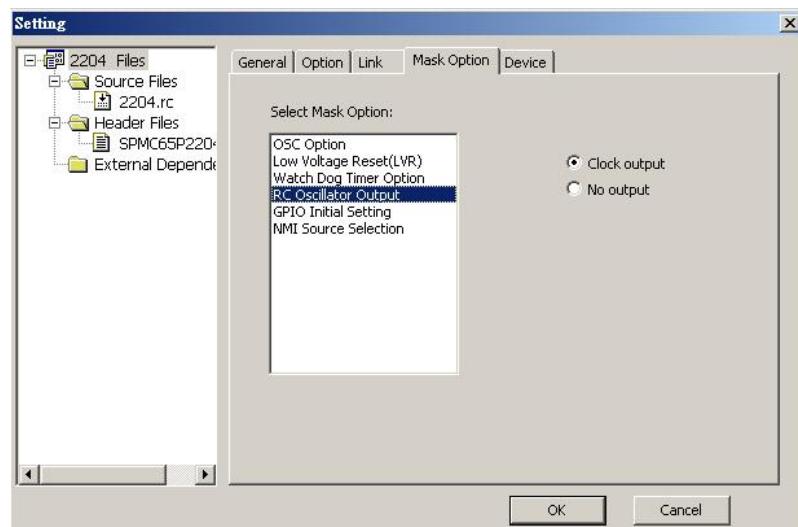


Figure 7-3 RC Oscillator Output

Frequency (Hz), VDD=5.0V	Resistor (Ω), C=50PF
$F_{SYS} = 135K$	75k
$F_{SYS} = 195K$	51k
$F_{SYS} = 480K$	20k
$F_{SYS} = 0.9M$	10k
$F_{SYS} = 1.6M$	5.1k
$F_{SYS} = 2.25M$	3.3k

Table 7-1 R-Oscillator Resistor VS Frequency table

7.3 Design Tips

1. In R-Oscillator mode, the operation clock may have $\pm 15\%$ deviation. See ELECTRICAL CHARACTERISTICS in detail.

7.4 Related Application Notes and Libraries

This section lists some application notes and libraries about various usage of SPMC65X family. The application notes may help user develop quickly and offer the concepts about the usage of this topic. Sunplus also supports some powerful libraries about general operation, mathematics, communication, etc. The purpose of library is to help user speed up development. The current application notes and libraries related to Oscillator are listed below.

Application Note:

Title	Application Note Series Number
© Sunplus Technology Co., Ltd.	PAGE 7-3



Oscillator type selection AN_O0330.DOC

Library:

Title	Application Note Series Number
SPMC65X series software basics manual	AN_O0100.DOC
SPMC65X series software advances 1 manual (data processing operation)	AN_O0101.DOC
SPMC65X series software advances 2 manual (mathematical operation)	AN_O0102.DOC

7.5 Revision History



SPMC65X Family PROGRAMMING GUIDE

Revision	Date	Remark
V0.1	03/31/2005	First edition

8 I/O Ports

8.1 Description

The SPMC65X family have six ports, Port A, Port B, Port C, Port D, Port E and Port F. These ports pins may be multiplexed with alternate functions for the peripheral features on the device. In general, when an initial reset state, all ports are used as a general purpose input port.

These I/O structures contain three parts: data, direction and attribution registers. Each corresponding bit in these ports should be given a value. For a step-by-step procedure on how to set up the I/O configuration, see Section 8.3 in detail.

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 / not float.
- c. The data setting affects the initial content of the pin. For inputs, it also determines the pull high or pull low setting. For outputs, it determines the output value.

The I/O configuration is set with control registers are shown in Table 8-1. N open drain can be implemented by setting (Dir Attrib Data) 110, then 100; or 100, then 110. P open drain can be implemented by setting (Dir Attrib Data) 111, then 101; or 101, then 111.

Dir	Attrib	Data	Function	Description
0	0	0	Pull Low	Input with pull-low
0	0	1	Pull High	Input with pull-high
1	0	1	Output High	Output Data
1	0	0	Output Low	Output Data
X	1	X	Float	Input with float
Others			Reserved	

Table 8-1 I/O Port configuration

The I/O PORT X (X=A~F) has 8 programmable I/Os that are controlled by I/O register P_IOX_Data, direction control register P_IOX_Dir, attribution control register P_IOX_Attrib, and data register P_IOX_Buf. P_IOX_Data is used to access Port I/O. Reading P_IOX_Data will get the pad status, and writing P_IOX_Data will store the data into P_IOX_Buf. P_IOX_Buf is a dedicated register to store the PX data and bit operation instruction should be applied in this register instead of P_IOX_Data. For example, writing the output data into P_IOX_Buf or P_IOX_Data has the same result exactly but the reading path is different. The P_IOX_Buf will be altered

incorrectly if the bit operations SET, TST, CLR and INV are performed on P_IOX_Data. Therefore, it is strongly suggested that user perform the bit operations using P_IOX_Buf (X=A,B,C,D,E,F) exclusively. PX[7:0] have pull-up/down resistors that can be configured with pull-up or pull-down resistors.

Figure 8-1 shows a typical I/O schematic. Figure 8-2 shows the GPIO initial setting, if “All Float”, the statuses of all GPIOs are floating when going through the POR. Otherwise, they are all pulling low.

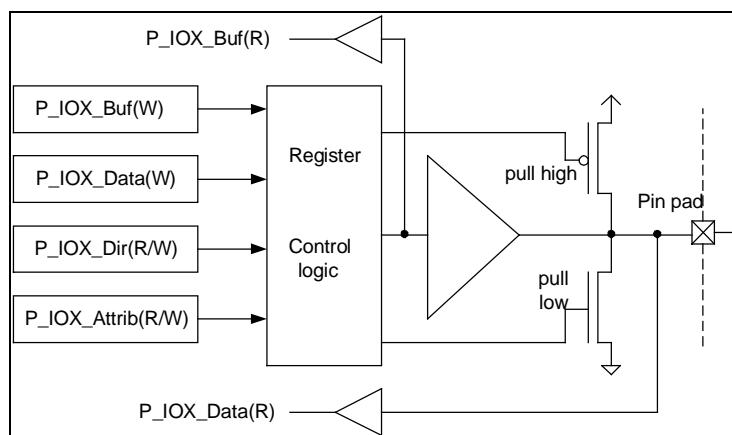


Figure 8-1 I/O schematic

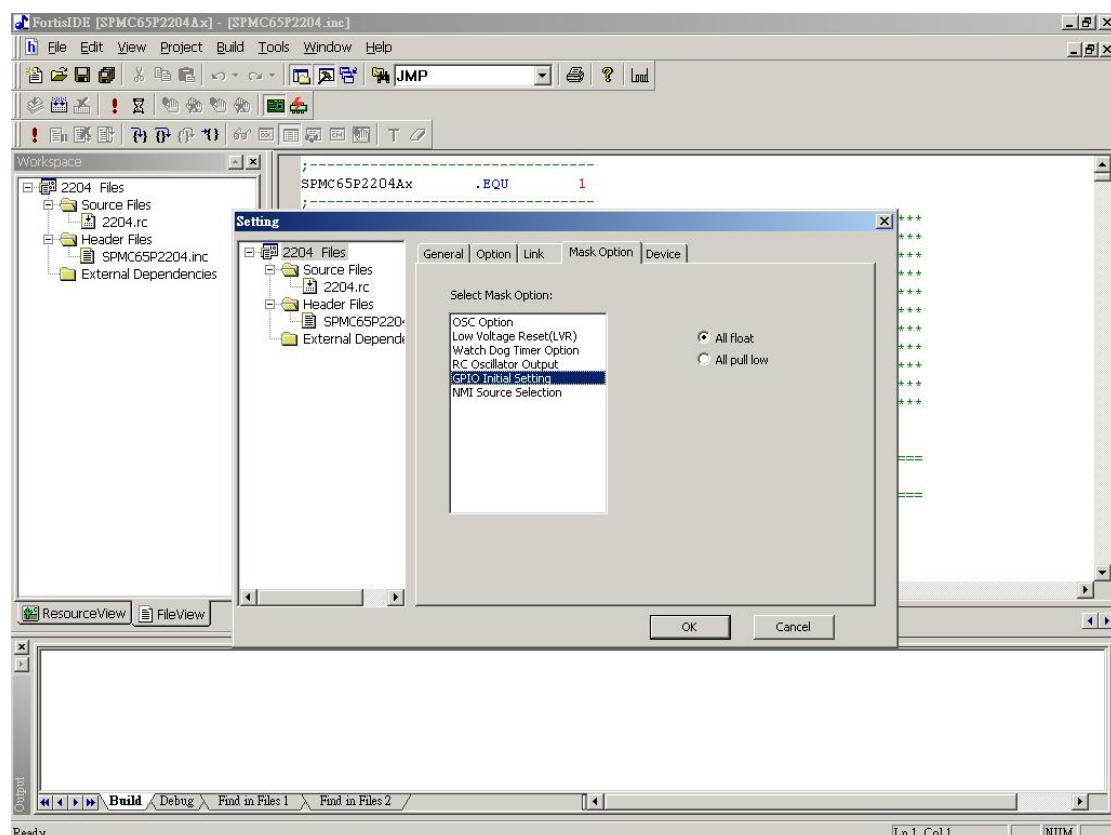


Figure 8-2 GPIO initial setting

8.1.1 Port A group

Port A's configurable pull-up/down resistors can be selected by I/O configuration registers (P_IOA_Data, P_IOA_Buf, P_IOA_Dir, and P_IOA_Attrib). In addition, each pin of Port A is shared with AD converter input or IRQ0 input. The default function of Port A is for I/O use. The corresponding pads on Port A serve the various features shown in Table 8-2 and the more descriptions of alternate functions A are explained in corresponding sections.

PIN	Rp	IN	OUT	Special Function
PA7	100K Up/Down	Schmitt-Trigger	4/10mA	AD7 analog input
PA6	100K Up/Down	Schmitt-Trigger	4/10mA	AD6 analog input
PA5	100K Up/Down	Schmitt-Trigger	4/10mA	AD5 analog input
PA4	100K Up/Down	Schmitt-Trigger	4/10mA	AD4 analog input
PA3	100K Up/Down	Schmitt-Trigger	4/10mA	AD3 analog input
PA2	100K Up/Down	Schmitt-Trigger	4/10mA	AD2 analog input
PA1	100K Up/Down	Schmitt-Trigger	4/10mA	AD1 analog input
PA0	100K Up/Down	Schmitt-Trigger	4/10mA	AD0 analog input

Table 8-2 Special Function of Port A

8.1.2 Port B Group

Port B's configurable pull-up/down resistors can be selected by I/O configuration registers (P_IOB_Data, P_IOB_Buf, P_IOB_Dir, and P_IOB_Attrib). In addition, total pins of Port B are shared with some special functions listed in Table 8-3. For example, PB7 can be used as AD8 analog input or ADC top reference voltage input pin. PB6 can be used as buzzer output pin, PB[5:4] can be used as external clock or interrupt input pins, PB[3:2] can be used as compare or PWM output pins and PB[3:2] can be used as capture or external clock input pins.

Take slew rate function for example, if slew rate function on PB[7:6] needed to be enable, programming SLOWE bit in P_IO_Opt (\$35.0) to '1' will enforce the output high to low transition time to 250ns ± 20% with 50pf pad loading at 2.0MHz CPU frequency. The corresponding pads on Port B serve the function of the various following feature in Table 8-3 and the more descriptions of alternative functions are explained in corresponding sections.

PIN	Rp	IN	OUT	Special Function
PB7	100K Up/Down	Schmitt-Trigger	4/20mA	AD8 analog input or ADC top reference voltage
PB6	100K Up/Down	Schmitt-Trigger	4/20mA	Buzzer output
PB5	100K Up/Down	Schmitt-Trigger	4/10mA	External interrupt 1 input/ Timer3 external clock input
PB4	100K Up/Down	Schmitt-Trigger	4/10mA	External interrupt 0 input/ Timer2 external clock input
PB3	100K Up/Down	Schmitt-Trigger	4/10mA	Timer1 compare output / PWM output

PB2	100K Up/Down	Schmitt-Trigger	4/10mA	Timer0 compare output / PWM output
PB1	100K Up/Down	Schmitt-Trigger	4/10mA	Capture1 event input to Timer1/ Timer1 external clock input
PB0	100K Up/Down	Schmitt-Trigger	4/10mA	Capture0 event input to Timer0/ Timer0 external clock input

Table 8-3 Special Function of Port B

8.1.3 Port C Group

Port C's configurable pull-up/down resistors can be selected by I/O configuration registers (P_IOC_Data, P_IOC_Buf, P_IOC_Dir, and P_IOC_Attrib). Except the general-purpose I/O, PC[7:0] also have alternative function with proper setting. For Example, PC[7:6] can be used as IIC, PC[5:4] can be used as UART interface, and PC[3:0] can be used as SPI interface. The corresponding pads on Port C serve the function of the various following feature in Table 8-4 and the more descriptions of alternative functions are explained in corresponding sections.

PIN	R _p	IN	OUT	Special Function
PC7	100K Up/Down	Schmitt-Trigger	4/10mA	IIC bus data input /output
PC6	100K Up/Down	Schmitt-Trigger	4/10mA	IIC bus clock input
PC5	100K Up/Down	Schmitt-Trigger	4/10mA	UART Receive signal
PC4	100K Up/Down	Schmitt-Trigger	4/10mA	UART Transfer signal
PC3	100K Up/Down	Schmitt-Trigger	4/10mA	SPI data output
PC2	100K Up/Down	Schmitt-Trigger	4/10mA	SPI data input
PC1	100K Up/Down	Schmitt-Trigger	4/10mA	SPI clock output / clock input
PC0	100K Up/Down	Schmitt-Trigger	4/10mA	SPI chip slave select bit

Table 8-4 Special Function of Port C

8.1.4 Port D Group

Port D's configurable pull-up/down resistors can be selected by I/O configuration registers (P_IOD_Data, P_IOD_Buf, P_IOD_Dir, and P_IOD_Attrib). Except the general-purpose I/O, PD[7:0] also have alternative function with proper setting. For example, PD[7:6] can be used as compare or PWM output pins. PD[5:4] can be used as interrupt or clock input pins, PD[3:2] can be used as compare or PWM output pins and PD[1:0] can be used as external interrupt input pins. The corresponding pads are assigned as Table 8-5 and the more descriptions of alternative functions on Port D are explained in corresponding sections.

PIN	R _p	IN	OUT	Special Function
PD7	100K Up/Down	Schmitt-Trigger	4/10mA	Timer5 compare output / PWM output
PD6	100K Up/Down	Schmitt-Trigger	4/10mA	Timer4 compare output / PWM output

PD5	100K Up/Down	Schmitt-Trigger	4/10mA	External interrupt 5 input/ Timer5 external clock input
PD4	100K Up/Down	Schmitt-Trigger	4/10mA	External interrupt 4 input/ Timer4 external clock input
PD3	100K Up/Down	Schmitt-Trigger	4/10mA	Timer2 compare output / PWM output
PD2	100K Up/Down	Schmitt-Trigger	4/10mA	Timer3 compare output / PWM output
PD1	100K Up/Down	Schmitt-Trigger	4/10mA	External interrupt 3 input
PD0	100K Up/Down	Schmitt-Trigger	4/10mA	External interrupt 2 input

Table 8-5 Special Function of Port D

8.1.5 Port E Group

Port E's configurable pull-up/down resistors can be selected by I/O configuration registers (P_IOE_Data, P_IOE_Buf, P_IOE_Dir, and P_IOE_Attrib). Except the general-purpose I/O, PE[6:2] also have alternative function with proper setting. For example, PE6 can be used as D/A output pin and PE[5:2] can be used for comparator function. The corresponding pads are assigned as Table 8-6 and the more descriptions of alternative functions on Port E are explained in corresponding sections.

PIN	Rp	IN	OUT	Special Function
PE7	100K Up/Down	Schmitt-Trigger	4/10mA	
PE6	100K Up/Down	Schmitt-Trigger	4/10mA	D/A output
PE5	100K Up/Down	Schmitt-Trigger	4/10mA	Comparator1 input
PE4	100K Up/Down	Schmitt-Trigger	4/10mA	Comparator1 reference input
PE3	100K Up/Down	Schmitt-Trigger	4/10mA	Comparator0 input
PE2	100K Up/Down	Schmitt-Trigger	4/10mA	Comparator0 reference input
PE1	100K Up/Down	Schmitt-Trigger	4/10mA	
PE0	100K Up/Down	Schmitt-Trigger	4/10mA	

Table 8-6 Special Function of Port E

8.1.6 Port F Group

Port F's configurable pull-up/down resistors can be selected by I/O configuration registers (P_IOF_Data, P_IOF_Buf, P_IOF_Dir, and P_IOF_Attrib). PF[7:0] are dedicated to general-purpose I/O pins without any alternative function. The corresponding pads are assigned as Table 8-7.

PIN	Rp	IN	OUT	Special Function
PF7	100K Up/Down	Schmitt-Trigger	4/10mA	
PF6	100K Up/Down	Schmitt-Trigger	4/10mA	
PF5	100K Up/Down	Schmitt-Trigger	4/10mA	

PF4	100K Up/Down	Schmitt-Trigger	4/10mA	
PF3	100K Up/Down	Schmitt-Trigger	4/10mA	
PF2	100K Up/Down	Schmitt-Trigger	4/10mA	
PF1	100K Up/Down	Schmitt-Trigger	4/10mA	
PF0	100K Up/Down	Schmitt-Trigger	4/10mA	

Table 8-7 Special Function of Port F

8.1.7 Slew Rate Control

The SPMC65X family support slew rate function on PB6 and PB7 Pins by software configuration. When slew rate enable bit is set to '1', the digital output on PB6 and PB7 will delay on falling edge with approximate 250ns which depended on system clock (F_{SYS}). The benefit of slew rate function is to prevent MCU from electric magnet interfere when long distant transmission. Detailed register setting and example are described as following.

8.2 Control Register

8.2.1 Port A Data Register (P_IOA_Data, \$00)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
P_IOA_Data							
R/W							
00h							

Bit [7:0] P_IOA_Data: Port A Data Register

Read: Read from Port A external I/O pin

Write: Write to Port A data latch (\$59)

8.2.2 Port A Direction Register (P_IOA_Dir, \$04)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
P_IOA_Dir							
R/W							
00h							

Bit [7:0] P_IOA_Dir: Port A Direction Register

0 = Input

1 = Output

8.2.3 Port A Attribute Register (P_IOA_Attrib, \$08)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
P_IOA_Attrib							
R/W							
00h							

Bit [7:0] P_IOA_Attrib: Port A Attribution Register

0 = Not float

1 = Input with float

8.2.4 Port A Data Latch Buffer Register (P_IOA_Buf, \$59) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
P_IOA_Buf							
R/W							
00h							

Bit [7:0] P_IOA_Buf: Port A Data Latch Buffer register

Note: Bit operation instruction should use this register to do operation.

8.2.5 Port B Data Register (P_IOB_Data, \$01)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
P_IOB_Data							
R/W							
00h							

Bit [7:0] P_IOB_Data: Port A Data Register

Read: Read from Port B external I/O pin

Write: Write to Port B data latch (\$5A)

8.2.6 Port B Direction Register (P_IOB_Dir, \$05)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
P_IOB_Dir							
R/W							
00h							

Bit [7:0] P_IOB_Dir: Port B Direction Register

0 = Input

1 = Output

8.2.7 Port B Attribute Register (P_IOB_Attrib, \$09)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
P_IOB_Attrib							
R/W							
00h							

Bit [7:0] P_IOB_Attrib: Port B Attribution Register

0 = Not float

1 = Input with float

8.2.8 Port B Data Latch Buffer Register (P_IOB_Buf, \$5A) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
P_IOB_Buf							
R/W							
00h							

Bit [7:0] P_IOB_Buf: Port B Data Latch Buffer register

Note: Bit operation instruction should use this register to do operation.

8.2.9 Port C Data Register (P_IOC_Data, \$02)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
P_IOC_Data							
R/W							
00h							

Bit [7:0] P_IOC_Data: Port C Data Register

Read: Read from Port C external I/O pin

Write: Write to Port C data latch (\$5B)

8.2.10 Port C Direction Register (P_IOC_Dir, \$06)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
P_IOC_Dir							
R/W							
00h							

Bit [7:0] P_IOC_Dir: Port C Direction Register

0 = Input

1 = Output

8.2.11 Port C Attribute Register (P_IOC_Attrib, \$0A)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
P_IOC_Attrib							
R/W							
00h							

Bit [7:0] P_IOC_Attrib: Port B Attribution Register

0 = Not float

1 = Input with float

8.2.12 Port C Data Latch Buffer Register (P_IOC_Buf, \$5B) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
P_IOC_Buf							
R/W							
00h							

Bit [7:0] P_IOC_Buf: Port B Data Latch Buffer register

Note: Bit operation instruction should use this register to do operation.

8.2.13 Port D Data Register (P_IOD_Data, \$03)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
P_IOD_Data							
R/W							
00h							

Bit [7:0] P_IOD_Data: Port D Data Register

Read: Read from Port D external I/O pin

Write: Write to Port D data latch (\$5C)

8.2.14 Port D Direction Register (P_IOD_Dir, \$07)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
P_IOD_Dir							
R/W							
00h							

Bit [7:0] P_IOD_Dir: Port C Direction Register

0 = Input

1 = Output

8.2.15 Port D Attribute Register (P_IOD_Attrib, \$0B)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
P_IOD_Attrib							
R/W							
00h							

Bit [7:0] P_IOD_Attrib: Port D Attribution Register

0 = Not float

1 = Input with float

8.2.16 Port D Data Latch Buffer Register (P_IOD_Buf, \$5C) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
P_IOD_Buf							
R/W							
00h							

Bit [7:0] P_IOD_Buf: Port B Data Latch Buffer register

Note: Bit operation instruction should use this register to do operation.

8.2.17 Port E Data Register (P_IOE_Data, \$40) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
P_IOE_Data							
R/W							
00h							

Bit [7:0] P_IOE_Data: Port E Data Register

Read: Read from Port E external I/O pin

Write: Write to Port E data latch (\$5D)

8.2.18 Port E Directon Register (P_IOE_Dir, \$42) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
P_IOE_Dir							
R/W							
00h							

Bit [7:0] P_IOE_Dir: Port E Direction Register

0 = Input

1 = Output

8.2.19 Port E Attribute Register (P_IOE_Attrib, \$44) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
P_IOE_Attrib							
R/W							
00h							

Bit [7:0] P_IOE_Attrib: Port D Attribution Register

0 = Not float

1 = Input with float

8.2.20 Port E Data Latch Buffer Register (P_IOE_Buf, \$5D) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
P_IOE_Buf							
R/W							
00h							

Bit [7:0] P_IOE_Buf: Port E Data Latch Buffer register

Note: Bit operation instruction should use this register to do operation.

8.2.21 Port F Data Register (P_IOF_Data, \$41) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
P_IOF_Data							
R/W							
00h							

Bit [7:0] P_IOF_Data: Port E Data Register

Read: Read from Port E external I/O pin

Write: Write to Port E data latch (\$5E)

8.2.22 Port F Direction Register (P_IOF_Dir, \$43) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
P_IOF_Dir							
R/W							
00h							

Bit [7:0] P_IOF_Dir: Port E Direction Register

0 = Input

1 = Output

8.2.23 Port F Attribute Register (P_IOF_Attrib, \$45) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
P_IOF_Attrib							
R/W							
00h							

Bit [7:0] P_IOF_Attrib: Port F Attribution Register

0 = Not float

1 = Input with float

8.2.24 Port F Data Latch Buffer Register (P_IOF_Buf, \$5E) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
P_IOF_Buf							
R/W							
00h							

Bit [7:0] P_IOF_Buf: Port F Data Latch Buffer register

Note: Bit operation instruction should use this register to do operation.

8.2.25 Slew Rate Control Register (P_IO_Opt, \$35)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
-	-	-	-	-	-	-	SLOWE
-	-	-	-	-	-	-	R/W
0	0	0	0	0	0	0	0

Note: This byte must be double-write.

Bit [7:1] Reserved

Bit 0 **SLOWE:** Slew rate controls enable selection bit

1= The slew rate level of PB[7 : 6] is slow

0= The slew rate level of PB[7 : 6] is normal.

(Without slew rate control)

8.3 Operation

- Decided which bits on the port X(X=A~F) would be used as output or input ,and then writing the corresponding bits into P_IOX_Dir register.
- The attribute setting a characteristic to the pin. Some kinds of features could be set to each pin by setting P_IOX_Attrib register, such as input float for input, pull high/low for output.
- The data register setting affects the initial value of pin. On input aspect, it determines the pull high or pull low setting. On output aspect, it determines the output value of pin.
- Note that each of pins supports at most 10mA current to drive except that PB[6:7] pins support at most 20mA.

[Example 8-1]: Set Port X(X=A~F)[7:0] as output with low data.

lda	#\$FF	; store accumulator with \$FF
sta	P_IOX_Dir	; store
lda	#\$00	
sta	P_IOX_Attrib	
sta	P_IOX_Data	

[Example 8-2]: Port X(A~F)[7:0] as Input with pulling Low.

lda	#\$00	; store accumulator with \$00
sta	P_IOX_Dir	; store
sta	P_IOX_Attrib	
sta	P_IOX_Data	

[Example 8-3]: Set Port X[3:0] as Output with Low data, Port X[7:4] as Input with pulling high.

lda	#\$0F	; store accumulator with \$0F
sta	P_IOX_Dir	
lda	#\$00	
sta	P_IOX_Attrib	
lda	#\$F0	
sta	P_IOX_Data	

[Example 8-4]: Set Port X[7:0] as Input with float.

lda	#\$FF	; store accumulator with \$FF
sta	P_IOX_Attrib	

[Example 8-5]: Use bit operation to set Port X[3:2] to high. (Use P_IOX_Buf instead of P_IOX_Data when bit operation)

set	P_IOX_Buf, 3	; Set bit3 to high
set	P_IOX_Buf, 2	; Set bit2 to high

[Example 8-6]: Use bit operation to clear Port X[3:2] to low. (Use P_IOX_Buf instead of P_IOX_Data when bit operation)

clr	P_IOX_Buf, 3	; Set bit3 to low
clr	P_IOX_Buf, 2	; Set bit2 to low

[Example 8-7]: Use bit operation to inverse Port X[2:1]. (Use P_IOX_Buf instead of P_IOX_Data when bit operation)

inv	P_IOX_Buf, 2	; Inverse bit2
inv	P_IOX_Buf, 1	; Inverse bit1

8.4 Design Tips

[Example 8-8] Shows how to initialize I/O port based on SPMC65P2404A. Please refer to an example of Fortis IDE.

n Set PortA, B, C and D as Input high

```
.SYNTAX 6502 ; process standard 6502 addressing syntax
.LINKLIST ; generate linklist information
.SYMBOLS ; generate symbolic debug information

.INCLUDE spmc65p2404a.inc

.PAGE0 ; define values in the range from $00 to $FF
;

.DATA ; define data storage section
;

.CODE
; *****
;*
;* Power on Reset Process - Main Program *
;*
;****

V_Reset:
    sei ; Disable interrupt
    ldx #C_STACK_BOTTOM ; Initial stack pointer at $00ff
    txa ; Transfer to stack point
;
    lda #$FF ; Clear Reset flag
    sta P_SYS_Ctrl
    sta P_SYS_Ctrl
;
    ldx #$00 ; X = $00

; GPIO port initial
    lda #$00
    sta P_IOA_Dir ; IOA is input
    sta P_IOB_Dir ; IOB is input
    sta P_IOC_Dir ; IOC is input
    sta P_IOD_Dir ; IOD is input

    sta P_IOA_Attrib
    sta P_IOB_Attrib
    sta P_IOC_Attrib
    sta P_IOD_Attrib
```

```

;
    lda      #$FF
    sta      P_IOA_Data           ; IOA = input high
    sta      P_IOB_Data           ; IOB = input high
    sta      P_IOC_Data           ; IOC = input high
    sta      P_IOD_Data           ; IOD = input high
;

L_Main:
    nop
    jmp      L_Main
;

; *****
; *          *
; *      Interrupt Vector Table      *
; *          *
; *****
VECTOR:     .SECTION
    DW      V_NMI                 ; may download program emulated either
    DW      V_Reset               ; in internal memory or external memory
    DW      V_IRQ                 ; dw define two bytes interrupt vector
;

; *****
; *          *
; *      End Of Interrupt Vector Table      *
; *          *
; *****
.END        ; end of program

```

8.5 Related Application Notes and Libraries

This section lists some application notes and libraries about various usage of SPMC65X family. The application notes may help user develop quickly and offer the concepts about the usage of this topic. Sunplus also supports some powerful libraries about general operation, mathematics, communication, etc. The purpose of library is to help user speed up development. The current application notes and libraries related to I/O Ports are listed below.

Application Note:

Title	Application Note Series Number
Momentary key (1 x 4) scan	AN_O0301.DOC

Switch key (x 4) + momentary key (4 x 4) scan	AN_O0304.DOC
LED (x 4) + momentary key (4 x 4) scan	AN_O0306.DOC
LED (x 4) + switch key (x 4) + momentary key (4 x 4) scan	AN_O0307.DOC
7segment LED display	AN_O0308.DOC
I/O simulate LCD display	AN_O0309.DOC
I/O simulate SPI communication	AN_O0310.DOC
I/O simulate I2C communication	AN_O0311.DOC
I/O simulate UART communication	AN_O0312.DOC
I/O access EEPROM (93c46)	AN_O0313.DOC
I/O access EEPROM (24c01)	AN_O0314.DOC
I/O extension	AN_O0315.DOC
I/O simulate A/D convert with RC	AN_O0316.DOC
I/O simulate D/A convert with PWM	AN_O0317.DOC
I/O simulate SIN wave output (R2R)	AN_O0318.DOC
I/O simulate melody output	AN_O0319.DOC
I/O configuration	AN_O0329.DOC

Library:

Title	Application Note Series Number
SPMC65X series software basics manual	AN_O0100.DOC
SPMC65X series software advances 1 manual (data processing operation)	AN_O0101.DOC
SPMC65X series software advances 2 manual (mathematical operation)	AN_O0102.DOC

8.6 Revision History

Revision	Date	Remark
V0.1	03/31/2005	First edition

9 Timers

9.1 Description

In SPMC65X family, as many as 6 channels of Timer are equipped and each one has different powerful CCP functions that are the abbreviation of Capture/Compare/PWM. It has four operation modes such as 8-bit/16-bit Timer/Event Counter, Capture, Compare and PWM. In addition, all of timers can be used either 8-bit or 16-bit operation except for PWM function. These functions can be easily configured with corresponding control registers setting. Its function summary is shown as Table 9-1:

	Timer/Event Counter		Capture		Compare		PWM
	8 bit	16 bit	8 bit	16 bit	8 bit	16 bit	
Timer 0	YES	YES	Width/Cycle	Width only	YES	YES	8 bit
Timer 1	YES	YES	Width/Cycle	Width only	YES	YES	12 bit
Timer 2	YES	YES	Width/Cycle	Width only	YES	YES	8 bit
Timer 3	YES	YES	Width/Cycle	Width only	YES	YES	12 bit
Timer 4	YES	YES	Width/Cycle	Width only	YES	YES	8 bit
Timer 5	YES	YES	Width/Cycle	Width/Cycle	YES	YES	16 bit

Table 9-1 Summary of Timer function for SPMC65X family

The timers have three types depending on different features:

1. Type I: (Timer0, Timer2 & Timer4)
 - **8 or 16-bit timer/counter**
 - 8 or 16-bit capture mode (8-bit with width/cycle measurement, 16-bit with width measurement)
 - 8 or 16-bit compare mode
 - 8-bit PWM mode
2. Type II: (Timer1 & Timer3)
 - **8 or 16-bit timer/counter**
 - 8 or 16-bit capture mode (8-bit with width/cycle measurement, 16-bit with width measurement)
 - 8 or 16-bit compare mode
 - 12-bit PWM mode
3. Type III: (Timer5)
 - **8 or 16-bit timer/counter**
 - 8 or 16-bit capture mode (8 or 16-bit with width/cycle measurement)
 - 8 or 16-bit compare mode
 - 16-bit PWM mode

These functions are set by P_TMRx_Ctrl0 registers (x=0~5). In this Chapter, we will introduce timer/counter first, and the others will be done in the following chapters.

9.2 Control Register

9.2.1 Timer0_1 Control Register 0 (P_TMR0_1_Ctrl0, \$11) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
-	T1FC2	T1FC1	T1FC0	-	T0FC2	T0FC1	T0FC0
-	R/W	R/W	R/W	-	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit 7 Reserved

Bit [6:4] **T1FC[2 : 0]**: Timer 1 function configuration bits

111 = 12-bit PWM

110 = 16-bit capture (Width)

101 = 16-bit Compare

100 = 16-bit Timer

011 = 8-bit Capture (Width, Cycle)

010 = 8-bit Compare

001 = 8-bit Timer

000 = Disable

Bit 3 Reserved

Bit [2:0] **T0FC[2 : 0]**: Timer0 Function Configuration bits

111 = 8-bit PWM

110 = 16-bit capture (Width)

101 = 16-bit Compare

100 = 16-bit Timer

011 = 8-bit Capture (Width, Cycle)

010 = 8-bit Compare

001 = 8-bit Timer

000 = Disable

9.2.2 Timer0_1 Control Register 1 (P_TMR0_1_Ctrl1, \$12) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
-	T1PSS2	T1PSS1	T1PSS0	-	T0PSS2	T0PSS1	T0PSS0
-	R/W	R/W	R/W	-	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit 7 Reserved

Bit [6:4] **T1PSS[2 : 0]**: Timer1 Pre-Scale Configuration bits

111 = External Event

110 = $F_{SYS} \div 512$

101 = $F_{SYS} \div 128$

100 = $F_{SYS} \div 32$

011 = $F_{SYS} \div 8$

010 = $F_{SYS} \div 4$

001 = $F_{SYS} \div 2$

000 = F_{SYS}

Bit 3 Reserved

Bit [2:0] **T0PSS[2 : 0]**: Timer0 Pre-Scale Configuration bits

111 = External Event

110 = $F_{SYS} \div 512$

101 = $F_{SYS} \div 128$

100 = $F_{SYS} \div 32$

011 = $F_{SYS} \div 8$

010 = $F_{SYS} \div 4$

001 = $F_{SYS} \div 2$

000 = F_{SYS}

F_{SYS} : Frequency of Clock Source

9.2.3 Timer0 Count Low Byte Register (**P_TMR0_Count**, \$13) (R)

9.2.4 Timer0 Preload Low Byte Register (**P_TMR0_Preload**, \$13) (W)

	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
W	T0PL7	T0PL6	T0PL5	T0PL4	T0PL3	T0PL2	T0PL1	T0PL0
R	T0CN7	T0CN6	T0CN5	T0CN4	T0CN3	T0CN2	T0CN1	T0CN0
	0	0	0	0	0	0	0	0

Bit [7:0] Write: **T0PL** [7:0] Timer0 Pre-Load Value (W)

Bit [7:0] Read: **T0CN** [7:0] Timer0 Count Value T0CN (R)

Note: Writing dataflow: Write high byte first then low byte, i.e., 16-bit timer mode: Writer P_TMR0_PreloadHi first then P_TMR0_Preload.

9.2.5 Timer0 Count High Byte Register (**P_TMR0_CountHi**, \$14) (R)

9.2.6 Timer0 Preload High Byte Register (**P_TMR0_PreloadHi**, \$14) (W)

	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
W	T0PL15	T0PL14	T0PL13	T0PL12	T0PL11	T0PL10	T0PL9	T0PL8
R	T0CN15	T0CN14	T0CN13	T0CN12	T0CN11	T0CN10	T0CN9	T0CN8
	0	0	0	0	0	0	0	0

Bit [7:0] Write: **T0PL** [15:8] Timer0 Pre-Load Value T0PL (W)
Bit [7:0] Read: **T0CN** [15:8] Timer0 Count Value T0CN (R)

9.2.7 Timer1 Count Low Byte Register (P_TMR1_Count, \$15) (R)

9.2.8 Timer1 Preload Low Byte Register (P_TMR1_Preload, \$15) (W)

	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
W	T1PL7	T1PL6	T1PL5	T1PL4	T1PL3	T1PL2	T1PL1	T1PL0
R	T1CN7	T1CN6	T1CN5	T1CN4	T1CN3	T1CN2	T1CN1	T1CN0
	0	0	0	0	0	0	0	0

Bit [7:0] Write: **T1PL** [7:0] Timer1 Pre-Load Value T1PL [7:0] (W)
Bit [7:0] Read: **T1CN** [7:0] Timer1 Count Value T1CN [7:0] (R)

Note: Writing dataflow: Write high byte first then low byte, i.e., 16-bit timer mode: Writer P_TMR1_PreloadHi first then P_TMR1_Preload.

9.2.9 Timer1 Count High Byte Register (P_TMR1_CountHi, \$16) (R)

9.2.10 Timer1 Preload High Byte Register (P_TMR1_PreloadHi, \$16) (W)

	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
W	T1PL15	T1PL14	T1PL13	T1PL12	T1PL11	T1PL10	T1PL9	T1PL8
R	T1CN15	T1CN14	T1CN13	T1CN12	T1CN11	T1CN10	T1CN9	T1CN8
	0	0	0	0	0	0	0	0

Bit [7:0] Write: **T1PL** [15:8] Timer1 Pre-Load Value T1PL (W)
Bit [7:0] Read: **T1CN** [15:8] Timer1 Count Value T1CN (R)

9.2.11 Timer2_3 Control Register 0 (P_TMR2_3_Ctrl0, \$18) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
-	T3FS2	T3FS1	T3FS0	-	T2FS2	T2FS1	T2FS0
-	R/W	R/W	R/W	-	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit 7 Reserved

Bit [6:4] **T3FS** [2:0]: Timer 3 function configuration bits

111 = 12-bit PWM

110 = 16-bit capture (Width)

101 = 16-bit Compare

100 = 16-bit Timer

011 = 8-bit Capture (Width, Cycle)

010 = 8-bit Compare

001 = 8-bit Timer

000 = Disable

Bit [3]	Reserved
Bit [2:0]	T2FS[2 : 0]: Timer2 Function Configuration bits
	111 = 8-bit PWM
	110 = 16-bit capture (Width)
	101 = 16-bit Compare
	100 = 16-bit Timer
	011 = 8-bit Capture (Width, Cycle)
	010 = 8-bit Compare
	001 = 8-bit Timer
	000 = Disable

9.2.12 Timer2_3 Control Register 1 (P_TMR2_3_Ctrl1, \$19) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
-	T3PSS2	T3PSS1	T3PSS0	-	T2PSS2	T2PSS1	T2PSS0
-	R/W	R/W	R/W	-	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit 7	Reserved
Bit [6:4]	T3PSS[2 : 0]: Timer3 Pre-Scale Configuration bits
	111 = External Event
	110 = $F_{SYS} \div 512$
	101 = $F_{SYS} \div 128$
	100 = $F_{SYS} \div 32$
	011 = $F_{SYS} \div 8$
	010 = $F_{SYS} \div 4$
	001 = $F_{SYS} \div 2$
	000 = F_{SYS}
Bit 3	Reserved
Bit [2:0]	T2PSS[2 : 0]: Timer2 Pre-Scale Configuration bits
	111 = External Event
	110 = $F_{SYS} \div 512$
	101 = $F_{SYS} \div 128$
	100 = $F_{SYS} \div 32$
	011 = $F_{SYS} \div 8$
	010 = $F_{SYS} \div 4$
	001 = $F_{SYS} \div 2$
	000 = F_{SYS}

F_{SYS} : Frequency of Clock Source

9.2.13 Timer2 Count Low Byte Register (P_TMR2_Count, \$1A) (R)

9.2.14 Timer2 Preload Low Byte Register (P_TMR2_Preload, \$1A) (W)

	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
W	T2PL7	T2PL6	T2PL5	T2PL4	T2PL3	T2PL2	T2PL1	T2PL0
R	T2CN7	T2CN6	T2CN5	T2CN4	T2CN3	T2CN2	T2CN1	T2CN0
	0	0	0	0	0	0	0	0

Bit [7:0] Write: **T2PL** [7:0] Timer2 Pre-Load Value T2PL [7:0] (W)

Bit [7:0] Read: **T2CN** [7:0] Timer2 Count Value T2CN [7:0] (R)

Note: Writing dataflow: Write high byte first then low byte, i.e., 16-bit timer mode: Writer

P_TMR2_PreloadHi first then P_TMR2_Preload.

9.2.15 Timer2 Count High Byte Register (P_TMR2_CountHi, \$1B) (R)

9.2.16 Timer2 Preload High Byte Register (P_TMR2_PreloadHi, \$1B) (W)

	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
W	T2PL15	T2PL14	T2PL13	T2PL12	T2PL11	T2PL10	T2PL9	T2PL8
R	T2CN15	T2CN14	T2CN13	T2CN12	T2CN11	T2CN10	T2CN9	T2CN8
	0	0	0	0	0	0	0	0

Bit [7:0] Write: **T2PL** [15:8] Timer2 Pre-Load Value T2PL [15:8] (W)

Bit [7:0] Read: **T2CN** [15:8] Timer2 Count Value T2CN [15:8] (R)

9.2.17 Timer3 Count Low Byte Register (P_TMR3_Count, \$1C) (R)

9.2.18 Timer3 Preload Low Byte Register (P_TMR3_Preload, \$1C) (W)

	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
W	T3PL7	T3PL6	T3PL5	T3PL4	T3PL3	T3PL2	T3PL1	T3PL0
R	T3CN7	T3CN6	T3CN5	T3CN4	T3CN3	T3CN2	T3CN1	T3CN0
	0	0	0	0	0	0	0	0

Bit [7:0] Write: **T3PL** [7:0] Timer3 Pre-Load Value T3PL [7:0] (W)

Bit [7:0] Read: **T3CN** [7:0] Timer3 Count Value T3CN [7:0] (R)

Note: Writing dataflow: Write high byte first then low byte, i.e., 16-bit timer mode: Writer

P_TMR3_PreloadHi first then P_TMR3_Preload.

9.2.19 Timer3 Count High Byte Register (P_TMR3_CountHi, \$1D) (R)
9.2.20 Timer3 Preload High Byte Register (P_TMR3_PreloadHi, \$1D) (W)

	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
W	T3PL15	T3PL14	T3PL13	T3PL12	T3PL11	T3PL10	T3PL9	T3PL8
R	T3CN15	T3CN14	T3CN13	T3CN12	T3CN11	T3CN10	T3CN9	T3CN8
	0	0	0	0	0	0	0	0

Bit [7:0] Write: **T3PL** [15:8] Timer3 Pre-Load Value T3PL [15:8] (W)

Bit [7:0] Read: **T3CN** [15:8] Timer3 Count Value T3CN [15:8] (R)

9.2.21 Timer4_5 Control Register 0 (P_TMR4_5_Ctrl0, \$1F) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
-	T5FS2	T5FS1	T5FS0	-	T4FS2	T4FS1	T4FS0
-	R/W	R/W	R/W	-	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit 7 Reserved

Bit [6:4] **T5FS** [2:0]: Timer 5 function configuration bits

111 = 16-bit PWM

110 = 16-bit capture (Width)

101 = 16-bit Compare

100 = 16-bit Timer

011 = 8-bit Capture (Width, Cycle)

010 = 8-bit Compare

001 = 8-bit Timer

000 = Disable

Bit [3] Reserved

Bit [2:0] **T4FS**[2 : 0]: Timer4 Function Configuration bits

111 = 8-bit PWM

110 = 16-bit capture (Width)

101 = 16-bit Compare

100 = 16-bit Timer

011 = 8-bit Capture (Width, Cycle)

010 = 8-bit Compare

001 = 8-bit Timer

000 = Disable

9.2.22 Timer4_5 Control Register 1 (P_TMR4_5_Ctrl1, \$20) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
-	T5PSS2	T5PSS1	T5PSS0	-	T4PSS2	T4PSS1	T4PSS0
-	R/W	R/W	R/W	-	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit 7 Reserved

Bit [6:4] **T5PSS[2 : 0]**: Timer5 Pre-Scale Configuration bits

111 = External Event

110 = $F_{SYS} \div 512$

101 = $F_{SYS} \div 128$

100 = $F_{SYS} \div 32$

011 = $F_{SYS} \div 8$

010 = $F_{SYS} \div 4$

001 = $F_{SYS} \div 2$

000 = F_{SYS}

Bit 3 Reserved

Bit [2:0] **T4PSS[2 : 0]**: Timer4 Pre-Scale Configuration bits

111 = External Event

110 = $F_{SYS} \div 512$

101 = $F_{SYS} \div 128$

100 = $F_{SYS} \div 32$

011 = $F_{SYS} \div 8$

010 = $F_{SYS} \div 4$

001 = $F_{SYS} \div 2$

000 = F_{SYS}

F_{SYS} : Frequency of Clock Source

9.2.23 Timer4 Count Low Byte Register (P_TMR4_Count, \$21) (R)
9.2.24 Timer4 Preload Low Byte Register (P_TMR4_Preload, \$21) (W)

	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
W	T4PL7	T4PL6	T4PL5	T4PL4	T4PL3	T4PL2	T4PL1	T4PL0
R	T4CN7	T4CN6	T4CN5	T4CN4	T4CN3	T4CN2	T4CN1	T4CN0
	0	0	0	0	0	0	0	0

Bit [7:0] Write: **T4PL** [7:0] Timer4 Pre-Load Value T4PL [7:0] (W)

Bit [7:0] Read: **T4CN** [7:0] Timer4 Count Value T4CN [7:0] (R)

Note: Writing dataflow: Write high byte first then low byte, i.e., 16-bit timer mode: Writer P_TMR4_PreloadHi first then P_TMR4_Preload.

9.2.25 Timer4 Count High Byte Register (P_TMR4_CountHi, \$22) (R)
9.2.26 Timer4 Preload High Byte Register (P_TMR4_PreloadHi, \$22) (W)

	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
W	T4PL15	T4PL14	T4PL13	T4PL12	T4PL11	T4PL10	T4PL9	T4PL8
R	T4CN15	T4CN14	T4CN13	T4CN12	T4CN11	T4CN10	T4CN9	T4CN8
	0	0	0	0	0	0	0	0

Bit [7:0] Write: **T4PL** [15:8] Timer4 Pre-Load Value T2PL [15:8] (W)

Bit [7:0] Read: **T4CN** [15:8] Timer4 Count Value T2CN [15:8] (R)

9.2.27 Timer5 Count Low Byte Register (P_TMR5_Count, \$23) (R)
9.2.28 Timer5 Preload Low Byte Register (P_TMR5_Preload, \$23) (W)

	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
W	T5PL7	T5PL6	T5PL5	T5PL4	T5PL3	T5PL2	T5PL1	T5PL0
R	T5CN7	T5CN6	T5CN5	T5CN4	T5CN3	T5CN2	T5CN1	T5CN0
	0	0	0	0	0	0	0	0

Bit [7:0] Write: **T5PL** [7:0] Timer5 Pre-Load Value T5PL [7:0] (W)

Bit [7:0] Read: **T5CN** [7:0] Timer5 Count Value T5CN [7:0] (R)

Note: Writing dataflow: Write high byte first then low byte, i.e., 16-bit timer mode: Writer P_TMR5_PreloadHi first then P_TMR5_Preload.

9.2.29 Timer5 Count High Byte Register (P_TMR5_CountHi, \$24) (R)
9.2.30 Timer5 Preload High Byte Register (P_TMR5_PreloadHi, \$24) (W)

	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
W	T5PL15	T5PL14	T5PL13	T5PL12	T5PL11	T5PL10	T5PL9	T5PL8
R	T5CN15	T5CN14	T5CN13	T5CN12	T5CN11	T5CN10	T5CN9	T5CN8
	0	0	0	0	0	0	0	0

Bit [7:0] Write: **T5PL** [15:8] Timer5 Pre-Load Value T5PL [15:8] (W)

Bit [7:0] Read: **T5CN** [15:8] Timer5 Count Value T5CN [15:8] (R)

9.3 Operation

SPMC65X family's devices have maximum six 8-bit Timers--- Timer0~5. Timer x (x = 0~5) acts as re-loadable 8/16-bit timer. In timer mode, there are several registers need to be configured such as timer function selection, 8-stage pre-scale counter and 16-bit pre-load register. Timer x's clock source is selected from 8-stage pre-scale and set by "Timer x Pre-scale Selection" register. Since clock source can be selected from 1 to 512 times the number of system clock, Timer x can be operated in a wide range of clock period. The purpose of 8/16-bit

pre-load register is to set initial or reload value into counter when timer starts or counter rolls over.

An 8/16-bit count-up counter is readable register and increases by every internal or external clock input. Take 8-bit timer for example, when the counter runs from 256 to 0, this overflow causes to generate the timer x overflow interrupt if timer x interrupt enable flag is set to '1' and counter register will be reloaded with pre-load value of timer. The counting behavior of Timer x is as Table 9-2. For a step-by-step procedure on how to set up the Timer overflow interrupt, see Section 9.4 in detail.

The Timer x Block Diagram is shown in Figure 9-1:

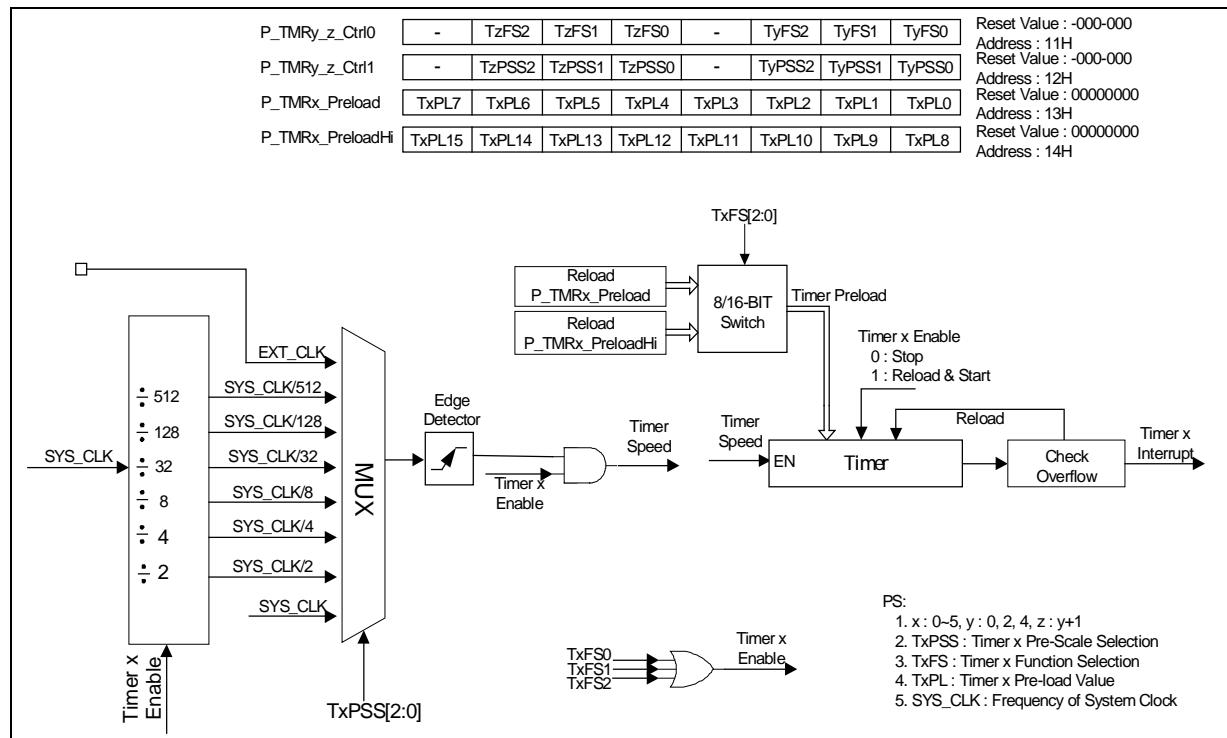


Figure 9-1 Timer X block diagram

8-bit timer DATA (Hex)	Interrupt Period (* T_TMRx)		16-bit timer (Hex)	Interrupt Period (* T_TMRx)	
	Hex	Decimal		Hex	Decimal
\$00	\$100	256	\$0000	\$10000	65536
\$01	\$FF	255	\$0001	\$FFFF	65535
.....
\$FE	\$02	2	\$FFFE	\$0002	2
\$FF	\$01	1	\$FFFF	\$0001	1

Table 9-2 Counting Behavior Of Timer X

9.3.1 8-bit Timer

The alternation functions of timer are decided by setting P_TMRy_z_Ctrl0 ($y=0, 2, 4; z=1, 3, 5$) for using Timer y/z as shown in following description. Each of 8-bit timers has counter register and control register. The counter register increased by every internal or external clock input. When the counter runs from 256 to 0, this overflow causes to generate the Timer x overflow interrupt if Timer x interrupt enable flag is set to '1' and counter register will be reloaded with pre-load value of timer.

The 8-bit Timer interrupt period can be calculated using Equation 9-1.

8-bit timer/counter

If

$$\text{DATA} = \text{P_TMRx_Preload}$$

$$T_{\text{TMRx}} = \text{Timer x Clock Period after pre-scaling}, x = 0\sim 5$$

Then

$$\text{Interrupt period} = \{\$100 - \text{DATA}\} * T_{\text{TMRx}}$$

Equation 9-1 Calculation for 8-bit Timer Interrupt Period

For example, analyze 8-bit timer --- Timer0.

Figure 9-2 shows a simplified block diagram of the 8-bit timer.

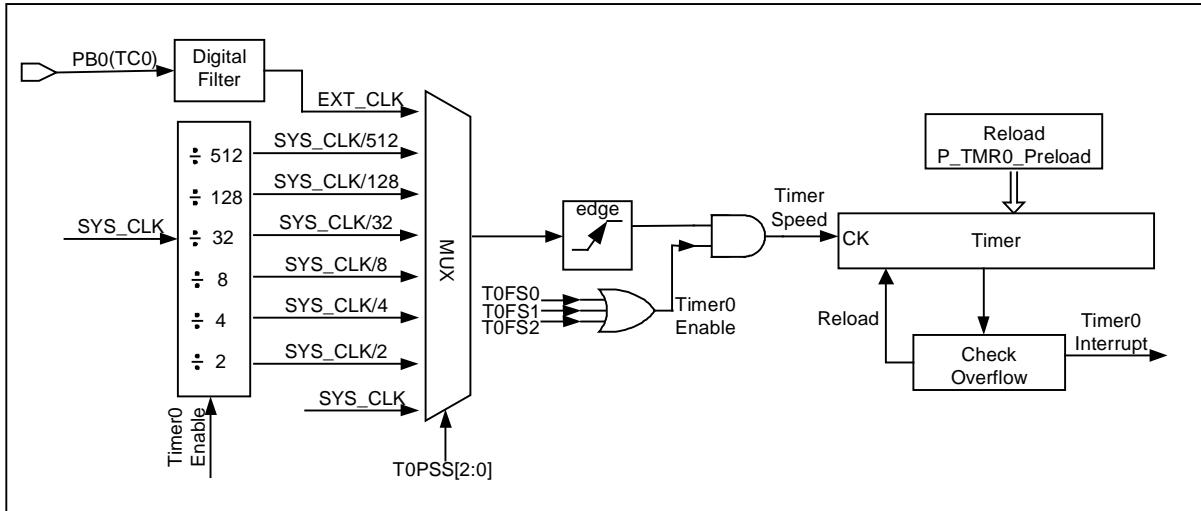


Figure 9-2 8-Bit Timer Block Diagram

Figure 9-3 shows the timer overflow and interrupt position.

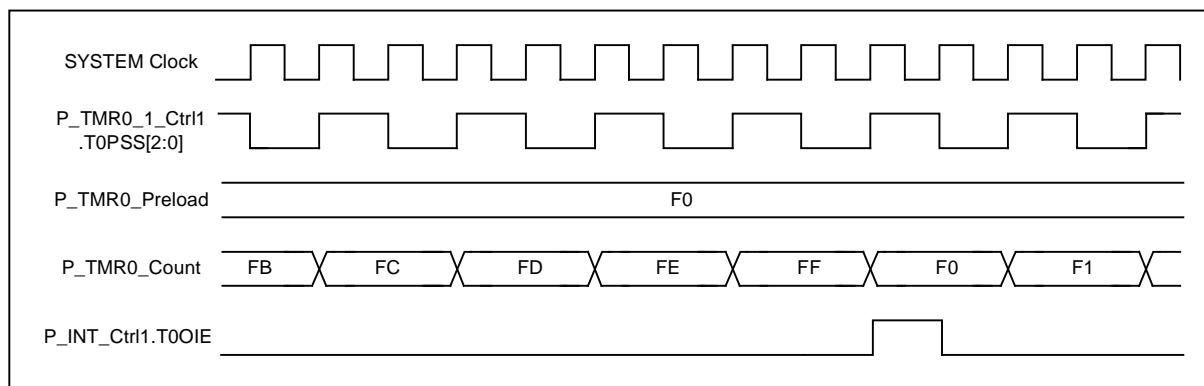


Figure 9-3 8-bit timer overflow

If P_TMR0_1_Ctrl0.T0FS [2:0]=001 and PB0 is set as input mode, then PB0 is the timer0 clock input and timer0 acts as event counter. Timer0 Counter Low Byte Register P_TMR0_Count (\$13) is used to write the assigned pre-load data and to read out the current counting value of Timer0. Once Timer0 overflows, it will set the corresponding flag and will generate interrupt request for service if the interrupt input is enabled, then Timer0 will reload the assigned pre-load data and keep running. Figure 9-4 shows the relation between 8-bit counting values and interrupt occurrence. Detailed registers setting are described in previous pages.

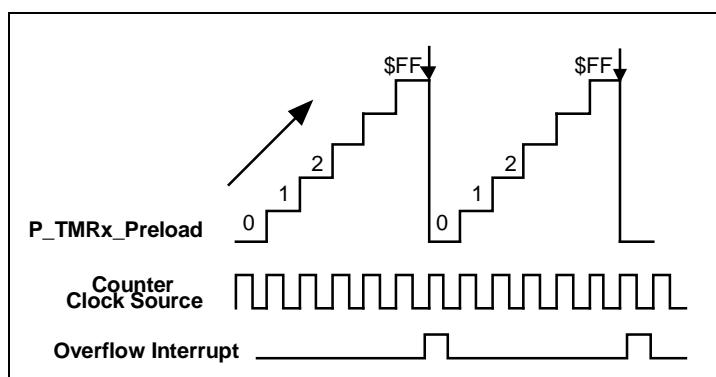


Figure 9-4 The Relation Between 8-bit Counting Values And Interrupt

[Example 9-1]: Set Timer0 as 8-bit timer operation.

ld a	#6	; Before starting timer, set Timer0 counter initial value first.
sta	P_TMR0_Preload	;
ld a	#C_T0FCS_Div_32	; Set Timer0 clock source is Fsys/32
sta	P_TMR0_1_Ctrl1	
ld a	#C_T08B_Timer	; Set Timer0 is 8-bit timer
sta	P_TMR0_1_Ctrl0	
ld a	#6	; Set Timer0 preload counter= 256-6= 250
sta	P_TMR0_Preload	; Fsys(8MHz)/32/250= 1KHz(1ms)

9.3.2 16-bit Timer

In SPMC65X family, if a chip equips with 8-bit CPU and data bus width is 8-bit, the user can't access 16-bit data simultaneously. In order to overcome this limitation in 16-bit timer/counter mode, the Timer x MSB (Most Significant Byte) data registers are designed with extra read/write buffer, which is shown below. User must read the LSB Byte first and then the MSB byte that is buffered automatically. This buffered value remains unchanged until the 16-bit read sequence is completed. Moreover, user must write the MSB byte first that will be buffered automatically and then the LSB byte. This buffered value remains unchanged until the 16-bit write sequence is completed and the buffered MSB byte will download into the timer0 with LSB byte simultaneously.

The relationship between data register and interrupt period can be calculated from the Equation 9-2

16-bit timer/counter

If

$$\text{DATA} = \{\text{P_TMRx_PreloadHi}, \text{P_TMRx_Preload}\}$$

$$\text{T_TMRx} = \text{Timer x Clock Period after pre-scaling, } x = 0\sim 5$$

Then

$$\text{Interrupt period} = \{\$10000 - \text{DATA}\} * \text{T_TMRx}$$

Equation 9-2 Calculation for 16-bit Timer Interrupt Period

For example, analyze 16-bit timer --- Timer0.

Detailed timer block diagram is shown in Figure 9-5.

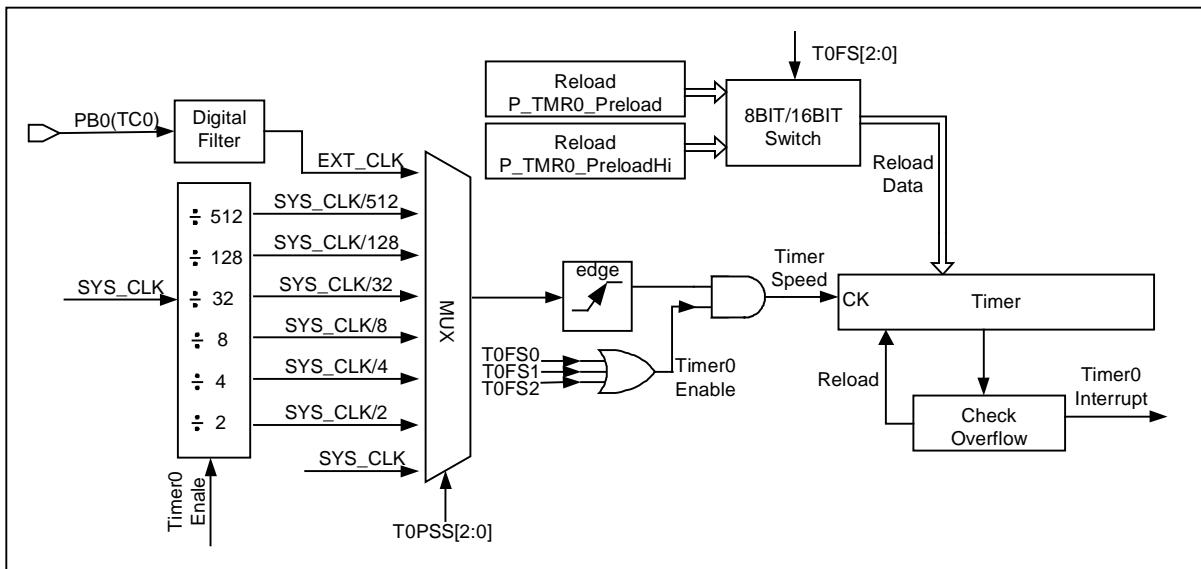


Figure 9-5 16-Bit Timer Block Diagram

Figure 9-6 shows the timer overflow and interrupt position.

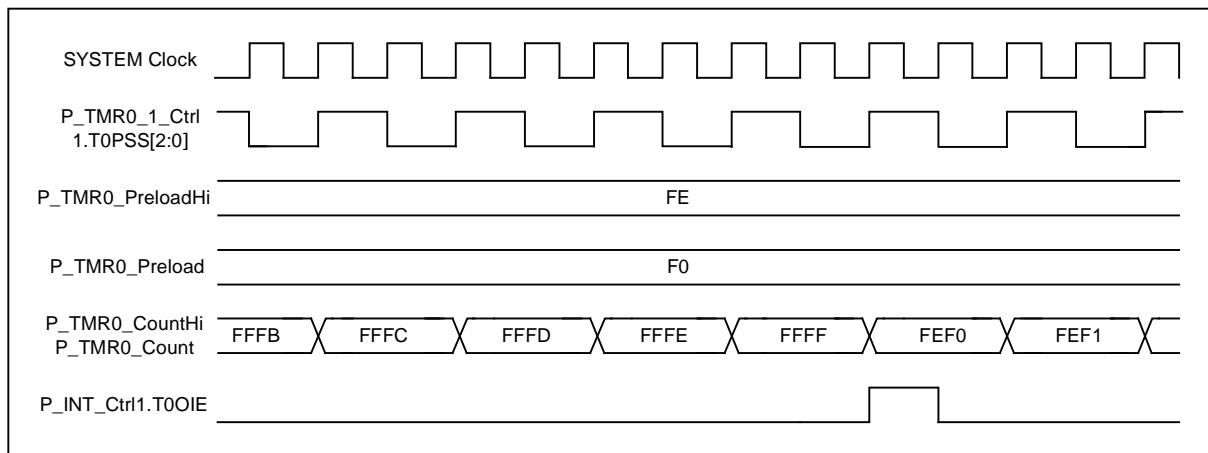


Figure 9-6 16-bit timer overflow

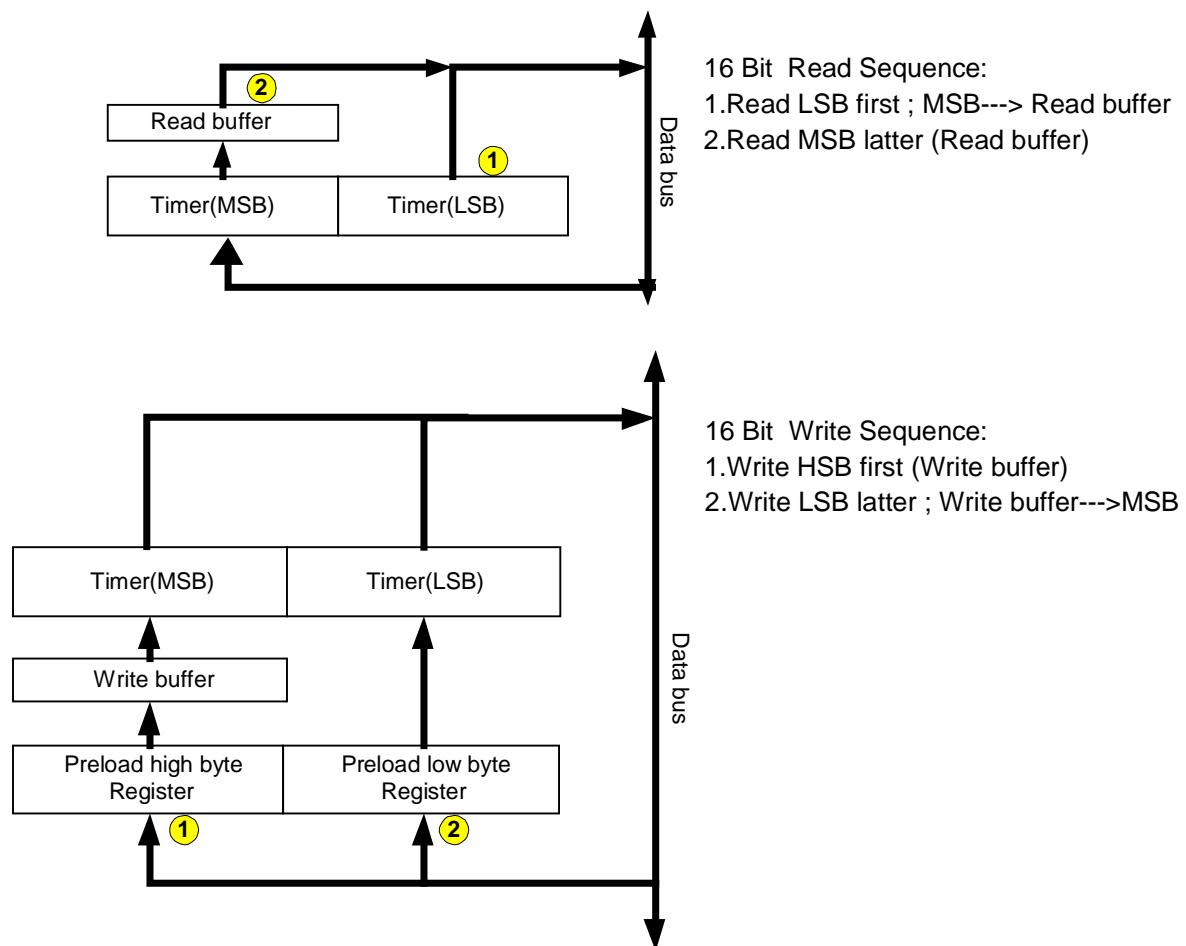


Figure 9-7 16-Bit Timer access method diagram

If P_TMR0_1_Ctrl0.T0FS [2:0]=100 and PB0 is set as input mode, then PB0 is the timer0 clock input and timer0 acts as event counter. Timer0 Counter Low Byte Register P_TMR0_Count (\$13) and Timer0 Counter High Byte Register P_TMR0_CountHi (\$14) are used to write the assigned pre-load data and to read out the current counting value of Timer0. Once Timer0 overflows, it will set the corresponding flag and will generate interrupt request for service if the interrupt input is enabled, then Timer0 will reload the assigned pre-load data and keep running. Figure 9-8 shows the relation between 16-bit counting values and interrupt occurrence. Detailed registers setting are described in previous pages.

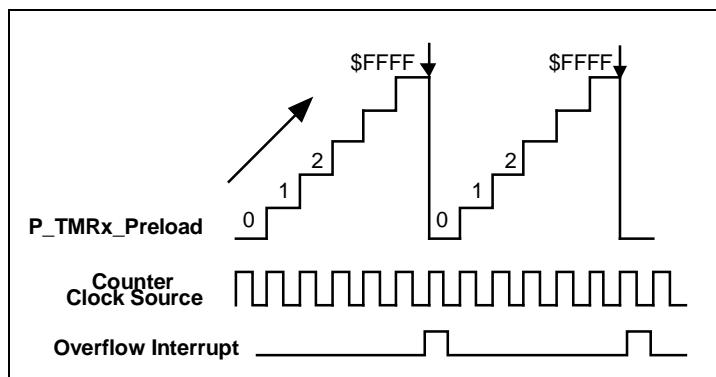


Figure 9-8 The Relation Between 16-bit Counting Values And Interrupt

[Example 9-2]: Set Timer0 as 16-bit timer operation

```

    lda    #253           ; Before starting timer, set Timer0 counter initial value first.
    sta    P_TMR0_PreloadHi ; Set high byte initial value first
    lda    #143
    sta    P_TMR0_Preload   ; Set low byte initial value latter
    lda    #C_T0FCS_Div_128 ; Set Timer0 clock source is Fsys/128
    sta    P_TMR0_1_Ctrl1   ; Set Timer0 is 16-bit timer
    lda    #C_T016B_Timer
    sta    P_TMR0_1_Ctrl0   ; 1'st set Timer0 preload high byte counter= 2x 256= 512
    lda    #253             ; 2'nd set Timer01 preload low byte counter= 256-143= 113
    sta    P_TMR0_PreloadHi
    lda    #143
    sta    P_TMR0_Preload   ; Fsys(8Mhz)/128/625= 100Hz(10ms)
  
```

9.4 Timer Interrupt

Interrupts can come from many sources. There are five interrupt sources related with Timer.

- | Timer0 Overflow Interrupt
- | Timer1 Overflow Interrupt

- | Timer2 Overflow Interrupt
- | Timer3 Overflow Interrupt
- | Timer4 Overflow Interrupt
- | Timer5 Overflow Interrupt

There are several steps for enabling timer overflow interrupt.

1. Interrupt Disable flag (I-flag) is set to “0” by using “SEI ” instruction.
2. Decided 8-bit or 16-bit timer and then selected a suitable timer for use.
3. Configured some registers related timer x(x = 0~5) you selected.
4. Enable corresponding timer x interrupt flag in the P_INT_Ctrl1(\$0F) register.
5. Using ‘CLI’ instruction to enable all interrupt function.
6. Now the timer overflow interrupt will be accepted.

[Example 9-3]: Set Timer0 as 8-bit timer operation and generate 1msec period interrupt by Timer0 overflow.

```

lde      #6                      ; Before starting timer, set Timer0 counter initial value
sta      P_TMR0_Preload          ; Set low byte initial value latter
lde      #C_T0FCS_Div_32         ; Set Timer0 clock source is Fsys/32
sta      P_TMR0_1_Ctrl1          ; Set Timer0 is 8-bit timer
lde      #C_T08B_Timer           ; Set Timer0 preload counter= 256-6= 250
sta      P_TMR0_Preload          ; Fsys(8MHz)/32/250= 1KHz(1ms)
lde      #$FF                     ; clear INT request flag
sta      P_INT_Flag1            ; enable INT
set      P_INT_Ctrl1, CB_INT_T0OIE ; enable Timer0 overflow
cli

```

[Example 9-4]: Set Timer1 as 16-bit timer operation and generate 10msec period interrupt by Timer1 overflow.

```

lde      #253                    ; Before starting timer, set Timer0 counter initial value.
sta      P_TMR1_PreloadHi        ; Set high byte initial value first
lde      #143                    ; Set Timer1 clock source is Fsys/128
sta      P_TMR0_1_Ctrl1          ; Set Timer1 is 16-bit timer
lde      P_TMR1_Preload          ; Set low byte initial value latter
lde      #C_T116B_Timer           ; 1'st set Timer1 preload high byte counter= 2x 256= 512
sta      P_TMR1_PreloadHi        ; 2'nd set Timer1 preload low byte counter= 256-143= 113
lde      #143                    ; Fsys(8Mhz)/128/625= 100Hz(10ms)
sta      P_TMR1_Preload          ; clear INT request flag
sta      P_INT_Flag1            ; enable INT
set      P_INT_Ctrl1, CB_INT_T1OIE ; enable Timer1 overflow

```

cli ; enable INT

9.5 Design Tips

[Example 9-5] Shows how to initialize Timer 1 as 16-bit timer based on SPMC65P2404A. Please refer to an example of Fortis IDE.

- Use Timer1 16-bit timer
- Set Timer1 clock source as Fcs/128
- Generate 262msec period interrupt by Timer1 overflow

```

.SYNTAX 6502           ; process standard 6502 addressing syntax
.LINKLIST                      ; generate linklist information
.SYMBOLS                        ; generate symbolic debug information

.INCLUDE    spmc65p2404a.inc

.PAGE0                         ; define values in the range from 00h to FFh
;
.DATA                          ; define data storage section
;
.CODE

; **** Power on Reset Process - Main Program ****
;*
;*
;* Power on Reset Process - Main Program
;*
;*
; ****

V_Reset:
  sei          ; Disable interrupt
  ldx #C_STACK_BOTTOM ; Initial stack pointer at $00ff
  txa          ; Transfer to stack point
;
  lda #$FF      ; Clear Reset flag
  sta P_SYS_Ctrl
  sta P_SYS_Ctrl
;
  jsr F_InitIOPort ; initial GPIO port
;
  lda #$C0      ; ($FFFF - $C000) x 128 / 8M ~= 262 ms
  sta P_TMR1_PreloadHi
  lda #$00
  sta P_TMR1_Preload
;
  lda #C_T1FCS_Div_128 ; Set Timer1 clock source is Fsys/128
;
```

```

sta      P_TMR0_1_Ctrl1
lda      #C_T116B_Timer           ; Set Timer1 is 16-bit timer
sta      P_TMR0_1_Ctrl0
;
lda      #$FF                   ; clear INT request flag
sta      P_INT_Flag0
sta      P_INT_Flag1
set      P_INT_Ctrl1, CB_INT_T1OIE ; enable Timer1 overflow
cli      ; enable INT
;
L_Main:
nop
jmp      L_Main
;
; *****
; *          *
; *      IRQ Interrupt Service Routine      *
; *          *
; *****
;
V_IRQ:
pha      ; push A register
txa      ; transfer X to A
pha      ; push A register (ie. push X)
;
; Timer 1 overflow interrupt ?
;
lda      P_INT_Flag1
and      #C_INT_T1OIF
beq      L_exit_irq
;
set      P_INT_Flag1, CB_INT_T1OIF
;
lda      P_IOA_Data
eor      #$FF
sta      P_IOA_Data           ; toggle P_IOA_Data
;
L_exit_irq:
pla      ; pop A register
tax      ; transfer A to X
pla      ; pop A register (ie. pop X)
;
rti
;
; *****

```

```

;*                                *
;*      Interrupt Vector Table    *
;*                                *
;*                                *
;*****                         *****
VECTOR:      .SECTION
    DW      V_NMI           ; may download program emulated either
    DW      V_Reset          ; in internal memory or external memory
    DW      V_IRQ            ; dw define two bytes interrupt vector
;
;*****                         *****
;*                                *
;*      End Of Interrupt Vector Table   *
;*                                *
;*****                         *****
;
.END          ; end of program

```

9.6 Related Application Notes and Libraries

This section lists some application notes and libraries about various usage of SPMC65X family. The application notes may help user develop quickly and offer the concepts about the usage of this topic. Sunplus also supports some powerful libraries about general operation, mathematics, communication, etc. The purpose of library is to help user speed up development. The current application notes and libraries related to Timer are listed below.

Application Note:

Title	Application Note Series Number
LED (x 4) + momentary key (4 x 4) scan	AN_O0306.DOC
LED (x 4) + switch key (x 4) + momentary key (4 x 4) scan	AN_O0307.DOC
7-segment LED display	AN_O0308.DOC
I/O simulate LCD display	AN_O0309.DOC
I/O simulate SPI communication	AN_O0310.DOC
I/O simulate I2C communication	AN_O0311.DOC
I/O simulate UART communication	AN_O0312.DOC
I/O access EEPROM (93c46)	AN_O0313.DOC
I/O access EEPROM (24c01)	AN_O0314.DOC
I/O simulate A/D convert with RC	AN_O0316.DOC
I/O simulate D/A convert with PWM	AN_O0317.DOC
I/O simulate SIN wave output (R2R)	AN_O0318.DOC
I/O simulate melody output	AN_O0319.DOC

Using the Timer0 CCP module	AN_O0331.DOC
Using the Timer1 CCP module	AN_O0332.DOC
Using the Timer2 CCP module	AN_O0333.DOC
Using the Timer3 CCP module	AN_O0334.DOC
Using the Timer4 CCP module	AN_O0335.DOC
Using the Timer5 CCP module	AN_O0336.DOC

Library:

Title	Application Note Series Number
SPMC65X series software basics manual	AN_O0100.DOC
SPMC65X series software advances 1 manual (data processing operation)	AN_O0101.DOC
SPMC65X series software advances 2 manual (mathematical operation)	AN_O0102.DOC

9.7 Revision History

Revision	Date	Remark
V0.1	03/31/2005	First edition

10 Capture

10.1 Description

In SPMC65X family, as many as 6 channels of Timer are equipped and each one can be set to operate as 8-bit/16-bit Capture function. In capture mode, it supports two transition modes such as Rising edge and Falling edge. In 8-bit Capture mode, the pulse width and cycle can be measured. But in 16-bit Capture mode, only pulse width can be measured except for Timer5. These functions can be easily configured with corresponding control registers setting. Its function summary is shown as below:

	Timer/Event Counter		Capture		Compare		PWM
	8 bit	16 bit	8 bit	16 bit	8 bit	16 bit	
Timer 0	YES	YES	Width/Cycle	Width only	YES	YES	8 bit
Timer 1	YES	YES	Width/Cycle	Width only	YES	YES	12 bit
Timer 2	YES	YES	Width/Cycle	Width only	YES	YES	8 bit
Timer 3	YES	YES	Width/Cycle	Width only	YES	YES	12 bit
Timer 4	YES	YES	Width/Cycle	Width only	YES	YES	8 bit
Timer 5	YES	YES	Width/Cycle	Width/Cycle	YES	YES	16 bit

Table 10-1 Summary of Capture function for SPMC65X family

The timers have three types depending on different features:

1. Type I: (Timer0, Timer2 & Timer4)
 - 8 or 16-bit timer/counter
 - **8 or 16-bit capture mode (8-bit with width/cycle measurement, 16-bit with width measurement)**
 - 8 or 16-bit compare mode
 - 8-bit PWM mode
2. Type II: (Timer1 & Timer3)
 - 8 or 16-bit timer/counter
 - **8 or 16-bit capture mode (8-bit with width/cycle measurement, 16-bit with width measurement)**
 - 8 or 16-bit compare mode
 - 12-bit PWM mode
3. Type III: (Timer5)
 - 8 or 16-bit timer/counter
 - **8 or 16-bit capture mode (8 or 16-bit with width/cycle measurement)**
 - 8 or 16-bit compare mode

n 16-bit PWM mode

10.2 Control Register

10.2.1 Timer0_1 Control Register 0 (P_TMR0_1_Ctrl0, \$11) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
-	T1FC2	T1FC1	T1FC0	-	T0FC2	T0FC1	T0FC0
-	R/W	R/W	R/W	-	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit 7 Reserved

Bit [6:4] **T1FC[2 : 0]**: Timer1 function configuration bits

111 = 12-bit PWM

110 = 16-bit capture (Width)

101 = 16-bit Compare

100 = 16-bit Timer

011 = 8-bit Capture (Width, Cycle)

010 = 8-bit Compare

001 = 8-bit Timer

000 = Disable

Bit 3 Reserved

Bit [2:0] **T0FC[2 : 0]**: Timer0 Function Configuration bits

111 = 8-bit PWM

110 = 16-bit capture (Width)

101 = 16-bit Compare

100 = 16-bit Timer

011 = 8-bit Capture (Width, Cycle)

010 = 8-bit Compare

001 = 8-bit Timer

000 = Disable

10.2.2 Timer0_1 Control Register 1 (P_TMR0_1_Ctrl1, \$12) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
-	T1PSS2	T1PSS1	T1PSS0	-	T0PSS2	T0PSS1	T0PSS0
-	R/W	R/W	R/W	-	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit 7 Reserved

Bit [6:4] **T1PSS[2 : 0]**: Timer1 Pre-Scale Configuration bits

111 = External Event

$$110 = F_{SYS} \div 512$$

$$101 = F_{SYS} \div 128$$

$$100 = F_{SYS} \div 32$$

$$011 = F_{SYS} \div 8$$

$$010 = F_{SYS} \div 4$$

$$001 = F_{SYS} \div 2$$

$$000 = F_{SYS}$$

Bit 3 Reserved

Bit [2:0] **T0PSS[2 : 0]**: Timer0 Pre-Scale Configuration bits

$$111 = \text{External Event}$$

$$110 = F_{SYS} \div 512$$

$$101 = F_{SYS} \div 128$$

$$100 = F_{SYS} \div 32$$

$$011 = F_{SYS} \div 8$$

$$010 = F_{SYS} \div 4$$

$$001 = F_{SYS} \div 2$$

$$000 = F_{SYS}$$

F_{SYS} : Frequency of Clock Source

10.2.3 Timer0 Capture Width Low Byte Register (**P_TMR0_Cap**, \$13) (R/W)

	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
W	T0PL7	T0PL6	T0PL5	T0PL4	T0PL3	T0PL2	T0PL1	T0PL0
R	T0CW7	T0CW6	T0CW5	T0CW4	T0CW3	T0CW2	T0CW1	T0CW0
	0	0	0	0	0	0	0	0

Bit [7:0] Write: **T0PL** [7:0] Timer0 Pre-Load Value T0PL [7:0] (W)

Bit [7:0] Read: **T0CW** [7:0] Timer0 Capture Width Value T0CW [7:0] (R)

Note: Writing dataflow: Write high byte first then low byte, i.e., 16-bit timer mode: Write P_TMR0_CapHi first then P_TMR0_Cap

10.2.4 Timer0 Capture Width High Byte Register (**P_TMR0_CapHi**, \$14) (R/W)

	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
W	T0PL15	T0PL14	T0PL13	T0PL12	T0PL11	T0PL10	T0PL9	T0PL8
R	T0CW15	T0CW14	T0CW13	T0CW12	T0CW11	T0CW10	T0CW9	T0CW8
	0	0	0	0	0	0	0	0

Bit [7:0] Write: **T0PL** [15:8] Timer0 Pre-Load Value T0PL [15:8] (W)

Bit [7:0] Read: **T0CW** [15:8] Timer0 Capture Width Value T0CW [15:8] (R)

10.2.5 Timer0 Capture Cycle Byte Register (P_TMR0_CapCycle8, \$14) (R/W)

	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
W	T0PL15	T0PL14	T0PL13	T0PL12	T0PL11	T0PL10	T0PL9	T0PL8
R	T0CC7	T0CC6	T0CC5	T0CC4	T0CC3	T0CC2	T0CC1	T0CC0
	0	0	0	0	0	0	0	0

 Bit [7:0] Write: **T0PL** [15:8] Timer0 Pre-Load Value T0PL [15:8] (W)

 Bit [7:0] Read: **T0CC** [7:0] Timer0 Capture Cycle Value T0CC [7:0] (R)

10.2.6 Timer1 Capture Width Low Byte Register (P_TMR1_Cap, \$15) (R/W)

	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
W	T1PL7	T1PL6	T1PL5	T1PL4	T1PL3	T1PL2	T1PL1	T1PL0
R	T1CW7	T1CW6	T1CW5	T1CW4	T1CW3	T1CW2	T1CW1	T1CW0
	0	0	0	0	0	0	0	0

 Bit [7:0] Write: **T1PL** [15:8] Timer1 Pre-Load Value T1PL [7:0] (W)

 Bit [7:0] Read: **T1CW** [7:0] Timer1 Capture Width Value T0CW [7:0] (R)

Note: Writing dataflow: Write high byte first then low byte, i.e., 16-bit timer mode: Write P_TMR1_CapHi first then P_TMR1_Cap

10.2.7 Timer1 Capture Width High Byte Register (P_TMR1_CapHi, \$16) (R/W)

	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
W	T1PL15	T1PL14	T1PL13	T1PL12	T1PL11	T1PL10	T1PL9	T1PL8
R	T1CW15	T1CW14	T1CW13	T1CW12	T1CW11	T1CW10	T1CW9	T1CW8
	0	0	0	0	0	0	0	0

 Bit [7:0] Write: **T1PL** [15:8] Timer1 Pre-Load Value T1PL [15:8] (W)

 Bit [7:0] Read: **T1CW** [15:8] Timer1 Capture Width Value T1CW [15:8] (R)

10.2.8 Timer1 Capture Cycle Byte Register (P_TMR1_CapCycle8, \$16) (R/W)

	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
W	T1PL15	T1PL14	T1PL13	T1PL12	T1PL11	T1PL10	T1PL9	T1PL8
R	T1CC7	T1CC6	T1CC5	T1CC4	T1CC3	T1CC2	T1CC1	T1CC0
	0	0	0	0	0	0	0	0

 Bit [7:0] Write: **T1PL** [15:8] Timer1 Pre-Load Value T1PL [15:8] (W)

 Bit [7:0] Read: **T1CC** [7:0] Timer1 Capture Cycle Value T1CC [7:0] (R)

10.2.9 Timer2_3 Control Register 0 (P_TMR2_3_Ctrl0, \$18) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
-	T3FC2	T3FC1	T3FC0	-	T2FC2	T2FC1	T2FC0
-	R/W	R/W	R/W	-	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit 7 Reserved

Bit [6:4] **T3FC[2 : 0]**: Timer3 Pre-Scale Configuration bits

111 = 12-bit PWM

110 = 16-bit capture (Width)

101 = 16-bit Compare

100 = 16-bit Timer

011 = 8-bit Capture (Width, Cycle)

010 = 8-bit Compare

001 = 8-bit Timer

000 = Disable

Bit 3 Reserved

Bit [2:0] **T2FC[2 : 0]**: Timer2 Pre-Scale Configuration bits

111 = 8-bit PWM

110 = 16-bit capture (Width)

101 = 16-bit Compare

100 = 16-bit Timer

011 = 8-bit Capture (Width, Cycle)

010 = 8-bit Compare

001 = 8-bit Timer

000 = Disable

F_{SYS} : Frequency of Clock Source

10.2.10 Timer2_3 Control Register 1 (P_TMR2_3_Ctrl1, \$19) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
-	T3PSS2	T3PSS1	T3PSS0	-	T2PSS2	T2PSS1	T2PSS0
-	R/W	R/W	R/W	-	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit 7 Reserved

Bit [6:4] **T3PSS[2 : 0]**: Timer3 Pre-Scale Configuration bits

111 = External Event

110 = F_{SYS} ÷ 512

101 = F_{SYS} ÷ 128

$$100 = F_{SYS} \div 32$$

$$011 = F_{SYS} \div 8$$

$$010 = F_{SYS} \div 4$$

$$001 = F_{SYS} \div 2$$

$$000 = F_{SYS}$$

Bit 3 Reserved

Bit [2:0] **T2PSS[2 : 0]**: Timer2 Pre-Scale Configuration bits

$$111 = \text{External Event}$$

$$110 = F_{SYS} \div 512$$

$$101 = F_{SYS} \div 128$$

$$100 = F_{SYS} \div 32$$

$$011 = F_{SYS} \div 8$$

$$010 = F_{SYS} \div 4$$

$$001 = F_{SYS} \div 2$$

$$000 = F_{SYS}$$

F_{SYS} : Frequency of Clock Source

10.2.11 Timer2 Capture Width Low Byte Register (**P_TMR2_Cap, \$1A**) (R/W)

	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
W	T2PL7	T2PL6	T2PL5	T2PL4	T2PL3	T2PL2	T2PL1	T2PL0
R	T2CW7	T2CW6	T2CW5	T2CW4	T2CW3	T2CW2	T2CW1	T2CW0
	0	0	0	0	0	0	0	0

Bit [7:0] Write: **T2PL** [15:8] Timer2 Pre-Load Value T2PL [15:8] (W)

Bit [7:0] Read: **T2CW** [7:0] Timer2 Capture Width Value T2CW [7:0] (R)

Note: Writing dataflow: Write high byte first then low byte, i.e., 16-bit timer mode: Write P_TMR2_CapHi first then P_TMR2_Cap

10.2.12 Timer2 Capture Width High Byte Register (**P_TMR2_CapHi, \$1B**) (R/W)

	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
W	T2PL15	T2PL14	T2PL13	T2PL12	T2PL11	T2PL10	T2PL9	T2PL8
R	T2CW15	T2CW14	T2CW13	T2CW12	T2CW11	T2CW10	T2CW9	T2CW8
	0	0	0	0	0	0	0	0

Bit [7:0] Write: **T2PL** [15:8] Timer2 Pre-Load Value T2PL [15:8] (W)

Bit [7:0] Read: **T2CW** [15:8] Timer2 Capture Width Value T2CW [15:8] (R)

10.2.13 Timer2 Capture Cycle Byte Register (P_TMR2_CapCycle8, \$1B) (R/W)

	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
W	T2PL15	T2PL14	T2PL13	T2PL12	T2PL11	T2PL10	T2PL9	T2PL8
R	T2CC7	T2CC6	T2CC5	T2CC4	T2CC3	T2CC2	T2CC1	T2CC0
	0	0	0	0	0	0	0	0

 Bit [7:0] Write: **T2PL** [15:8] Timer2 Pre-Load Value T2PL [15:8] (W)

 Bit [7:0] Read: **T2CC** [7:0] Timer2 Capture Cycle Value T2CC [7:0] (R)

10.2.14 Timer3 Capture Width Low Byte Register (P_TMR3_Cap, \$1C) (R/W)

	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
W	T3PL7	T3PL6	T3PL5	T3PL4	T3PL3	T3PL2	T3PL1	T3PL0
R	T3CW7	T3CW6	T3CW5	T3CW4	T3CW3	T3CW2	T3CW1	T3CW0
	0	0	0	0	0	0	0	0

 Bit [7:0] Write: **T3PL** [15:8] Timer3 Pre-Load Value T3PL [7:0] (W)

 Bit [7:0] Read: **T3CW** [7:0] Timer3 Capture Width Value T3CW [7:0] (R)

Note: Writing dataflow: Write high byte first then low byte, i.e., 16-bit timer mode: Write P_TMR3_CapHi first then P_TMR3_Cap

10.2.15 Timer3 Capture Width High Byte Register (P_TMR3_CapHi, \$1D) (R/W)

	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
W	T3PL15	T3PL14	T3PL13	T3PL12	T3PL11	T3PL10	T3PL9	T3PL8
R	T3CW15	T3CW14	T3CW13	T3CW12	T3CW11	T3CW10	T3CW9	T3CW8
	0	0	0	0	0	0	0	0

 Bit [7:0] Write: **T3PL** [15:8] Timer3 Pre-Load Value T3PL [15:8] (W)

 Bit [7:0] Read: **T3CW** [15:8] Timer3 Capture Width Value T3CW [15:8] (R)

10.2.16 Timer3 Capture Cycle Byte Register (P_TMR3_CapCycle8, \$1D) (R/W)

	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
W	T3PL15	T3PL14	T3PL13	T3PL12	T3PL11	T3PL10	T3PL9	T3PL8
R	T3CC7	T3CC6	T3CC5	T3CC4	T3CC3	T3CC2	T3CC1	T3CC0
	0	0	0	0	0	0	0	0

 Bit [7:0] Write: **T3PL** [15:8] Timer3 Pre-Load Value T3PL [15:8] (W)

 Bit [7:0] Read: **T3CC** [7:0] Timer3 Capture Cycle Value T3CC [7:0] (R)

10.2.17 Timer4_5 Control Register 0 (P_TMR4_5_Ctrl0, \$1F) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
-	T5FC2	T5FC1	T5FC0	-	T4FC2	T4FC1	T4FC0
-	R/W	R/W	R/W	-	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit 7 Reserved

Bit [6:4] **T5FC[2:0]**: Timer 5 function configuration bits

111 = 16-bit PWM

110 = 16-bit capture (Width, Cycle)

101 = 16-bit Compare

100 = 16-bit Timer

011 = 8-bit Capture (Width, Cycle)

010 = 8-bit Compare

001 = 8-bit Timer

000 = Disable

Bit 3 Reserved

Bit [2:0] **T4FC[2 : 0]**: Timer 4 Function Configuration bits

111 = 8-bit PWM

110 = 16-bit capture (Width)

101 = 16-bit Compare

100 = 16-bit Timer

011 = 8-bit Capture (Width, Cycle)

010 = 8-bit Compare

001 = 8-bit Timer

000 = Disable

F_{SYS} : Frequency of Clock Source

10.2.18 Timer4_5 Control Register 1 (P_TMR4_5_Ctrl1, \$20) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
-	T5PSS2	T5PSS1	T5PSS0	-	T4PSS2	T4PSS1	T4PSS0
-	R/W	R/W	R/W	-	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit 7 Reserved

Bit [6:4] **T5PSS[2 : 0]**: Timer5 Pre-Scale Configuration bits

111 = External Event

110 = F_{SYS} ÷ 512

101 = F_{SYS} ÷ 128

100 = F_{SYS} ÷ 32

011 = $F_{SYS} \div 8$

010 = $F_{SYS} \div 4$

001 = $F_{SYS} \div 2$

000 = F_{SYS}

Bit 3 Reserved

Bit [2:0] **T4PSS[2 : 0]**: Timer4 Pre-Scale Configuration bits

111 = External Event

110 = $F_{SYS} \div 512$

101 = $F_{SYS} \div 128$

100 = $F_{SYS} \div 32$

011 = $F_{SYS} \div 8$

010 = $F_{SYS} \div 4$

001 = $F_{SYS} \div 2$

000 = F_{SYS}

F_{SYS} : Frequency of Clock Source

10.2.19 Timer4 Capture Width Low Byte Register (**P_TMR4_Cap, \$21**) (R/W)

	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
W	T4PL7	T4PL6	T4PL5	T4PL4	T4PL3	T4PL2	T4PL1	T4PL0
R	T4CW7	T4CW6	T4CW5	T4CW4	T4CW3	T4CW2	T4CW1	T4CW0
	0	0	0	0	0	0	0	0

Bit [7:0] Write: **T4PL** [15:8] Timer4 Pre-Load Value T4PL [15:8] (W)

Bit [7:0] Read: **T4CW** [7:0] Timer4 Capture Width Value T4CW [7:0] (R)

Note: Writing dataflow: Write high byte first then low byte, i.e., 16-bit timer mode: Write

P_TMR4_CapHi first then **P_TMR4_Cap**

10.2.20 Timer4 Capture Width High Byte Register (**P_TMR4_CapHi, \$22**) (R/W)

	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
W	T4PL15	T4PL14	T4PL13	T4PL12	T4PL11	T4PL10	T4PL9	T4PL8
R	T4CW15	T4CW14	T4CW13	T4CW12	T4CW11	T4CW10	T4CW9	T4CW8
	0	0	0	0	0	0	0	0

Bit [7:0] Write: **T4PL** [15:8] Timer4 Pre-Load Value T4PL [15:8] (W)

Bit [7:0] Read: **T4CW** [15:8] Timer4 Capture Width Value T4CW [15:8] (R)

10.2.21 Timer4 Capture Cycle Byte Register (P_TMR4_CapCycle8, \$22) (R/W)

	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
W	T4PL15	T4PL14	T4PL13	T4PL12	T4PL11	T4PL10	T4PL9	T4PL8
R	T4CC7	T4CC6	T4CC5	T4CC4	T4CC3	T4CC2	T4CC1	T4CC0
	0	0	0	0	0	0	0	0

Bit [7:0] Write: **T4PL** [15:8] Timer4 Pre-Load Value T4PL [15:8] (W)

Bit [7:0] Read: **T4CC** [7:0] Timer4Capture Cycle Value T4CC [7:0] (R)

10.2.22 Timer5 Capture Width Low Byte Register (P_TMR5_Cap, \$23) (R/W)

	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
W	T5PL7	T5PL6	T5PL5	T5PL4	T5PL3	T5PL2	T5PL1	T5PL0
R	T5CW7	T5CW6	T5CW5	T5CW4	T5CW3	T5CW2	T5CW1	T5CW0
	0	0	0	0	0	0	0	0

Bit [7:0] Write: **T5PL** [15:8] Timer5 Pre-Load Value T5PL [15:8] (W)

Bit [7:0] Read: **T5CW** [7:0] Timer5Capture Width Value T5WC [7:0] (R)

Note: Writing dataflow: Write high byte first then low byte, i.e., 16-bit timer mode: Write P_TMR5_CapHi first then P_TMR5_Cap

10.2.23 Timer5 Capture Width High Byte Register (P_TMR5_CapHi, \$24) (R/W)

	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
W	T5PL15	T5PL14	T5PL13	T5PL12	T5PL11	T5PL10	T5PL9	T5PL8
R	T5CW15	T5CW14	T5CW13	T5CW12	T5CW11	T5CW10	T5CW9	T5CW8
	0	0	0	0	0	0	0	0

Bit [7:0] Write: **T5PL** [15:8] Timer5 Pre-Load Value T5PL [15:8] (W)

Bit [7:0] Read: **T5CW** [15:8] Timer5 Capture Width Value T5CW [15:8] (R)

10.2.24 Timer5 Capture Cycle Byte Register (P_TMR5_CapCycle8, \$24) (R/W)

	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
W	T5PL15	T5PL14	T5PL13	T5PL12	T5PL11	T5PL10	T5PL9	T5PL8
R	T5CC7	T5CC6	T5CC5	T5CC4	T5CC3	T5CC2	T5CC1	T5CC0
	0	0	0	0	0	0	0	0

Bit [7:0] Write: **T5PL** [15:8] Timer5 Pre-Load Value T5PL [15:8] (W)

Bit [7:0] Read: **T5CC** [7:0] Timer5Capture Cycle Value T5CC [7:0] (R)

10.2.25 Timer5 Capture Cycle Low Byte Register (P_TMR5_CapCycleLo, \$25) (R)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
T5CC7	T5CC6	T5CC5	T5CC4	T5CC3	T5CC2	T5CC1	T5CC0
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

 Bit [7:0] Read :**T5CC** [15:8] Timer5 Capture Cycle Value T5CC [7:0] (R)

10.2.26 Timer5 Capture Cycle High Byte Register (P_TMR5_CapCycleHi, \$5F) (R)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
T5CC15	T5CC14	T5CC13	T5CC12	T5CC11	T5CC10	T5CC9	T5CC8
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

 Bit [7:0] Read :**T5CC** [15:8] Timer5 Capture Cycle Value T5CC [15:8] (R)

10.2.27 Capture Control Register (P_CAP_Ctrl, \$58) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
CAPOPT	CAPIP4	CAPIP3	CAP1P2	CAPIP1	CAP1P0	CAP1ES	CAP0ES
R/W							
0	0	0	0	0	0	0	0

 Bit 7 **CAPOPT:** Capture data hold selection bit.

1=Hold capture data

0=Update captured date if new data is received

 Bit 6 **CAPIP4:** Capture4 interrupt evoke polarity selection bit

1=Be same with CAP4ES edge setting

0=Be opposite to CAP4ES edge setting

 Bit 5 **CAPIP3:** Capture3 interrupt evoke polarity selection bit

1=Be same with CAP3ES edge setting

0=Be opposite to CAP3ES edge setting

 Bit 4 **CAPIP2:** Capture2 interrupt evoke polarity selection bit

1=Be same with CAP2ES edge setting

0=Be opposite to CAP2ES edge setting

 Bit 3 **CAPIP1:** Capture1 interrupt evoke polarity selection bit

1=Be same with CAP1ES edge setting

0=Be opposite to CAP1ES edge setting

 Bit 2 **CAPIP0:** Capture0 interrupt evoke polarity selection bit

1=Be same with CAP0ES edge setting

0=Be opposite to CAP0ES edge setting

 Bit 1 **CAP1ES:** Polarity edge selection bit in Capture1 of Timer1 bit

1=Falling edge clear counter

0=Rising edge clear counter

Bit 0 **CAP0ES:** Polarity edge selection bit in Capture0 of Timer0 bit

1= Falling edge clear counter

0= Rising edge clear counter

10.3 Operation

When Timer is in 8/16-bit Capture mode, capture input pin (e.g. Timer0 VS PB0) must be set as input mode and then acts as the capture event input. The prescaler must be selected properly in order to get suitable resolution. According to getting width or period of captured signal, the capture clear polarity bit and interrupt evoke polarity bit must be selected rightly. Some registers related Capture operations are P_CAP_Ctrl (\$58), P_IRQ_Opt0 (\$33) and P_IRQ_Opt1 (\$34). Figure 10-1 shows Capture Block Diagram. For a step-by-step procedure on how to set up the Capture interrupt, see Section 10.4 in detail.

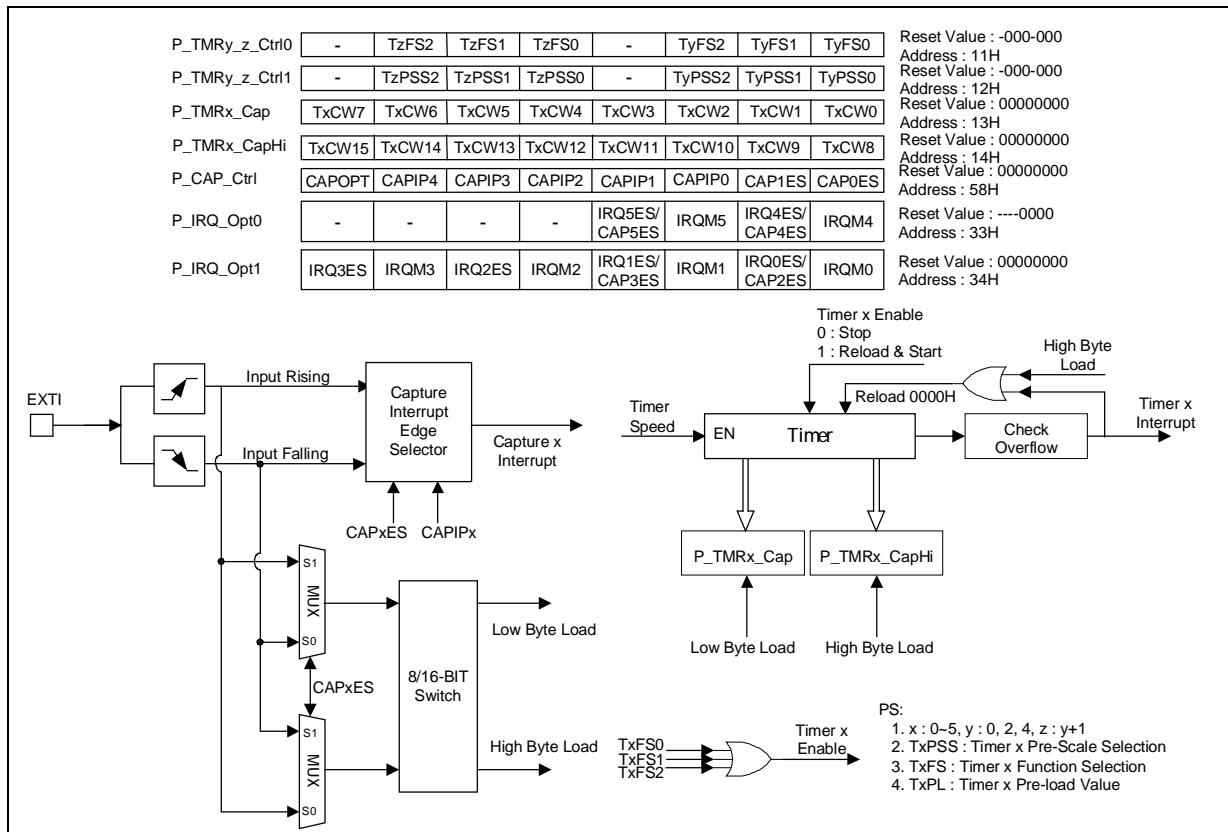


Figure 10-1 Capture Block Diagram

10.3.1 8-bit Capture

When 8-bit capture mode is selected, the suggested setting of the interrupt evoke polarity is the same with capture clear polarity. For example, if the capture clear polarity is set as falling (CAPxES=1) and interrupt evoke polarity is the same with capture clear polarity (CAPIPx = 1), then the counter is cleared on the falling edge of the external input signal. On the following rising edge, the counter value is transferred to P_TMRx_Cap that represents the pulse width value and then the counter continue counting. On the next falling edge, the counter value is transferred to P_TMRx_CapHi that represents cycle value and a CAPxIF interrupt flag is generated and the counter value is cleared again. The counter continues to operate next capture if CAPOPT=0(Update captured

date if new data is received(CAPOPT = 0), otherwise data register keep its value(CAPOPT = 1).

In addition, the timer overflow interrupt in capture mode is very useful when pulse width of captured signal is wider than the maximum period of timer. For example, the pulse width of captured signal is wider than the timer data value (\$33) over 1 timer. When capture interrupt is occurred, the captured value is less than wanted value. It can be obtained correct value by counting the number of timer overflow occurrence.

There is still a difference in 8-bit Capture mode between 8-bit timers and 16-bit timers. In 16-bit timers, except timer 5, they can only be used for measuring the pulse width data because they have to take two bytes of registers for saving capture data. But in 8-bit timers, they can be used to measure both width and cycle of pulse. The 8-Bit capture timing is shown in Figure 10-2.

The Capture module also provides a hold function for user to catch input signal once. By setting the P_CAP_Ctrl.CAPOPT to 1, the input signal will only be captured once. After first capture, the data will be hold till user read them out. Once user read the capture data, the input signal will be captured and hold again. This feature can be used to measure single pulse width/cycle. The 8-Bit capture and hold operation is shown in Figure 10-3.

8-bit Capture Width and Cycle calculation is shown in Equation 10-1.

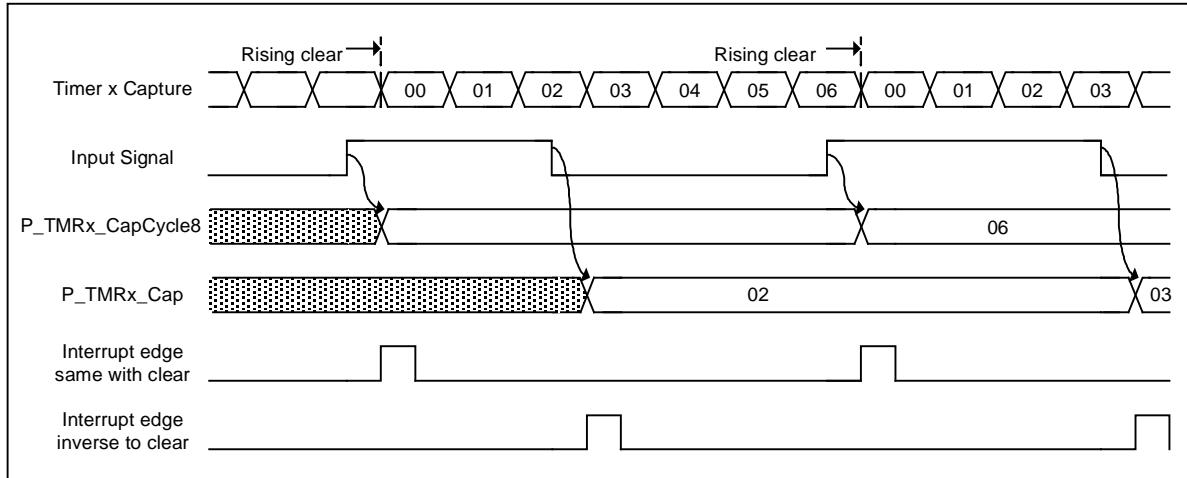


Figure 10-2 8-Bit Capture Waveform

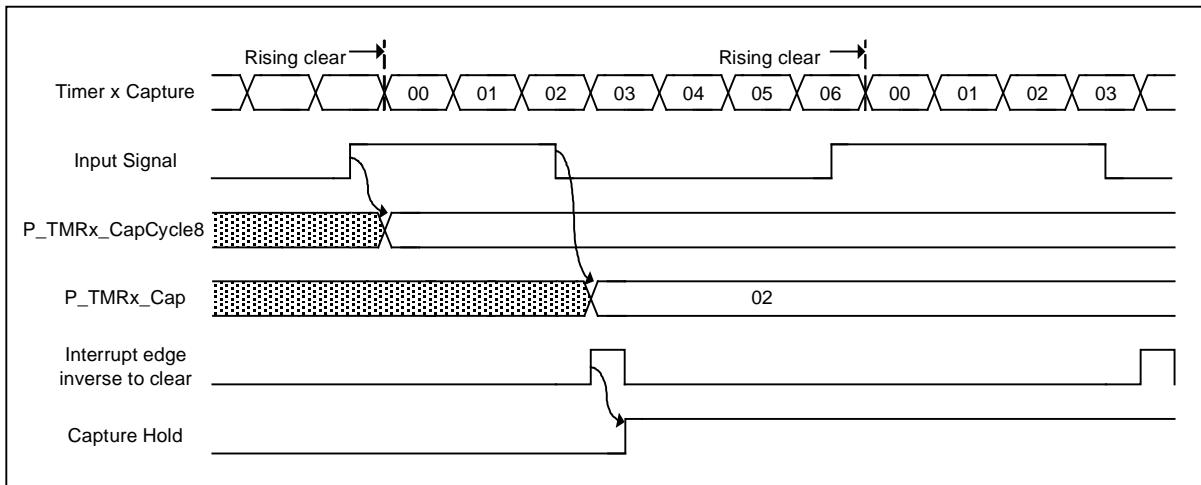


Figure 10-3 8-Bit Capture and Hold waveform

```

if
T_TMRx = Timer x Clock Period after pre-scaling, x = 0 ~ 5
then
8-bit timer/counter

```

$$\text{Capture Width} = [\text{P_TMRx_Cap} + 1] * \text{T_TMRx}$$

$$\text{Capture Cycle} = [\text{P_TMRx_CapHi} + 1] * \text{T_TMRx}$$

Note: This capture width/cycle value should have one time x clock period tolerance.

Equation 10-1: Calculation for 8-bit Capture Width and Cycle

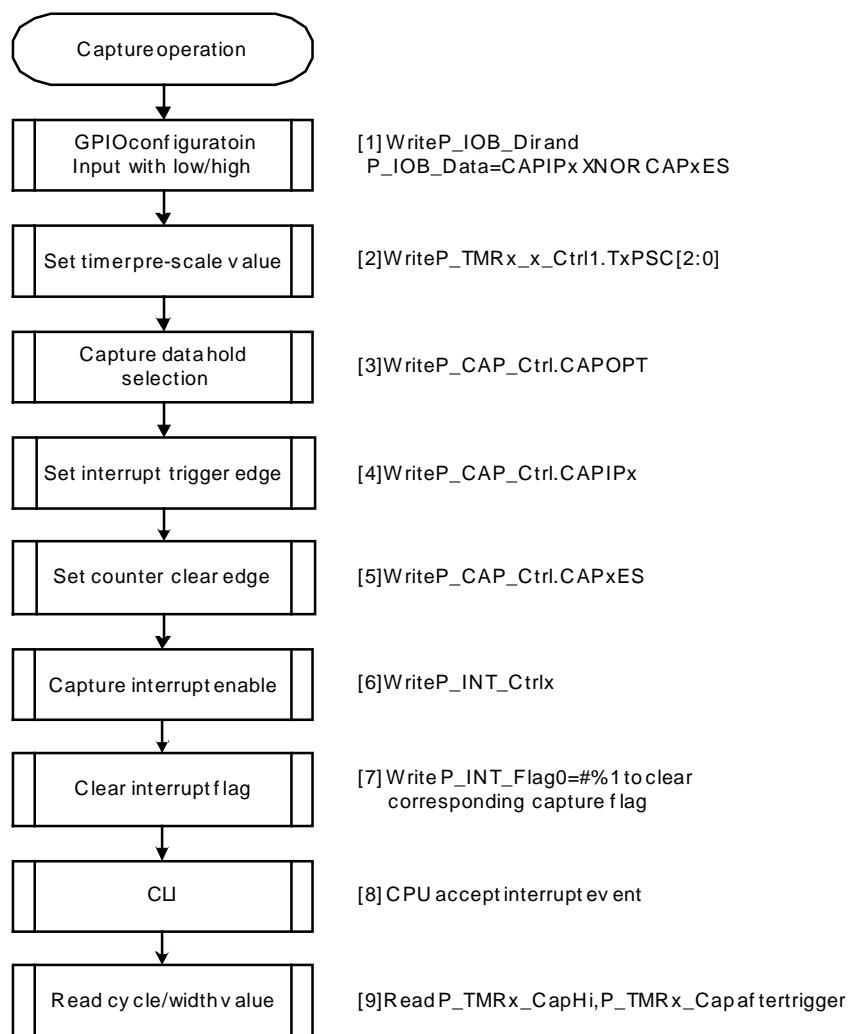


Figure 10-4 Capture software flowchart

[Example 10-1]: Set Timer0 as 8-bit capture operation.

lde	#00000000B	
sta	P_IOB_Data	
sta	P_IOB_Attrib	
lde	#11111110B	; Set PB0 as input pull low for CAP0
sta	P_IOB_Dir	
lde	#C_T0FCS_Div_512	; Set Timer0 clock source is Fsys/512(15.6KHz~61Hz)
sta	P_TMR0_1_Ctrl1	
lde	#C_T08B_CAP	; Set Timer0 is 8-bit capture
sta	P_TMR0_1_Ctrl0	
lde	#(C_CAP_IP0+C_CAP0_ES)	; Falling edge sample data, falling edge CAP0 int evoke.
sta	P_CAP_Ctrl	; Input pulse low width measurement on PB0

10.3.2 16-bit Capture

When 16-bit capture mode is selected, the suggested setting of interrupt evoke polarity is opposite to capture clear polarity. For example, if the capture clear polarity is set as falling (CAPxES=1) and the interrupt evoke polarity is opposite to capture clear polarity (CAPIPx=0), that is rising edge, then the counter is cleared while external input go falling. On the following rising edge, the counter values are transferred to P_TMRx_Cap and P_TMRx_CapHi, and a CAPxIF interrupt flag is generated, then the counter continues counting. On the next falling edge, the counter value is cleared and continues to operate for next capture if CAPOPT=0 (Update captured date if new data is received), otherwise data register keep its contents.

In addition, the timer overflow interrupt in capture mode is very useful when pulse width of captured signal is more wider than the maximum period of timer. For example, the pulse width of captured signal is wider than the timer data value (\$FFFF) over 1 timer. When capture interrupt is occurred, the captured value is less than wanted value. It can be obtained correct value by counting the number of timer overflow occurrence. As mentioned above, pulse width value of captured signal can be obtained easily. As to cycle calculation, user can exploit software with capture interruption to obtain cycle of captured signal. For example, when entering interrupt service routine, changed the interrupt evoke polarity. The cycle of captured signal can be obtained when next capture interrupt comes. Figure 10-5 shows 16-bit capture operation. 16-bit Capture Width calculation is shown in Equation 10-2.

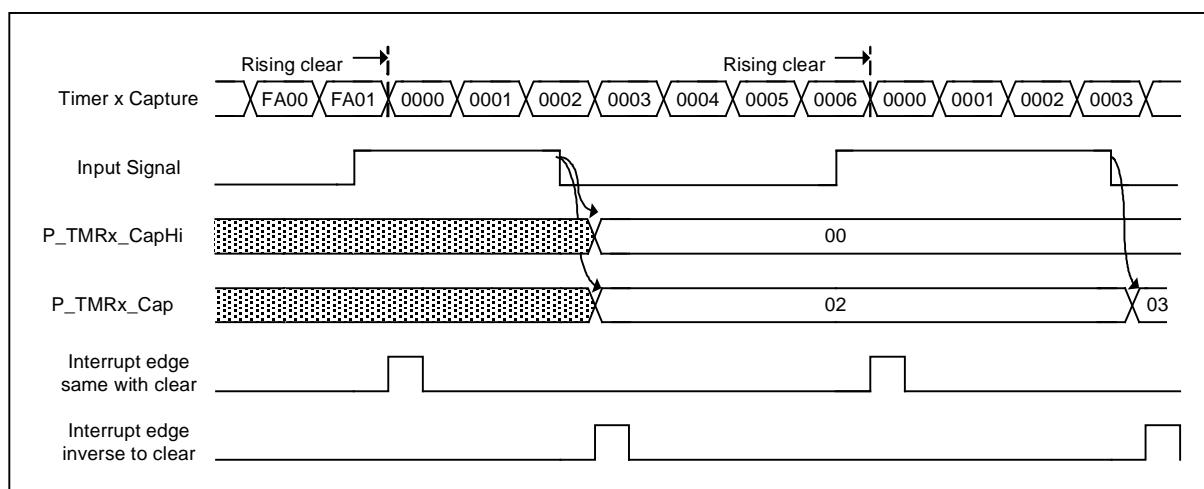


Figure 10-5 16-bit Capture Operation

The relationship between data register and capture width or period can be calculated from the below equation:

Equation 10-2: Calculation for 16-bit Capture Width

```
If
  T_TMRx = Timer x Clock Period after pre-scaling, x = 0 ~ 5
then
16-bit timer/counter
```

Capture Width=[{ P_TMRx_CapHi, P_TMRx_Cap }+1]* T_TMRx

Note: This capture width/cycle value should have one time x clock period tolerance.

[Example 10-2]: Set Timer1 as 16-bit capture operation .

```
lda      #00000000B
sta      P_IOB_Data
sta      P_IOB_Attrib
lda      #11111101B          ; Set PB1 as input pull low for CAP1
sta      P_IOB_Dir
lda      #C_T1FCS_Div_512    ; Set Timer1 clock source is Fsys/128(62.5KHz~1Hz)
sta      P_TMR0_1_Ctrl1
lda      #C_T116B_CAP         ; Set Timer1 is 16-bit capture
sta      P_TMR0_1_Ctrl0
lda      #(C_CAP_IP1+C_CAP1_ES) ; Falling edge sample data, falling edge CAP1 int evoke.
sta      P_CAP_Ctrl           ; Input pulse low width measurement on PB1
```

10.4 Capture Interrupt

Interrupts can come from many sources. There are five interrupt sources related with Capture.

- | Timer0 Capture Interrupt
- | Timer1 Capture Interrupt
- | Timer2 Capture Interrupt
- | Timer3 Capture Interrupt
- | Timer4 Capture Interrupt
- | Timer5 Capture Interrupt

There are several steps for enabling Capture interrupt.

1. Interrupt Disable flag (I-flag) is set to “1” by using “SEI ” instruction.
2. Decided 8-bit or 16-bit Capture and then configured timer x(x=0~5) function configuration.
3. Configured some registers related for capture operation in the timer x(x = 0~5) you selected.
4. Enable corresponding Timer x Capture interrupt flag in the P_IRQ_Opt0(\$33), P_IRQ_Opt1 \$34) register.

5. Using 'CLI' instruction to enable all interrupt function.
6. Now the Timer x Capture interrupt will be accepted.

[Example 10-3]: Set Timer1 as 16-bit capture operation and enable capture interrupt.

```

 lda      #00000000B
 sta      P_IOB_Data
 sta      P_IOB_Attrib
 lda      #11111101B          ; Set PB1 as input pull low for CAP1
 sta      P_IOB_Dir
 lda      #C_T1FCS_Div_512    ; Set Timer1 clock source is Fsys/128(62.5KHz~1Hz)
 sta      P_TMR0_1_Ctrl1
 lda      #C_T116B_CAP         ; Set Timer1 is 16-bit capture
 sta      P_TMR0_1_Ctrl0
 lda      #(C_CAP_IP1+C_CAP1_ES) ; Falling edge sample data, falling edge CAP1 int evoke.
 sta      P_CAP_Ctrl           ; Input pulse low width measurement on PB1

L_TestT1L61:
 lda      P_INT_Flag1
 and     # C_INT_CAP1IF       ; CAP1 INT flag ? (capture data ready)
 beq     L_TestT1L61          ; no
 set     P_INT_Flag1, CB_INT_CAP1IF
                           ; clear CAP1 INT flag
 lda      P_TMR1_Cap          ; A <- pulse low width low byte
 idx     P_TMR1_CapHi         ; X <- pulse low width high byte
 jmp     L_TestT1L61

```

10.5 Design Tips

[Example 10-4] Shows how to initialize the timer0 as capture operation based on SPMC65P2404A, please refer to an example of Fortis IDE.

- Use Timer0 8-bit timer
- Set Timer0 clock source as Fsys/512
- Set Timer0 is 8-bit capture width mode
- Falling edge clear, also evoke interrupt

```

.SYNTAX 6502                  ; process standard 6502 addressing syntax
.LINKLIST                         ; generate linklist information
.SYMBOLS                            ; generate symbolic debug information

.INCLUDE    spmc65p2404a.inc

.PAGE0                                ; define values in the range from $00 to $FF
;
.DATA                                 ; define data storage section
;

```

```

.CODE
; **** Power on Reset Process - Main Program ****
;*
;*
;V_Reset:
    sei          ; Disable interrupt
    ldx #C_STACK_BOTTOM ; Initial stack pointer at $00FF
    txa          ; Transfer to stack point
;
    lda #$FF      ; Clear Reset flag
    sta P_SYS_Ctrl
    sta P_SYS_Ctrl
;
    jsr F_InitIOPort ; initial GPIO port
;
    lda #$00
    sta P_IOB_Data
    sta P_IOB_Attrib
;
    lda #11111110B
    sta P_IOB_Dir ; PB0/TC0 as input low for CAP0
;
    lda #C_T0FCS_Div_512 ; Set Timer0 clock source is Fsys/512
    sta P_TMR0_1_Ctrl1
;
    lda #C_T08B_CAP ; Set Timer0 is 8-bit capture width mod
    sta P_TMR0_1_Ctrl0
;
    lda #(C_CAP0_ES + C_CAP_IP0); falling edge clear, also evoke interrupt
    sta P_CAP_Ctrl ; input pulse low width measurement on PB0
;
    lda #$FF      ; clear INT request flag
    sta P_INT_Flag0
    sta P_INT_Flag1
    set P_INT_Ctrl1, CB_INT_CAP0IE
;
    cli          ; enable Timer0 capture interrupt
;
    cli          ; enable INT
;
L_Main:
    nop
    jmp L_Main
;
```

```

F_InitIOPort:
;

;      /// Initial Port A      ///
;

    lda      #00000000B
    sta      P_IOA_Data
    lda      #00000000B
    sta      P_IOA_Attrib
    lda      #1111111B          ; PortA as input with pull-low
    sta      P_IOA_Dir
;

    rts
;
*****          *
;*          IRQ Interrupt Service Routine          *
;*          *
;*****          *

V_IRQ:
    pha              ; push A register
    txa              ; transfer X to A
    pha              ; push A register (ie. push X)
;

; Timer 0 capture interrupt ?
;

    lda      P_INT_Flag1
    and      #C_INT_CAP0IF
    beq      L_exit_irq
;

    set      P_INT_Flag1, CB_INT_CAP0IF
;

    lda      P_TMR0_Cap
    sta      G_CapWidthBuf0      ; if input frequency = 100 Hz, G_CapWidthBuf0 ~= 78
;

    lda      P_IOA_Data
    eor      #$FF
    sta      P_IOA_Data          ; toggle P_IOA_Data
;

L_exit_irq:
    pla              ; pop A register
    tax              ; transfer A to X
    pla              ; pop A register (ie. pop X)
;

    rti
;
```

```

; ****
;          *
;          * Interrupt Vector Table      *
;          *                                *
; ****
; VECTOR:      .SECTION
DW      V_NMI           ; may download program emulated either
DW      V_Reset          ; in internal memory or external memory
DW      V_IRQ            ; dw define two bytes interrupt vector
;
; ****
;          *
;          * End Of Interrupt Vector Table   *
;          *                                *
; ****
; .END           ; end of program

```

10.6 Related Application Notes and Libraries

This section lists some application notes and libraries about various usage of SPMC65X family. The application notes may help user develop quickly and offer the concepts about the usage of this topic. Sunplus also supports some powerful libraries about general operation, mathematics, communication, etc. The purpose of library is to help user speed up development. The current application notes and libraries related to Capture are listed below.

Application Note:

Title	Application Note Series Number
Infrared Remote control	AN_O0320.DOC
Zero cross detection	AN_O0322.DOC
Using the Timer0 CCP module	AN_O0331.DOC
Using the Timer1 CCP module	AN_O0332.DOC
Using the Timer2 CCP module	AN_O0333.DOC
Using the Timer3 CCP module	AN_O0334.DOC
Using the Timer4 CCP module	AN_O0335.DOC
Using the Timer5 CCP module	AN_O0336.DOC

Library:

Title	Application Note Series Number
SPMC65X series software basics manual	AN_O0100.DOC



SPMC65X series software advances 1 manual (data processing operation) AN_O0101.DOC

SPMC65X series software advances 2 manual (mathematical operation) AN_O0102.DOC

10.7 Revision History

Revision	Date	Remark
V0.1	03/31/2005	First edition

11 Compare

11.1 Description

In SPMC65X family, as many as 6 channels of Timer are equipped and each one can be set to operate as 8-bit/16-bit Compare function. Compare function can be easily configured with corresponding control registers setting. The behavior of compare is when timer matched compare register, the corresponding pin decided which timer is in use will pulse out with 50:50 duty square. The detailed compare output waveforms are shown in Figure 11-1 . Table 11-1 shows the summary of compare function for SPMC65X family. Detailed registers setting are described in Section11.2.

	Timer/Event Counter		Capture		Compare		PWM
	8 bit	16 bit	8 bit	16 bit	8 bit	16 bit	
Timer 0	YES	YES	Width/Cycle	Width only	YES	YES	8 bit
Timer 1	YES	YES	Width/Cycle	Width only	YES	YES	12 bit
Timer 2	YES	YES	Width/Cycle	Width only	YES	YES	8 bit
Timer 3	YES	YES	Width/Cycle	Width only	YES	YES	12 bit
Timer 4	YES	YES	Width/Cycle	Width only	YES	YES	8 bit
Timer 5	YES	YES	Width/Cycle	Width/Cycle	YES	YES	16 bit

Table 11-1 Summary of Compare function for SPMC65X family

The timers have three types depending on different features:

1. Type I: (Timer0, Timer2 & Timer4)
 - 8 or 16-bit timer/counter
 - 8 or 16-bit capture mode (8-bit with width/cycle measurement, 16-bit with width measurement)
 - **8 or 16-bit compare mode**
 - 8-bit PWM mode
2. Type II: (Timer1 & Timer3)
 - 8 or 16-bit timer/counter
 - 8 or 16-bit capture mode (8-bit with width/cycle measurement, 16-bit with width measurement)
 - **8 or 16-bit compare mode**
 - 12-bit PWM mode
3. Type III: (Timer5)
 - 8 or 16-bit timer/counter
 - 8 or 16-bit capture mode (8 or 16-bit with width/cycle measurement)
 - **8 or 16-bit compare mode**

n 16-bit PWM mode

11.2 Control Register

11.2.1 Timer0_1 Control Register 0 (P_TMR0_1_Ctrl0, \$11) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
-	T1FS2	T1FS1	T1FS0	-	T0FS2	T0FS1	T0FS0
-	R/W	R/W	R/W	-	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit 7 Reserved

Bit [6:4] **T1FS[2 : 0]**: Timer 1 function configuration bits

111 = 12-bit PWM

110 = 16-bit capture (Width)

101 = 16-bit Compare

100 = 16-bit Timer

011 = 8-bit Capture (Width, Cycle)

010 = 8-bit Compare

001 = 8-bit Timer

000 = Disable

Bit 3 Reserved

Bit [2:0] **T0FS[2 : 0]**: Timer0 Function Configuration bits

111 = 8-bit PWM

110 = 16-bit capture (Width)

101 = 16-bit Compare

100 = 16-bit Timer

011 = 8-bit Capture (Width, Cycle)

010 = 8-bit Compare

001 = 8-bit Timer

000 = Disable

11.2.2 Timer0_1 Control Register 1 (P_TMR0_1_Ctrl1, \$12) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
-	T1PSS2	T1PSS1	T1PSS0	-	T0PSS2	T0PSS1	T0PSS0
-	R/W	R/W	R/W	-	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit 7 Reserved

Bit [6:4] **T1PSS[2 : 0]**: Timer1 Pre-Scale Configuration bits

111 = External Event

110 = $F_{SYS} \div 512$

101 = $F_{SYS} \div 128$

$$100 = F_{SYS} \div 32$$

$$011 = F_{SYS} \div 8$$

$$010 = F_{SYS} \div 4$$

$$001 = F_{SYS} \div 2$$

$$000 = F_{SYS}$$

Bit 3 Reserved

Bit [2:0] **T0PSS[2 : 0]**: Timer0 Pre-Scale Configuration bits

$$111 = \text{External Event}$$

$$110 = F_{SYS} \div 512$$

$$101 = F_{SYS} \div 128$$

$$100 = F_{SYS} \div 32$$

$$011 = F_{SYS} \div 8$$

$$010 = F_{SYS} \div 4$$

$$001 = F_{SYS} \div 2$$

$$000 = F_{SYS}$$

F_{SYS} : Frequency of Clock Source

11.2.3 Timer0 Compare Low Byte Register (**P_TMR0_Comp**, \$13) (R/W)

	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
W	T0CP7	T0CP6	T0CP5	T0CP4	T0CP3	T0CP2	T0CP1	T0CP0
R	T0CN7	T0CN6	T0CN5	T0CN4	T0CN3	T0CN2	T0CN1	T0CN0
	0	0	0	0	0	0	0	0

Bit [7:0] Write: **T0CP** [7:0] Timer0 Compare Value T0CP [7:0] (W)

Bit [7:0] Read: **T0CN** [7:0] Timer0 Count Value T0CN [7:0] (R)

Note: Writing dataflow: Write high byte first then low byte, i.e., 16-bit timer mode: Write P_TMR0_CompHi first then P_TMR0_Comp

11.2.4 Timer0 Compare High Byte Register (**P_TMR0_CompHi**, \$14) (R/W)

	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
W	T0CP15	T0CP14	T0CP13	T0CP12	T0CP11	T0CP10	T0CP9	T0CP8
R	T0CN15	T0CN14	T0CN13	T0CN12	T0CN11	T0CN10	T0CN9	T0CN8
	0	0	0	0	0	0	0	0

Bit [7:0] Write: **T0CP** [15:8] Timer0 Compare Value T0CP [15:8] (W)

Bit [7:0] Read: **T0CN** [15:8] Timer0 Count Value T0CN [15:8] (R)

11.2.5 Timer1 Compare Low Byte Register (P_TMR1_Comp, \$15) (R/W)

	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
W	T1CP7	T1CP6	T1CP5	T1CP4	T1CP3	T1CP2	T1CP1	T1CP0
R	T1CN7	T1CN6	T1CN5	T1CN4	T1CN3	T1CN2	T1CN1	T1CN0
	0	0	0	0	0	0	0	0

Bit [7:0] Write: **T1CP** [7:0] Timer1 Compare Value T1CP [7:0] (W)

Bit [7:0] Read: **T1CN** [7:0] Timer1 Count Value T1CN [7:0] (R)

Note: Writing dataflow: Write high byte first then low byte, i.e., 16-bit timer mode: Write P_TMR1_CompHi first then P_TMR1_Comp

11.2.6 Timer1 Compare High Byte Register (P_TMR1_CompHi, \$16) (R/W)

	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
W	T1CP15	T1CP14	T1CP13	T1CP12	T1CP11	T1CP10	T1CP9	T1CP8
R	T1CN15	T1CN14	T1CN13	T1CN12	T1CN11	T1CN10	T1CN9	T1CN8
	0	0	0	0	0	0	0	0

Bit [7:0] Write: **T1CP** [15:8] Timer1 compare Value T1CP [15:8] (W)

Bit [7:0] Read: **T1CN** [15:8] Timer1 Count Value T1CN [15:8] (R)

11.2.7 Timer2_3 Control Register 0 (P_TMR2_3_Ctrl0, \$18) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
-	T3FS2	T3FS1	T3FS0	-	T2FS2	T2FS1	T2FS0
-	R/W	R/W	R/W	-	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit 7 Reserved

Bit [6:4] **T3FS[2 : 0]**: Timer 3 function configuration bits

111 = 12-bit PWM

110 = 16-bit capture (Width)

101 = 16-bit Compare

100 = 16-bit Timer

011 = 8-bit Capture (Width, Cycle)

010 = 8-bit Compare

001 = 8-bit Timer

000 = Disable

Bit 3 Reserved

Bit [2:0] **T2FS[2 : 0]**: Timer2 Function Configuration bits

111 = 8-bit PWM

110 = 16-bit capture (Width)

101 = 16-bit Compare

100 = 16-bit Timer

011 = 8-bit Capture (Width, Cycle)

010 = 8-bit Compare

001 = 8-bit Timer

000 = Disable

11.2.8 Timer2_3 Control Register 1 (P_TMR2_3_Ctrl1, \$19) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
-	T3PSS2	T3PSS1	T3PSS0	-	T2PSS2	T2PSS1	T2PSS0
-	R/W	R/W	R/W	-	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit 7 Reserved

Bit [6:4] **T3PSS** [2 : 0]: Timer3 Pre-Scale Configuration bits

111 = External Event

110 = $F_{SYS} \div 512$

101 = $F_{SYS} \div 128$

100 = $F_{SYS} \div 32$

011 = $F_{SYS} \div 8$

010 = $F_{SYS} \div 4$

001 = $F_{SYS} \div 2$

000 = F_{SYS}

Bit 3 Reserved

Bit [2:0] **T2PSS** [2 : 0]: Timer2 Pre-Scale Configuration bits

111 = External Event

110 = $F_{SYS} \div 512$

101 = $F_{SYS} \div 128$

100 = $F_{SYS} \div 32$

011 = $F_{SYS} \div 8$

010 = $F_{SYS} \div 4$

001 = $F_{SYS} \div 2$

000 = F_{SYS}

F_{SYS} : Frequency of Clock Source

11.2.9 Timer2 Compare Low Byte Register (P_TMR2_Comp, \$1A) (R/W)

	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
W	T2CP7	T2CP6	T2CP5	T2CP4	T2CP3	T2CP2	T2CP1	T2CP0
R	T2CN7	T2CN6	T2CN5	T2CN4	T2CN3	T2CN2	T2CN1	T2CN0
	0	0	0	0	0	0	0	0

Bit [7:0] Write: **T2CP** [7:0] Timer2 Compare Value T2CP [7:0] (W)

Bit [7:0] Read: **T2CN** [7:0] Timer2 Count Value T2CN [7:0] (R)

Note: Writing dataflow: Write high byte first then low byte, i.e., 16-bit timer mode: Write P_TMR2_CompHi first then P_TMR2_Comp

11.2.10 Timer2 Compare High Byte Register (P_TMR2_CompHi, \$1B) (R/W)

	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
W	T2CP15	T2CP14	T2CP13	T2CP12	T2CP11	T2CP10	T2CP9	T2CP8
R	T2CN15	T2CN14	T2CN13	T2CN12	T2CN11	T2CN10	T2CN9	T2CN8
	0	0	0	0	0	0	0	0

Bit [7:0] Write: **T2CP** [15:8] Timer2 compare Value T2CP [15:8] (W)

Bit [7:0] Read: **T2CN** [15:8] Timer2 Count Value T2CN [15:8] (R)

11.2.11 Timer3 Compare Low Byte Register (P_TMR3_Comp, \$1C) (R/W)

	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
W	T3CP7	T3CP6	T3CP5	T3CP4	T3CP3	T3CP2	T3CP1	T3CP0
R	T3CN7	T3CN6	T3CN5	T3CN4	T3CN3	T3CN2	T3CN1	T3CN0
	0	0	0	0	0	0	0	0

Bit [7:0] Write: **T3CP** [7:0] Timer3 Compare Value T3CP [7:0] (W)

Bit [7:0] Read: **T3CN** [7:0] Timer3 Count Value T3CN [7:0] (R)

Note: Writing dataflow: Write high byte first then low byte, i.e., 16-bit timer mode:

Write P_TMR3_CompHi first then P_TMR3_Comp

11.2.12 Timer3 Compare High Byte Register (P_TMR3_CompHi, \$1D) (R/W)

	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
W	T3CP15	T3CP14	T3CP13	T3CP12	T3CP11	T3CP10	T3CP9	T3CP8
R	T3CN15	T3CN14	T3CN13	T3CN12	T3CN11	T3CN10	T3CN9	T3CN8
	0	0	0	0	0	0	0	0

Bit [7:0] Write: **T3CP** [15:8] Timer3 Compare Value T3CP [15:8] (W)

Bit [7:0] Read: **T3CN** [15:8] Timer3 Count Value T3CN [15:8] (R)

11.2.13 Timer4_5 Control Register 0 (P_TMR4_5_Ctrl0, \$1F) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
-	T5FS2	T5FS1	T5FS0	-	T4FS2	T4FS1	T4FS0
-	R/W	R/W	R/W	-	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit 7 Reserved

Bit [6:4] **T5FS[2 : 0]**: Timer 5 function configuration bits

111 = 16-bit PWM

110 = 16-bit capture (Width)

101 = 16-bit Compare

100 = 16-bit Timer

011 = 8-bit Capture (Width, Cycle)

010 = 8-bit Compare

001 = 8-bit Timer

000 = Disable

Bit 3 Reserved

Bit [2:0] **T4FS[2 : 0]**: Timer4 Function Configuration bits

111 = 8-bit PWM

110 = 16-bit capture (Width)

101 = 16-bit Compare

100 = 16-bit Timer

011 = 8-bit Capture (Width, Cycle)

010 = 8-bit Compare

001 = 8-bit Timer

000 = Disable

11.2.14 Timer4_5 Control Register 1 (P_TMR4_5_Ctrl1, \$20) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
-	T5PSS2	T5PSS1	T5PSS0	-	T4PSS2	T4PSS1	T4PSS0
-	R/W	R/W	R/W	-	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit 7 Reserved

Bit [6:4] **T5PSS [2 : 0]**: Timer5 Pre-Scale Configuration bits

111 = External Event

110 = $F_{SYS} \div 512$

101 = $F_{SYS} \div 128$

$$100 = F_{SYS} \div 32$$

$$011 = F_{SYS} \div 8$$

$$010 = F_{SYS} \div 4$$

$$001 = F_{SYS} \div 2$$

$$000 = F_{SYS}$$

Bit 3 Reserved

Bit [2:0] **T4PSS** [2 : 0]: Timer4 Pre-Scale Configuration bits

$$111 = \text{External Event}$$

$$110 = F_{SYS} \div 512$$

$$101 = F_{SYS} \div 128$$

$$100 = F_{SYS} \div 32$$

$$011 = F_{SYS} \div 8$$

$$010 = F_{SYS} \div 4$$

$$001 = F_{SYS} \div 2$$

$$000 = F_{SYS}$$

F_{SYS} : Frequency of Clock Source

11.2.15 Timer4 Compare Low Byte Register (**P_TMR4_Comp**, \$21) (R/W)

	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
W	T4CP7	T4CP6	T4CP5	T4CP4	T4CP3	T4CP2	T4CP1	T4CP0
R	T4CN7	T4CN6	T4CN5	T4CN4	T4CN3	T4CN2	T4CN1	T4CN0
	0	0	0	0	0	0	0	0

Bit [7:0] Write: **T4CP** [7:0] Timer4 Compare Value T4CP [7:0] (W)

Bit [7:0] Read: **T4CN** [7:0] Timer4 Count Value T4CN [7:0] (R)

Note: Writing dataflow: Write high byte first then low byte, i.e., 16-bit timer mode: Write P_TMR4_CompHi first then P_TMR4_Comp

11.2.16 Timer4 Compare High Byte Register (**P_TMR4_CompHi**, \$22) (R/W)

	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
W	T4CP15	T4CP14	T4CP13	T4CP12	T4CP11	T4CP10	T4CP9	T4CP8
R	T4CN15	T4CN14	T4CN13	T4CN12	T4CN11	T4CN10	T4CN9	T4CN8
	0	0	0	0	0	0	0	0

Bit [7:0] Write: **T4CP** [15:8] Timer4 Compare Value T4CP [15:8] (W)

Bit [7:0] Read: **T4CN** [15:8] Timer4 Count Value T4CN [15:8] (R)

11.2.17 Timer5 Compare Low Byte Register (P_TMR5_Comp, \$23) (R/W)

	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
W	T5CP7	T5CP6	T5CP5	T5CP4	T5CP3	T5CP2	T5CP1	T5CP0
R	T5CN7	T5CN6	T5CN5	T5CN4	T5CN3	T5CN2	T5CN1	T5CN0
	0	0	0	0	0	0	0	0

Bit [7:0] Write: **T5CP** [7:0] Timer5 Compare Value T5CP [7:0] (W)

Bit [7:0] Read: **T5CN** [7:0] Timer5 Count Value T5CN [7:0] (R)

Note: Writing dataflow: Write high byte first then low byte, i.e., 16-bit timer mode: Write P_TMR5_CompHi first then P_TMR5_Comp

11.2.18 Timer5 Compare High Byte Register (P_TMR5_CompHi, \$24) (R/W)

	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
W	T5CP15	T5CP14	T5CP13	T5CP12	T5CP11	T5CP10	T5CP9	T5CP8
R	T5CN15	T5CN14	T5CN13	T5CN12	T5CN11	T5CN10	T5CN9	T5CN8
	0	0	0	0	0	0	0	0

Bit [7:0] Write: **T5CP** [15:8] Timer5 Compare Value T5CP [15:8] (W)

Bit [7:0] Read: **T5CN** [15:8] Timer5 Count Value T5CN [15:8] (R)

11.3 Operation

When Timer x acts as an 8/16-bit Compare output, corresponding pin (e.g. Timer 0 = PB2) outputs up to an 8/16-bit resolution compare output with 7 stages pre-scale selection. The compare data is set at “Timer x Compare Register” – P_TMRx_Comp and “Timer x Compare High Byte Register P_TMRxCompHi (16-bit only). The timer x counts up from the compare data and when the timer x generate the 1st overflow signal, the compare output a high level to corresponding pin. When the timer x generates the 2nd overflow signal, the compare output outputs a low level to corresponding pin. In other words, the waveform of corresponding pin always outputted with 50:50 duty square.

11.3.1 8-bit Compare

8-bit compare output is shown in Figure 11-1, When 8-bit compare mode is selected and take timer0 for example, T0FS[2 : 0] in P_TMR0_1_Ctrl0 (\$12) register should be set as “010” and compare data should be set into P_TMR0_Comp (\$13) register. When the timer0 counts up from the compare data and generates the 1st overflow signal, the compare output outputs a high level to PB2 and Timer0 overflow interrupt occurred simultaneously if timer 0 overflow interrupt enable flag is set to ‘1’. When the timer0 generates the 2nd overflow signal, the compare output outputs a low level to PB2. In other words, the duty of compare output is fixed to 50. Detailed compare output waveforms are shown in Figure 11-1. The output frequency of Compare operation can be

calculated using Equation 11-1. For a step-by-step procedure on how to set up the Compare overflow interrupt, see Section 11.4 in detail.

Equation 11-1: Calculation for 8-bit Compare operation

$$f_{\text{Comp}} = \frac{F_{\text{sys}}}{\text{Timer_prescaler} \times (256 - \text{Timer_Compare_value})} \times \frac{1}{2}$$

f_{Comp} : Compare output frequency

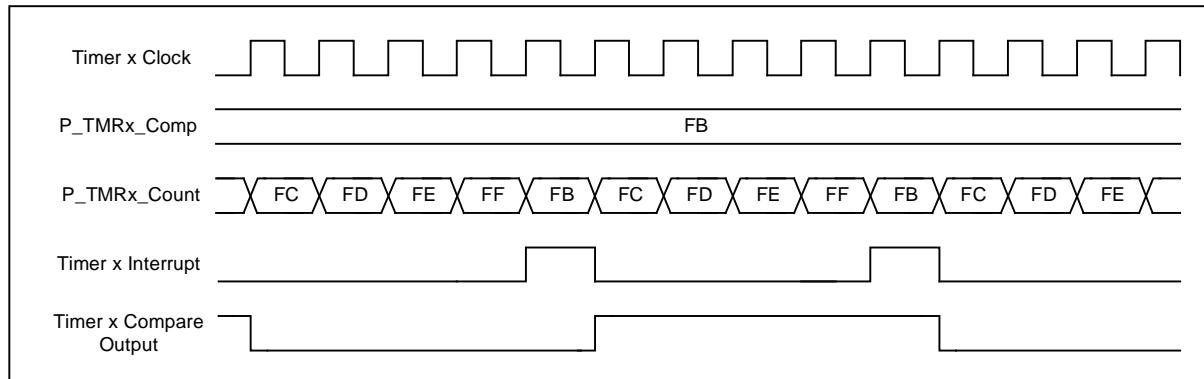


Figure 11-1 8-bit Compare waveform

【Example 11-1】: Set Timer0 as 8-bit timer Compare operation.

```

lDa      #156           ; Before starting timer, set Timer0 counter initial value first.
sta      P_TMR0_Preload   ;
lDa      #C_T1FCS_Div_128 ; Set Timer0 clock source is Fsys/128
sta      P_TMR0_1_Ctrl1
lDa      #C_T08B_COMP     ; Set Timer0 is 8-bit compare output
sta      P_TMR0_1_Ctrl0
lDa      #156             ; Set Timer0 preload counter= 256-156= 100
sta      P_TMR0_Preload   ; Fsys(8MHz)/128/100= 625Hz on PB2

```

11.3.2 16-bit Compare

16-bit compare operation is similar to 8-bit compare except that compare registers composed of two registers can be set to 16-bit-wide. Take timer1 for example, in 16-bit Compare mode, the high bytes of 8-bit data should be set to P_TMR1_CompHi (\$16) register, the low bytes of 8-bit data should be set to P_TMR1_Comp (\$15) register

and T1FS[2 : 0] in P_TMR0_1_Ctrl0 (\$12) register should be set as “101”. When the timer1 counts up from the compare data and generates the 1st overflow signal, the compare output outputs a high level to PB3 and Timer1 overflow interrupt occurred simultaneously if timer 1 overflow interrupt enable flag is set to ‘1’. When the timer1 generates the 2nd overflow signal, the compare output outputs a low level to PB3. In other words, the duty of compare output is fixed to 50. Equation 11-2 shows the formula of 16-bit compare output frequency. Detailed compare output waveforms are shown in Figure 11-2.

$$f_{Comp} = \frac{F_{sys}}{\text{Timer_prescaler} \times (65536 - \text{Timer_Compare_value})} \times \frac{1}{2}$$

f_{Comp} : Compare output frequency

Equation 11-2: Calculation for 16-bit Compare operation

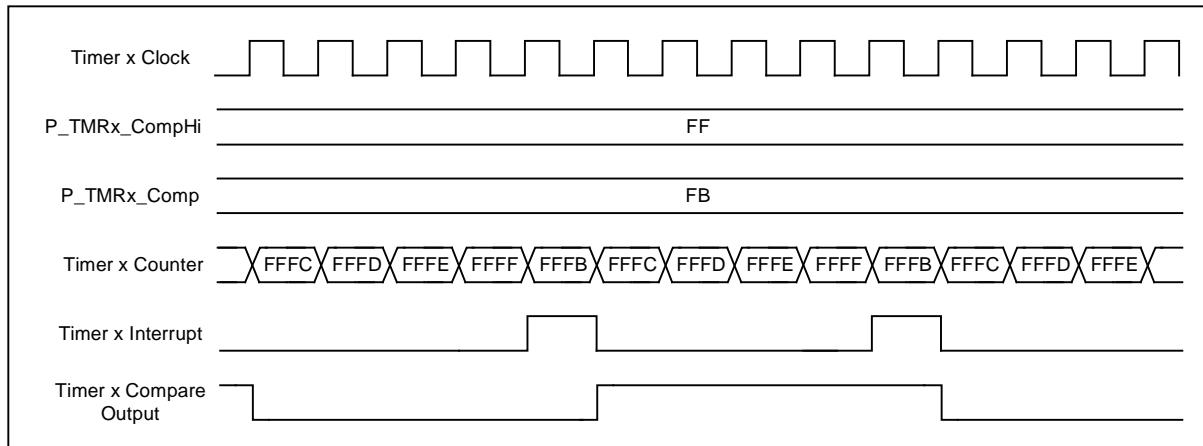


Figure 11-2 16-Bit Compare Waveforms

【Example 11-2】: Set Timer1 as 16-bit compare operation and generate output by Timer1 compare function.

```

    lda      #02                      ; Before starting timer, set Timer1 counter initial value first.
    sta      P_TMR1_PreloadHi        ; Set high byte
    lda      #143                     ;
    sta      P_TMR1_Preload          ; Set low byte
    lda      #C_T1FCS_Div_128       ; Set Timer1 clock source is Fsys/128
    sta      P_TMR0_1_Ctrl1
    lda      #C_T116B_COMP           ; Set Timer1 is 16-bit compare output
    sta      P_TMR0_1_Ctrl0
  
```

```
lde      #02          ; 1'st set Timer1 preload high byte counter= 2x 256= 512
sta     P_TMR1_PreloadHi
lde      #143         ; 2'nd set Timer1 preload low byte counter= 256-143= 113
sta     P_TMR1_Preload    ; Fsys(8Mhz)/128/625= 100Hz(10ms)on PB3
```

11.4 Compare Interrupt

Interrupts can come from many sources. There are five interrupt sources related with Compare.

- | Timer0 Compare Interrupt
- | Timer1 Compare Interrupt
- | Timer2 Compare Interrupt
- | Timer3 Compare Interrupt
- | Timer4 Compare Interrupt
- | Timer5 Compare Interrupt

There are several steps for enabling Timer Compare interrupt.

1. Interrupt Disable flag (I-flag) is set to “1” by using “SEI ” instruction.
2. Decided 8-bit or 16-bit timer and then selected a suitable timer for use.
3. Configured some registers related for compare operation in the timer x(x = 0~5) you selected.
4. Enable corresponding timer x interrupt flag in the P_INT_Ctrl1(\$0F) register.
5. Using ‘CLI’ instruction to enable all interrupt function.
6. Now the timer Compare interrupt will be accepted.

11.5 Design Tips

[Example 11-3]: Shows how to initialize the Timer1 as compare operation based on SPMC65P2404A. Please refer to an example of Fortis IDE.

- n Use Timer1 16-bit compare output
- n Set Timer1 clock source as Fcs/2
- n Output frequency is 61 Hz on PB3

```

.SYNTAX 6502           ; process standard 6502 addressing syntax
.LINKLIST            ; generate linklist information
.SYMBOLS              ; generate symbolic debug information

.INCLUDE      spmc65p2404a.inc

.PAGE0                 ; define values in the range from 00h to FFh
;
; .DATA                  ; define data storage section
;
; .CODE
; ****
;*
;* Power on Reset Process - Main Program *
;*
; ****

V_Reset:
    sei          ; Disable interrupt
    ldx #C_STACK_BOTTOM ; Initial stack pointer at $00ff
    txs          ; Transfer to stack point
;
    lda #$FF      ; Clear Reset flag
    sta P_SYS_Ctrl
    sta P_SYS_Ctrl
;
    jsr F_InitIOPort ; initial GPIO port;
    lda #$80      ; Fsys (8MHz) / (2 x ($FFFF - $8000)) = 61 Hz on PB3
    sta P_TMR1_CompHi
    lda #00
    sta P_TMR1_Comp
;
    lda #C_T1FCS_Div_2 ; Set Timer1 clock source is Fcs/2
    sta P_TMR0_1_Ctrl1
    lda #C_T116B_COMP   ; Set Timer1 is 16-bit compare output
    sta P_TMR0_1_Ctrl0
;
    lda #$80      ; Fsys (8MHz) / (2 x ($FFFF - $8000)) = 61 Hz on PB3
    sta P_TMR1_CompHi
    lda #00
    sta P_TMR1_Comp
;
L_Main:
    nop
    jmp L_Main

```

```
;
F_InitIOPort:
;
;      /// Initial Port A      ///
;

    lda      #00000000B
    sta      P_IOA_Data
    lda      #00000000B
    sta      P_IOA_Attrib
    lda      #1111111B          ; PortA as input with pull-low
    sta      P_IOA_Dir
;

    rts
;
***** *
;*
;*      Interrupt Vector Table *
;*
;*
***** *
;*
;*      VECTOR:      .SECTION
;*      DW      V_NMI           ; may download program emulated either
;*      DW      V_Reset         ; in internal memory or external memory
;*      DW      V_IRQ           ; dw define two bytes interrupt vector
;*
;*
***** *
;*
;*      End Of Interrupt Vector Table *
;*
;*
***** *
;*
;*      .END                   ; end of program
;
```

11.6 Related Application Notes and Libraries

This section lists some application notes and libraries about various usage of SPMC65X family. The application notes may help user develop quickly and offer the concepts about the usage of this topic. Sunplus also supports some powerful libraries about general operation, mathematics, communication, etc. The purpose of library is to help user speed up development. The current application notes and libraries related to Compare are listed below.

Application Note:

Title	Application Note Series Number
Using the Timer0 CCP module	AN_O0331.DOC
Using the Timer1 CCP module	AN_O0332.DOC
Using the Timer2 CCP module	AN_O0333.DOC

Using the Timer3 CCP module	AN_O0334.DOC
Using the Timer4 CCP module	AN_O0335.DOC
Using the Timer5 CCP module	AN_O0336.DOC

Library:

Title	Application Note Series Number
SPMC65X series software basics manual	AN_O0100.DOC
SPMC65X series software advances 1 manual (data processing operation)	AN_O0101.DOC
SPMC65X series software advances 2 manual (mathematical operation)	AN_O0102.DOC

11.7 Revision History

Revision	Date	Remark
V0.1	03/31/2005	First edition

12 PWM

12.1 Description

In SPMC65X family, as many as 6 channels of Timer are equipped and each one can be set to operate as PWM (pulse width modulation) function. In PWM mode, timers can be divided into three types. For example, Timer 0/2/4 can only output up to an 8-bit PWM resolution, Timer 1/3 can output up to a 12-bit PWM resolution and Timer5 can output up to a 16-bit PWM resolution. Table 12-1 shows the summary of PWM function for SPMC65X family. PWM function can be easily configured with corresponding control registers setting.

The behavior of PWM is that output square waveform which duty and period can be modified by setting period and duty registers. Table 12-2 shows the relation of resolution and frequency. It supports user to choose PWM frequency with suited resolution. The detailed PWM output waveforms are shown in Figure 12-2 and detailed registers setting are described in Section12.2.

	Timer/Event Counter		Capture		Compare		PWM
	8 bit	16 bit	8 bit	16 bit	8 bit	16 bit	
Timer 0	YES	YES	Width/Cycle	Width only	YES	YES	8 bit
Timer 1	YES	YES	Width/Cycle	Width only	YES	YES	12 bit
Timer 2	YES	YES	Width/Cycle	Width only	YES	YES	8 bit
Timer 3	YES	YES	Width/Cycle	Width only	YES	YES	12 bit
Timer 4	YES	YES	Width/Cycle	Width only	YES	YES	8 bit
Timer 5	YES	YES	Width/Cycle	Width/Cycle	YES	YES	16 bit

Table 12-1 Summary of PWM function for SPMC65X family

The timers have three types depending on different features:

1. Type I: (Timer0, Timer2 & Timer4)
 - 8 or 16-bit timer/counter
 - 8 or 16-bit capture mode (8-bit with width/cycle measurement, 16-bit with width measurement)
 - 8 or 16-bit compare mode
 - **8-bit PWM mode**
2. Type II: (Timer1 & Timer3)
 - 8 or 16-bit timer/counter
 - 8 or 16-bit capture mode (8-bit with width/cycle measurement, 16-bit with width measurement)
 - 8 or 16-bit compare mode
 - **12-bit PWM mode**
3. Type III: (Timer5)
 - 8 or 16-bit timer/counter

- 8 or 16-bit capture mode (8 or 16-bit with width/cycle measurement)
- 8 or 16-bit compare mode
- **16-bit PWM mode**

Resolution	PWM Frequency			
	Pre-Scale= $F_{sys} \div 1$	Pre-Scale $= F_{sys} \div 2$	Pre-Scale $= F_{sys} \div 4$	Pre-Scale $= F_{sys} \div 8$
16-Bit Operation	122HZ	61HZ	30.5 HZ	15.25 HZ
12-Bit Operation	1.95KHZ	975HZ	487HZ	243.5HZ
11-Bit Operation	3.9 KHZ	1.95KHZ	975HZ	487HZ
10-Bit Operation	7.8 KHZ	3.9 KHZ	1.95KHZ	975HZ
9-Bit Operation	15.6 KHZ	7.8 KHZ	3.9 KHZ	1.95KHZ
8-Bit Operation	31.2 KHZ	15.6 KHZ	7.8 KHZ	3.9 KHZ

Note: System operation at 8MHz

Table 12-2 Resolution VS PWM Frequency

12.2 Control Register

12.2.1 Timer0_1 Control Register 0 (P_TMR0_1_Ctrl0, \$11) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
-	T1FS2	T1FS1	T1FS0	-	T0FS2	T0FS1	T0FS0
-	R/W	R/W	R/W	-	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit 7 Reserved

Bit [6:4] **T1FS[2 : 0]**: Timer 1 function configuration bits

111 = 12-bit PWM

110 = 16-bit capture (Width)

101 = 16-bit Compare

100 = 16-bit Timer

011 = 8-bit Capture (Width, Cycle)

010 = 8-bit Compare

001 = 8-bit Timer

000 = Disable

Bit 3 Reserved

Bit [2:0] **T0FS[2 : 0]**: Timer0 Function Configuration bits

111 = 8-bit PWM

110 = 16-bit capture (Width)

101 = 16-bit Compare

100 = 16-bit Timer

011 = 8-bit Capture (Width, Cycle)

010 = 8-bit Compare

001 = 8-bit Timer

000 = Disable

12.2.2 Timer0_1 Control Register 1 (P_TMR0_1_Ctrl1, \$12) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
-	T1PSS2	T1PSS1	T1PSS0	-	T0PSS2	T0PSS1	T0PSS0
-	R/W	R/W	R/W	-	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit 7 Reserved

Bit [6:4] **T1PSS[2 : 0]**: Timer1 Pre-Scale Configuration bits

111 = External Event

110 = $F_{SYS} \div 512$

101 = $F_{SYS} \div 128$

100 = $F_{SYS} \div 32$

011 = $F_{SYS} \div 8$

010 = $F_{SYS} \div 4$

001 = $F_{SYS} \div 2$

000 = F_{SYS}

Bit 3 Reserved

Bit [2:0] **T0PSS[2 : 0]**: Timer0 Pre-Scale Configuration bits

111 = External Event

110 = $F_{SYS} \div 512$

101 = $F_{SYS} \div 128$

100 = $F_{SYS} \div 32$

011 = $F_{SYS} \div 8$

010 = $F_{SYS} \div 4$

001 = $F_{SYS} \div 2$

000 = F_{SYS}

F_{SYS} : Frequency of Clock Source

12.2.3 Timer0 PWM Period Register (P_TMR0_PWMPeriod, \$13) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
T0PP7	T0PP6	T0PP5	T0PP4	T0PP3	T0PP2	T0PP1	T0PP0
R/W							
0	0	0	0	0	0	0	0

Bit [7:0]: **T0PP** [7:0]: Timer0 PWM Period Value T0PP [7:0] (8-bit PWM mode)

12.2.4 Timer0 PWM Duty Register (P_TMR0_PWMduty, \$14) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
T0PD7	T0PD6	T0PD5	T0PD4	T0PD3	T0PD2	T0PD1	T0PD0
R/W							
0	0	0	0	0	0	0	0

Bit [7:0] **T0PD** [7:0]: Timer0 PWM Duty Value T0PD [7:0] (8-bit PWM mode)

8-bit PWM mode:

Write P_TMR0_PWMduty first then P_TMR0_PWMPeriod when enabling 8-bit PWM mode.

12.2.5 Timer1 PWM Period Register (P_TMR1_PWMPeriod, \$15) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
T1PP7	T1PP6	T1PP5	T1PP4	T1PP3	T1PP2	T1PP1	T1PP0
R/W							
0	0	0	0	0	0	0	0

Bit [7:0] **T1PP** [7:0]: Timer1 PWM Period Value T1PP [7:0] (12-bit PWM mode)

Note: Writing dataflow: Write high byte first then low byte.

12-bit PWM mode:

Write P_TMR1_DutyPeriod first then P_TMR1_PWMPeriod when updating Period.

Write P_TMR1_DutyPeriod first then P_TMR1_PWMduty when updating Duty.

12.2.6 Timer1 PWM Duty Period Register (P_TMR1_DutyPeriod, \$16) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
T1PD11	T1PD10	T1PD9	T1PD8	T1PP11	T1PP10	T1PP9	T1PP8
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit [7:4] **T1PD** [11:8]: Timer1 PWM Duty Value T1PD [11:8] (12-bit PWM mode)

Bit [3:0] **T1PP** [11:8]: Timer1 PWM Period Value T1PP [11:8] (12-bit PWM mode)

12.2.7 Timer1 PWM Duty Register (P_TMR1_PWMduty, \$17) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
T1PD7	T1PD6	T1PD5	T1PD4	T1PD3	T1PD2	T1PD1	T1PD0
R/W							
0	0	0	0	0	0	0	0

Bit [7:0] **T1PD** [7:0]: Timer1 PWM Duty Value T1PD [7:0] (12-bit PWM mode)

12.2.8 Timer2_3 Control Register 0 (P_TMR2_3_Ctrl0, \$18) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
-	T3FS2	T3FS1	T3FS0	-	T2FS2	T2FS1	T2FS0
-	R/W	R/W	R/W	-	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit 7 Reserved

Bit [6:4] **T3FS**[2 : 0]: Timer 3 function configuration bits

111 = 12-bit PWM

110 = 16-bit capture (Width)

101 = 16-bit Compare

100 = 16-bit Timer

011 = 8-bit Capture (Width, Cycle)

010 = 8-bit Compare

001 = 8-bit Timer

000 = Disable

Bit 3 Reserved

Bit [2:0] **T2FS**[2 : 0]: Timer2 Function Configuration bits

111 = 8-bit PWM

110 = 16-bit capture (Width)

101 = 16-bit Compare

100 = 16-bit Timer

011 = 8-bit Capture (Width, Cycle)

010 = 8-bit Compare

001 = 8-bit Timer

000 = Disable

12.2.9 Timer2_3 Control Register 1 (P_TMR2_3_Ctrl1, \$19) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
-	T3PSS2	T3PSS1	T3PSS0	-	T2PSS2	T2PSS1	T2PSS0
-	R/W	R/W	R/W	-	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit 7 Reserved

Bit [6:4] **T3PSS[2 : 0]**: Timer3 Pre-Scale Configuration bits

111 = External Event

110 = $F_{SYS} \div 512$

101 = $F_{SYS} \div 128$

100 = $F_{SYS} \div 32$

011 = $F_{SYS} \div 8$

010 = $F_{SYS} \div 4$

001 = $F_{SYS} \div 2$

000 = F_{SYS}

Bit 3 Reserved

Bit [2:0] **T2PSS[2 : 0]**: Timer2 Pre-Scale Configuration bits

111 = External Event

110 = $F_{SYS} \div 512$

101 = $F_{SYS} \div 128$

100 = $F_{SYS} \div 32$

011 = $F_{SYS} \div 8$

010 = $F_{SYS} \div 4$

001 = $F_{SYS} \div 2$

000 = F_{SYS}

F_{SYS} : Frequency of Clock Source

12.2.10 Timer2 PWM Period Register (P_TMR2_PWMPeriod, \$1A) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
T2PP7	T2PP6	T2PP5	T2PP4	T2PP3	T2PP2	T2PP1	T2PP0
R/W							
0	0	0	0	0	0	0	0

Bit [7:0] **T2PP** [7:0]: Timer2 PWM Period Value T2PP [7:0] (8-bit PWM mode)

12.2.11 Timer2 PWM Duty Register (P_TMR2_PWMDDuty, \$1B) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
T2PD7	T2PD6	T2PD5	T2PD4	T2PD3	T2PD2	T2PD1	T2PD0
R/W							
0	0	0	0	0	0	0	0

Bit [7:0] **T2PD** [7:0]: Timer2 PWM Duty Value T2PD [7:0] (8-bit PWM mode)

8-bit PWM mode:

Write P_TMR2_PWMDDuty first then P_TMR2_PWMPeriod when enabling 8-bit PWM mode.

12.2.12 Timer3 PWM Period Register (P_TMR3_PWMPeriod, \$1C) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
T3PP7	T3PP6	T3PP5	T3PP4	T3PP3	T3PP2	T3PP1	T3PP0
R/W							
0	0	0	0	0	0	0	0

Bit [7:0] **T3PP** [7:0]: Timer3 PWM Period Value T3PP [7:0] (12-bit PWM mode)

Note: Writing dataflow: Write high byte first then low byte.

12-bit PWM mode:

Write P_TMR3_DutyPeriod first then P_TMR3_PWMPeriod when updating Period.

Write P_TMR3_DutyPeriod first then P_TMR3_PWMDDuty when updating Duty.

12.2.13 Timer3 PWM Duty Period Register (P_TMR3_DutyPeriod, \$1D) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
T3PD11	T3PD10	T3PD9	T3PD8	T3PP11	T3PP10	T3PP9	T3PP8
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit [7:4] **T3PD** [11:8]: Timer3 PWM Duty Value T3PD [11:8] (12-bit PWM mode)

Bit [3:0] **T3PP** [11:8]: Timer3 PWM Period Value T3PP [11:8] (12-bit PWM mode)

12.2.14 Timer3 PWM Duty Register (P_TMR3_PWMDDuty, \$1E) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
T3PD7	T3PD6	T3PD5	T3PD4	T3PD3	T3PD2	T3PD1	T3PD0
R/W							
0	0	0	0	0	0	0	0

Bit [7:0] **T3PD** [7:0]: Timer3 PWM Duty Value T3PD [7:0] (12-bit PWM mode)

12.2.15 Timer4_5 Control Register 0 (P_TMR4_5_Ctrl0, \$1F) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
-	T5FS2	T5FS1	T5FS0	-	T4FS2	T4FS1	T4FS0
-	R/W	R/W	R/W	-	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit 7 Reserved

Bit [6:4] **T5FS[2 : 0]: Timer 5 function configuration bits**

111 = 16-bit PWM

110 = 16-bit capture (Width)

101 = 16-bit Compare

100 = 16-bit Timer

011 = 8-bit Capture (Width, Cycle)

010 = 8-bit Compare

001 = 8-bit Timer

000 = Disable

Bit 3 Reserved

Bit [2:0] **T4FS[2 : 0]: Timer4 Function Configuration bits**

111 = 8-bit PWM

110 = 16-bit capture (Width)

101 = 16-bit Compare

100 = 16-bit Timer

011 = 8-bit Capture (Width, Cycle)

010 = 8-bit Compare

001 = 8-bit Timer

000 = Disable

12.2.16 Timer4_5 Control Register 1 (P_TMR4_5_Ctrl1, \$20) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
-	T5PSS2	T5PSS1	T5PSS0	-	T4PSS2	T4PSS1	T4PSS0
-	R/W	R/W	R/W	-	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit 7 Reserved

Bit [6:4] **T5PSS[2 : 0]: Timer5 Pre-Scale Configuration bits**

111 = External Event

110 = $F_{SYS} \div 512$

101 = $F_{SYS} \div 128$

100 = $F_{SYS} \div 32$

011 = $F_{SYS} \div 8$

$010 = F_{SYS} \div 4$

$001 = F_{SYS} \div 2$

$000 = F_{SYS}$

Bit 3 Reserved

Bit [2:0] **T4PSS[2 : 0]**: Timer4 Pre-Scale Configuration bits

111 = External Event

$110 = F_{SYS} \div 512$

$101 = F_{SYS} \div 128$

$100 = F_{SYS} \div 32$

$011 = F_{SYS} \div 8$

$010 = F_{SYS} \div 4$

$001 = F_{SYS} \div 2$

$000 = F_{SYS}$

F_{SYS} : Frequency of Clock Source

12.2.17 Timer4 PWM Period Register (P_TMR4_PWMPeriod, \$21) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
T4PP7	T4PP6	T4PP5	T4PP4	T4PP3	T4PP2	T4PP1	T4PP0
R/W							
0	0	0	0	0	0	0	0

Bit [7:0] **T4PP** [7:0]: Timer4 PWM Period Value T4PP [7:0] (8-bit PWM mode)

12.2.18 Timer4 PWM Duty Register (P_TMR4_PWMDDuty, \$22) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
T4PD7	T4PD6	T4PD5	T4PD4	T4PD3	T4PD2	T4PD1	T4PD0
R/W							
0	0	0	0	0	0	0	0

Bit [7:0] **T4PD** [7:0]: Timer4 PWM Duty Value T4PD [7:0] (8-bit PWM mode)

8-bit PWM mode:

Write P_TMR4_PWMDDuty first then P_TMR4_PWMPeriod when enabling 8-bit PWM mode.

12.2.19 Timer5 PWM Period Low Byte Register (P_TMR5_PWMPeriodLo, \$23) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
T5PP7	T5PP6	T5PP5	T5PP4	T5PP3	T5PP2	T5PP1	T5PP0
R/W							
0	0	0	0	0	0	0	0

Bit [7:0] **T5PP** [7:0]: Timer5 PWM Period Value T5PP [7:0] (16-bit PWM mode)

Note: Writing dataflow: Write high byte first then low byte.

16-bit PWM mode:

Write P_TMR5_PWMPeriodHi first then P_TMR5_PWMPeriodLo when updating Period.

Write P_TMR5_PWMPeriodHi first then P_TMR5_PWMPeriodLo when updating Duty.

12.2.20 Timer5 PWM Period High Byte Register (P_TMR5_PWMPeriodHi, \$24) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
T5PP15	T5PP14	T5PP13	T5PP12	T5PP11	T5PP10	T5PP9	T5PP8
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit [7:0] **T5PP** [15:8]: Timer5 PWM Period Value T5PD [15:8] (16-bit PWM mode)

12.2.21 Timer5 PWM Duty Low Byte Register (P_TMR5_PWMPeriodLo, \$25) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
T5PD7	T5PD6	T5PD5	T5PD4	T5PD3	T5PD2	T5PD1	T5PD0
R/W							
0	0	0	0	0	0	0	0

Bit [7:0] **T5PD** [7:0]: Timer5 PWM Duty Value T5PD [7:0] (16-bit PWM mode)

12.2.22 Timer5 PWM Duty High Byte Register (P_TMR5_PWMPeriodHi, \$5F) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
T5PD15	T5PD14	T5PD13	T5PD12	T5PD11	T5PD10	T5PD9	T5PD8
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit [7:0] **T5PD** [15:8]: Timer5 PWM Duty Value T5PD [15:8] (16-bit PWM mode)

12.3 Operation

12.3.1 8-bit PWM (Type I)

In SPMC65X family, Timer0, Timer2 and Timer4 belong to Type I – 8-bit PWM (Pulse Width Modulation). When TxFS[2:0] setting is 111, Timer x acts as an 8-bit PWM generator, and corresponding pin (e.g. Timer0 \rightarrow PB2) automatically become up to an 8-bit resolution PWM output. The period of the PWM is set at “Timer x PWM Period Register”, P_TMRx_PWMPeriod and the duty of the PWM is set at “Timer x PWM Duty Register”, P_TMRx_PWMPeriod. In the 8-bit PWM mode, the user must write P_TMRx_PWMPeriod first then P_TMRx_PWMPeriod when enabling 8-bit PWM function. The counting clock source is selected at TxPSS[2:0]. Counting is started from P_TMRx_PWMPeriod that is defined PWM period and corresponding pin (e.g. Timer0 \rightarrow PB2) output low level. The counter keeps up counting and its values are compared with P_TMRx_PWMPeriod which is defined PWM Duty. If a match is found, its output toggles to high level. Next, The counter keeps up counting until Timer x overflow. Every time Timer x overflow, the counter is updated with new P_TMRx_PWMPeriod and P_TMRx_PWMPeriod and next PWM output sequence are generated. Figure 12-1 shows a block diagram of 8-bit PWM and PWM period and duty can be calculated using the Equation 12-1. For a step-by-step procedure on how to set up the PWM Period interrupt, see Section12.4 in detail.

If

PRDREG= P_T0DATA0[7:0]
 DUTYREG=T0DATA1[7:0]
 T_TMRx= Timer x Clock Period after pre-scaling

Then

PWM PERIOD=[\\$100H-PRDREG]*T_TMRx
 PWM DUTY=[\\$FFH-DUTYREG]*T_TMRx

Note: x = 0, 2, 4

Equation 12-1:Calculation for 8-bit PWM Period and Duty

Detailed relationship between register setting and real value are shown in Table 12-3.

PRDREG (Hex)	PWM PERIOD (* T_TMRx)		DUTYREG (Hex)	PWM DUTY(* T_TMRx)	
	Hex	Decimal		Hex	Decimal
\$00	\$100	256	\$00	\$FF	255
\$01	\$FF	255	\$01	\$FE	254
.....
\$FE	\$02	2	\$FE	\$01	1
\$FF	\$01	1	\$FF	\$00	0

Table 12-3 The relationship between register setting and real value for 8-bit PWM operation

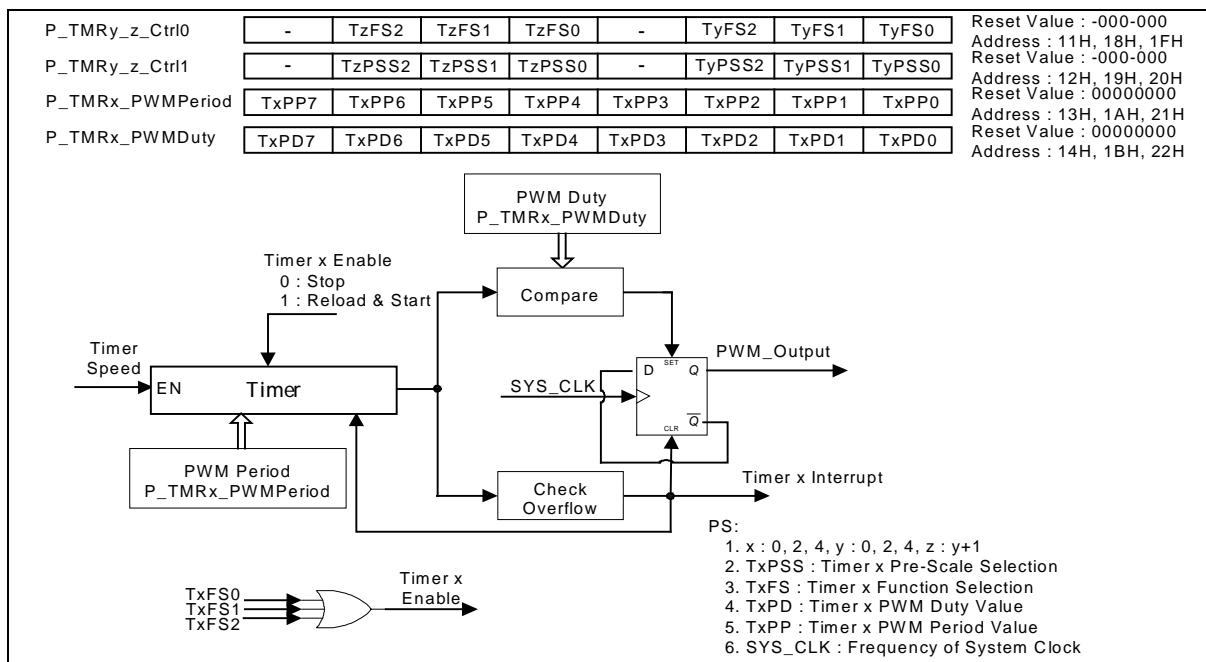


Figure 12-1 8-bit PWM Block Diagram

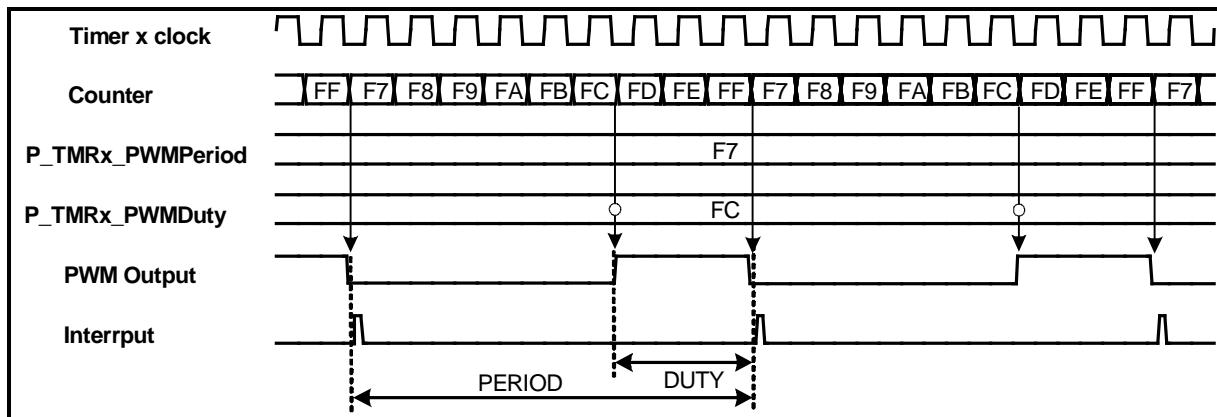


Figure 12-2 8-bit PWM Operation

【Example 12-1】: Set Timer0 as 8-bit PWM operation.

```

    lda #178                      ;Before starting timer, set duty and period initial value first
    sta P_TMR0_PWMDDuty           ;Set duty value
    lda #$00
    sta P_TMR0_PWMPeriod          ;Set period value
    lda #C_T0FCS_Div_8            ; Set Timer0 clock source is Fcs/8
    sta P_TMR0_1_Ctrl1
    lda #C_T08B_PWM                ; Set Timer0 is 8-bit PWM
    sta P_TMR0_1_Ctrl0
    lda #178
    sta P_TMR0_PWMDDuty           ; duty ratio = (255 - 178) / 255 ~= 30% on PB2

```

```
lda      #$00
sta      P_TMR0_PWMPeriod           ; PWM frequency = 8.0MHz / (255 x 8) = 3.9 KHz
```

12.3.2 12-bit PWM (Type II)

In SPMC65X family, Timer1 and Timer3 belong to Type II – 12-bit PWM (Pulse Width Modulation). When TxFS [2:0] setting is 111, Timer x acts as a 12-bit PWM generator, and a corresponding pin (e.g. Timer1 = PB3) automatically become up to an 12-bit resolution PWM output. The period of the PWM is set at “Timer x PWM Period Register”, P_TMRx_PWMPeriod and “Timer x PWM Duty Period Register”, P_TMRx_DutyPeriod [3:0]; moreover, the duty of the PWM is set at “Timer x PWM Duty Period Register”, P_TMRx_DutyPeriod [7:4] and “Timer x PWM Duty Register”, P_TMRx_PWMduty. The counting clock source is selected at TxPSS [2:0]. Counting is started from {P_TMRx_DutyPeriod [3:0], P_TMRx_PWMPeriod [7:0]} that is defined PWM period and corresponding output (e.g. Timer1 = PB3) low level. The counter keeps up counting and its contents are compared with {P_TMRx_DutyPeriod [7:4], P_TMRx_PWMduty [7:0]} which is defined PWM Duty. If a match is found, its corresponding output toggles to high level. Next, The counter keeps up counting until Timer x overflow. Every time Timer x overflow, the counter is updated with new duty and period setting, and next PWM output sequence is generated.

Since this chip equips with 8-bit CPU and data bus width is 8-bit, the user can't access 16-bit data simultaneously; hence, in the 12-bit PWM mode, the user must write the P_TMRx_DutyPeriod (MSB byte) first that will be buffered automatically and then the P_TMRx_PWMPeriod (period) / P_TMRx_PWMduty (duty) that is LSB byte. This buffered value remains unchanged until the 12-bit write sequence is completed and the buffered MSB byte will download into the Timer x with LSB byte simultaneously. Figure 12-3 shows a block diagram of 12-bit PWM and PWM period and duty can be calculated using the Equation 12-2. Operating waveforms are shown in Figure 12-4, Figure 12-5 and Figure 12-6.

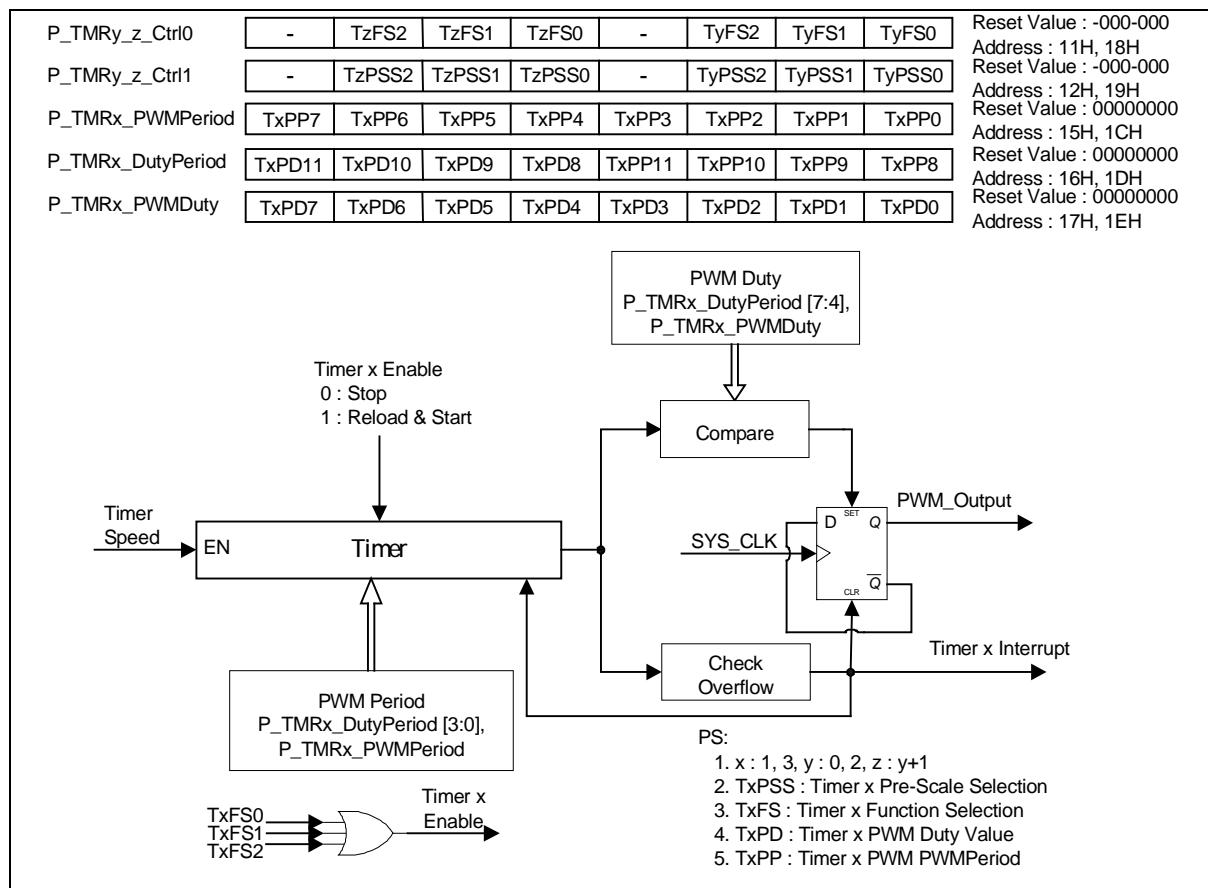


Figure 12-3 12-bit PWM Block Diagram

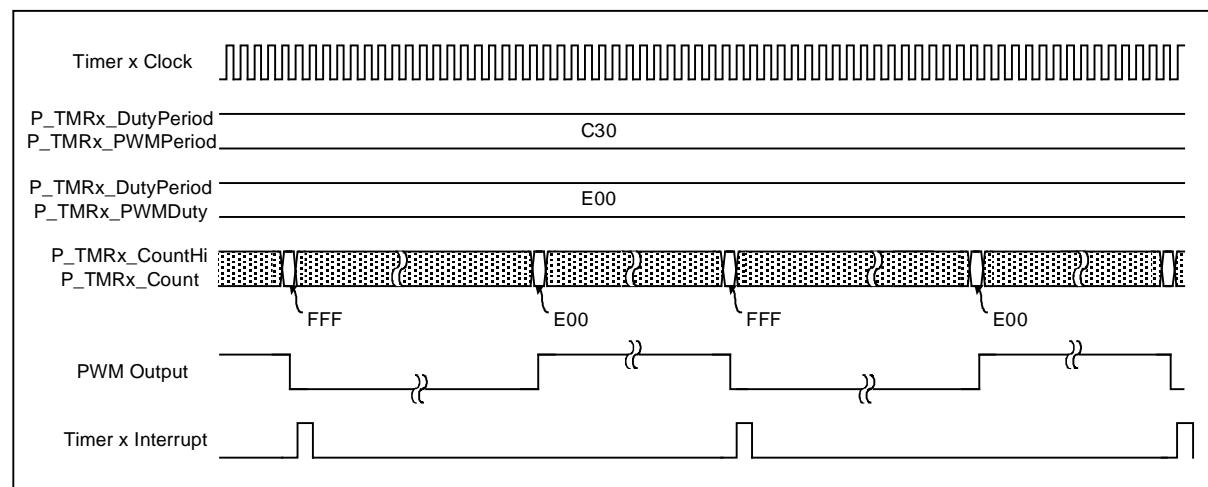


Figure 12-4 12-Bit PWM Operating Waveform

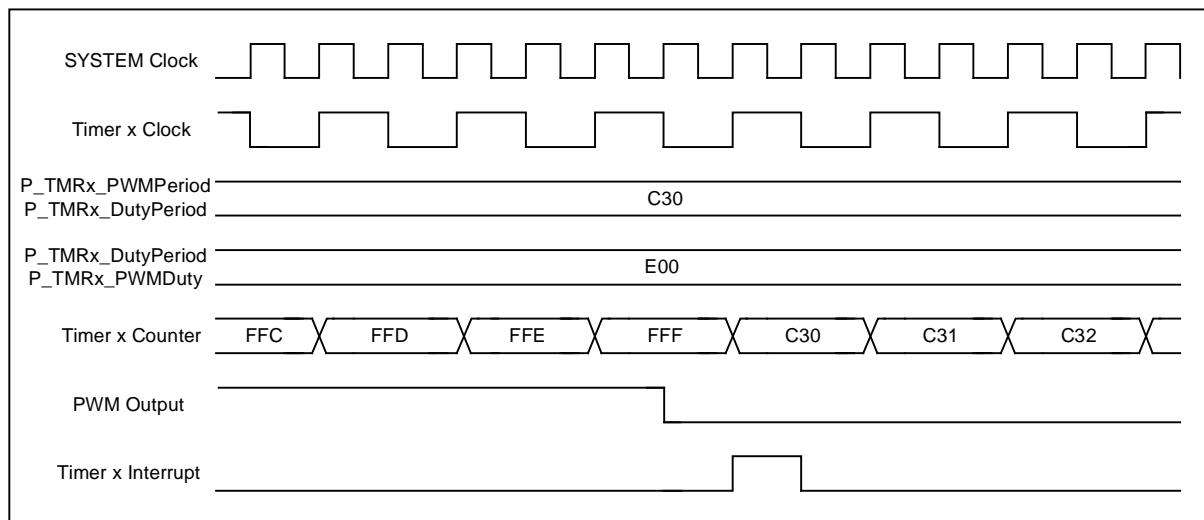


Figure 12-5 12-Bit PWM reload waveform

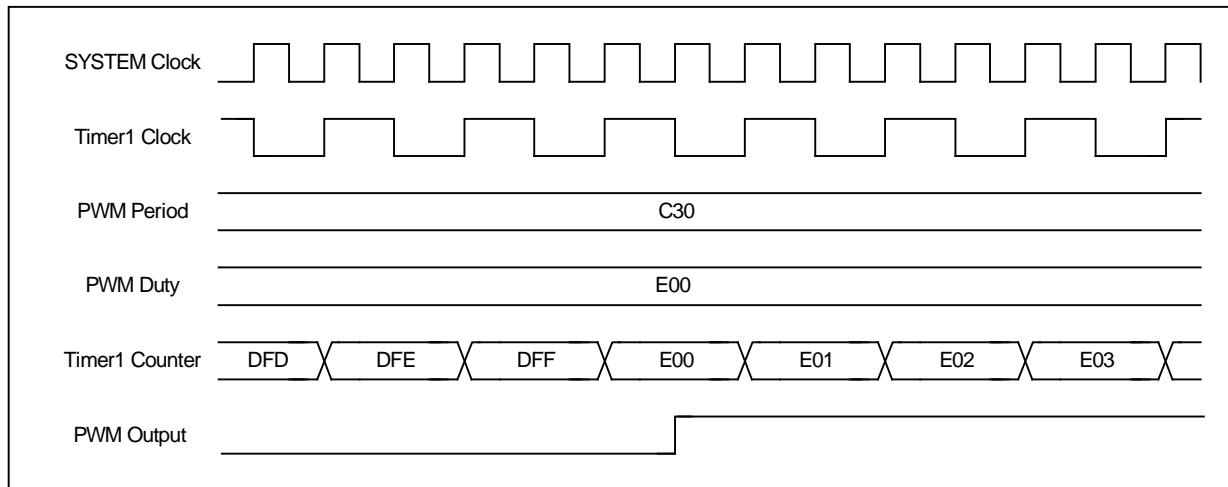


Figure 12-6 12-Bit PWM transient waveform

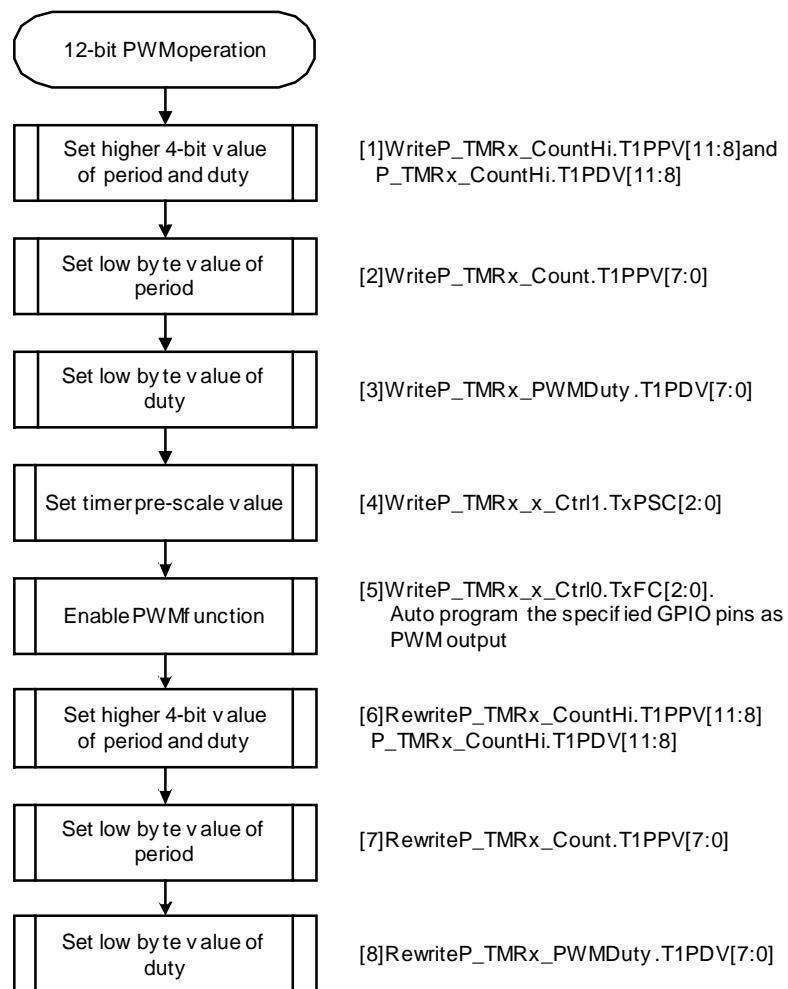


Figure 12-7 12-Bit PWM software flowchart

```

If
    PRDREG={P_TMRx_DuyPeriod [3:0], P_TMRx_PWMPeriod [7:0]}
    DUTYREG={P_TMRx_DuyPeriod [7:4], P_TMRx_PWMPer [7:0]}
    T_TMRx = Timer x Clock Period after pre-scaling
Then
    PWM PERIOD=[$1000-PRDREG]* T_TMRx
    PWM DUTY=[$FFFH-DUTYREG]* T_TMRx
Note: x = 1, 3
  
```

Equation 12-2 Calculation for 12-bit PWM Period and Duty

Detailed relationship between register setting and real value are shown in Table 12-4.

PRDREG (Hex)	PWM PERIOD (*T_TMRx)		DUTYREG (Hex)	PWM DUTY (*T_TMRx)	
	Hex	Decimal		Hex	Decimal

\$000	\$1000	4096	\$000	\$FFF	4095
\$001	\$FFF	4095	\$001	\$FFE	4094
.....
\$FFE	\$002	2	\$FFE	\$001	1
\$FFF	\$001	1	\$FFF	\$000	0

Table 12-4 The relationship between register setting and real value for 12-bit PWM operation

【Example 12-2】 : Set Timer1 as 12bit PWM operation.

lda	#\$70	; Before starting timer, set duty and period initial value first
sta	P_TMR1_DutyPeriod	; Set duty value
lda	#\$00	
sta	P_TMR1_PWMPeriod	; Set Period value
lda	#\$FF	
sta	P_TMR1_PWMDDuty	;
lda	#C_T1FCS_Div_8	; Set Timer1 clock source is Fsys/8
sta	P_TMR0_1_Ctrl1	
lda	#C_T112B_PWM	; Set Timer1 is 12-bit PWM
sta	P_TMR0_1_Ctrl0	
lda	#\$70	
sta	P_TMR1_DutyPeriod	; PWM duty \$7FF, PWM period \$1000
lda	#\$00	
sta	P_TMR1_PWMPeriod	
lda	#\$FF	
sta	P_TMR1_PWMDDuty	; PWM output 244Hz 50% duty ratio on PB3

12.3.3 16-bit PWM (Type III)

In SPMC65X family, Timer5 belong to Type III – 16-bit PWM (Pulse Width Modulation). When T5FC[2:0] setting is 111, Timer5 acts as an 16-bit PWM waveform generator, and pin PD7 automatically become up to an 16-bit resolution PWM output. The period of the PWM is set at “Timer5 PWM Period Low Register” – P_TMR5_PWMPeriodLo (\$23) and “Timer5 PWM Period High Register” – P_TMR5_PWMPeriodHi (\$24); moreover, the duty of the PWM is set at the “Timer5 PWM Duty Low Register” – P_TMR5_PWMDDutyLo (\$25) and “Timer5 PWM Duty High Register” – P_TMR5_PWMDDutyHi (\$5F). The counting clock source is selected at T5PSS [2:0]. Counting is started from { P_TMR5_PWMPeriodHi, P_TMR5_PWMPeriodLo} value that is defined PWM period and PD7 output low level. The counter keeps up counting and its values are compared with { P_TMR5_PWMDDutyHi, P_TMR5_PWMDDutyLo} that is defined PWM Duty. If a match is found, its PD7 toggles output to high level. Next, The counter keeps up counting until timer5 overflow. Every time timer5 overflow, the

counter value is updated with new duty and period setting, and next PWM output sequence are generated. Since this chip equips with 8-bit CPU and data bus width is 8-bit, the user can't access 16-bit data simultaneously; hence, in the 16-bit PWM mode, the user must write the P_TMR5_PWMPeriodHi (MSB byte) first that will be buffered automatically and then the P_TMR5_PWMPeriodLo that is LSB byte in PWM period setting; moreover, the user must write the P_TMR5_PWMDDutyHi (MSB byte) first that will be buffered automatically and then the P_TMR5_PWMDDutyLo that is LSB byte in PWM duty setting. This buffered value remains unchanged until the 16-bit write sequence is completed and the buffered MSB byte will download into the Timer5 with LSB byte simultaneously. Figure 12-8 shows a block diagram of 16-bit PWM and PWM period and duty can be calculated using the Equation 12-3.

If

```
PRDREG={P_TMR5_PWMPeriodHi, P_TMR5_PWMPeriodLo}
DUTYREG={P_TMR5_PWMDDutyHi, P_TMR5_PWMDDutyLo}
T_TMR5= Timer5 Clock Period after pre-scaling
```

Then

```
PWM PERIOD=[$10000H-PRDREG]* T_TMR5
PWM DUTY=[$FFFF-DUTYREG]* T_TMR5
```

Equation 12-3:Calculation for 16-bit PWM Period and Duty

Detail relationship between register setting and real value are shown in Table 12-5.

PRDREG (Hex)	PWM PERIOD (*T_TMR5)		DUTYREG (Hex)	PWM DUTY (*T_MR5)	
	Hex	Decimal		Hex	Decimal
\$0000	\$10000	65536	\$0000	\$FFFF	65535
\$0001	\$FFFF	65535	\$0001	\$FFFE	65534
.....
\$FFFE	\$0002	2	\$FFFE	\$0001	1
\$FFFF	\$0001	1	\$FFFF	\$0000	0

Table 12-5 The relationship between register setting and real value for 12-bit PWM operation

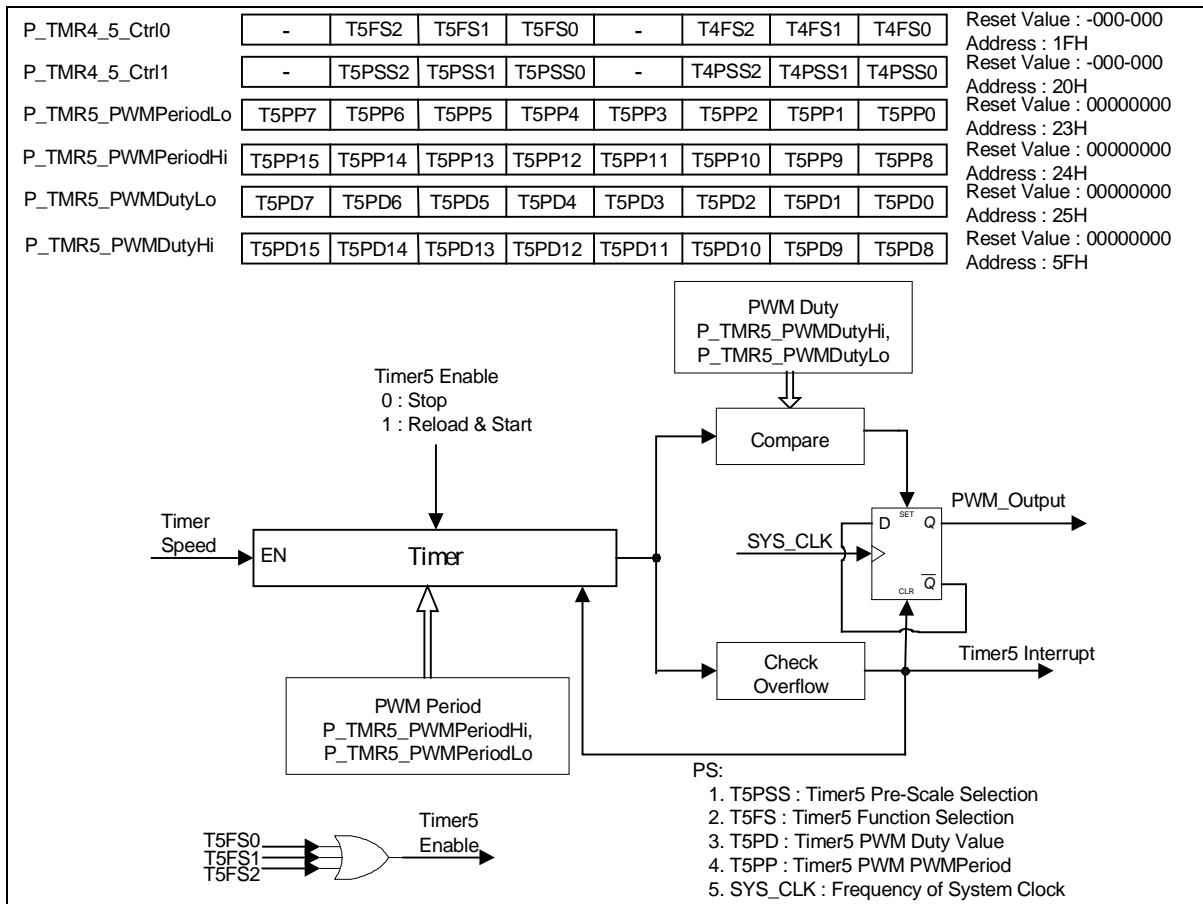


Figure 12-8 16-bit PWM Block Diagram

【Example 12-3】: Set Timer5 as 16bit PWM operation.

```

lda    #$A0                      ; Before starting timer, set duty and period initial value first
sta    P_TMR5_PWMDDutyHi        ; Set high byte duty value
lda    #$00
sta    P_TMR5_PWMDDutyLo        ; Set low byte duty value
lda    #$80
sta    P_TMR5_PWMPeriodHi       ; Set high byte period value
lda    #$00
sta    P_TMR5_PWMPeriodLo       ; Set low byte period value
lda    #C_T5FCS_Div_1          ; Set Timer5 clock source is Fcs/1
sta    P_TMR4_5_Ctrl1
lda    #C_T516B_PWM             ; Set Timer5 is 16-bit PWM
sta    P_TMR4_5_Ctrl0
lda    #$A0
sta    P_TMR5_PWMDDutyHi
lda    #$00
sta    P_TMR5_PWMDDutyLo        ; duty ratio = ($FFFF - $A000) / ($FFFF - $8000) ~= 75% on PD7
lda    #$80

```

```
sta      P_TMR5_PWMPeriodHi  
lda      #$00  
sta      P_TMR5_PWMPeriodLo      ; PWM frequency = 8.0MHz / ($FFFF - $8000) ~= 244 Hz
```

12.4 PWM Interrupt

Interrupts can come from many sources. There are five interrupt sources related with PWM.

- | Timer0 PWM Period Interrupt
- | Timer1 PWM Period Interrupt
- | Timer2 PWM Period Interrupt
- | Timer3 PWM Period Interrupt
- | Timer4 PWM Period Interrupt
- | Timer5 PWM Period Interrupt

There are several steps for enabling Timer PWM Period interrupt.

1. Interrupt Disable flag (I-flag) is set to “1” by using “SEI ” instruction.
2. Decided 8-bit, 12-bit or 16-bit resolution and then selected a suitable timer for use.
3. Configured some registers related for PWM operation in the timer x(x = 0~5) you selected.
4. Enable corresponding timer x interrupt flag in the P_INT_Ctrl1(\$0F) register.
5. Using ‘CLI’ instruction to enable all interrupt function.
6. Now PWM period interrupt will be accepted.

12.5 Design Tips

[Example 12-4] : Shows how to initialize the Timer1 as 12 bit PWM operation based on SPMC65P2404A. Please refer to an example of Fortis IDE.

- n Use Timer1 12-bit PWM
- n Set Timer1 clock source as Fcs/8
- n PWM frequency is 244 Hz
- n Duty ratio is 75% on PB3

```

.SYNTAX 6502           ; process standard 6502 addressing syntax
.LINKLIST            ; generate linklist information
.SYMBOLS              ; generate symbolic debug information

.INCLUDE      spmc65p2404a.inc

.PAGE0          ; define values in the range from $00 to $FF
;
.DATA           ; define data storage section
;
.CODE

; *****
;*
;* Power on Reset Process - Main Program
;*
;****

V_Reset:
    sei           ; Disable interrupt
    ldx      #C_STACK_BOTTOM ; Initial stack pointer at $00FF
    txs           ; Transfer to stack point
;
    lda      #$FF           ; Clear Reset flag
    sta      P_SYS_Ctrl
    sta      P_SYS_Ctrl
;
    jsr      F_InitIOPort ; initial GPIO port;
    lda      #$30           ; high nibble = bit 11-8 of 12-bit PWM
    sta      P_TMR1_DutyPeriod
    lda      #$00
    sta      P_TMR1_PWMPPeriod ; PWM frequency = 8.0MHz / (4095 x 8) = 244 Hz;
    lda      #$FF           ; low byte of 12-bit PWM
    sta      P_TMR1_PWMDDuty ; duty ratio = (4095 - 1023) / 4095 ~= 75% on PB3
    lda      #C_T1FCS_Div_8 ; Set Timer1 clock source is Fcs/8
    sta      P_TMR0_1_Ctrl1
    lda      #C_T112B_PWM   ; Set Timer1 is 12-bit PWM
    sta      P_TMR0_1_Ctrl0
;
    lda      #$30           ; high nibble = bit 11-8 of 12-bit PWM
    sta      P_TMR1_DutyPeriod
    lda      #$00
    sta      P_TMR1_PWMPPeriod ; PWM frequency = 8.0MHz / (4095 x 8) = 244 Hz
;
    lda      #$FF           ; low byte of 12-bit PWM
    sta      P_TMR1_PWMDDuty ; duty ratio = (4095 - 1023) / 4095 ~= 75% on PB3
;
```

```

;
    lda      #$FF           ; clear INT request flag
    sta      P_INT_Flag0
    sta      P_INT_Flag1
    set      P_INT_Ctrl1, CB_INT_T1OIE ; enable Timer1 overflow interrupt
    cli      ; enable INT
;

L_Main:
    nop
    jmp      L_Main
;

F_InitIOPort:
;

;      /// Initial Port A      ///
;

    lda      #00000000B
    sta      P_IOA_Data
    lda      #0000000B
    sta      P_IOA_Attrib
    lda      #1111111B          ; PortA as input with pull-low
    sta      P_IOA_Dir
;

    rts
;
***** *
.*      *
.*      IRQ Interrupt Service Routine      *
.*      *
;
***** *

V_IRQ:
    pha      ; push A register
    txa      ; transfer X to A
    pha      ; push A register (ie. push X)
;

; Timer 1 overflow interrupt ?

;

    lda      P_INT_Flag1
    and      #C_INT_T1OIF
    beq      L_exit_irq
;

    set      P_INT_Flag1, CB_INT_T1OIF
    lda      #$30              ; re-write higher 4-bit value of period and duty

    sta      P_TMR1_DutyPeriod
    lda      #$00              ; re-write low byte value of period
;
```

```

sta      P_TMR1_PWMPPeriod      ; PWM frequency = 8.0MHz / (4095 x 8) = 244 Hz
lda      #$FF                  ; re-write low byte value of duty
sta      P_TMR1_PWMDuty       ; duty ratio = (4095 - 1023) / 4095 ~= 75% on PB3
;
lda      P_IOA_Data
eor      #$FF
sta      P_IOA_Data           ; toggle P_IOA_Data
;
L_exit_irq:
pla      ; pop A register
tax      ; transfer A to X
pla      ; pop A register (ie. pop X)
;
rti
;
; *****
; *          *
; *      Interrupt Vector Table   *
; *          *
; *****
VECTOR:    .SECTION
DW       V_NMI                 ; may download program emulated either
DW       V_Reset                ; in internal memory or external memory
DW       V_IRQ                 ; dw define two bytes interrupt vector
;
; *****
; *          *
; *      End Of Interrupt Vector Table   *
; *          *
; *****
.END        ; end of program

```

12.6 Related Application Notes and Libraries

This section lists some application notes and libraries about various usage of SPMC65X family. The application notes may help user develop quickly and offer the concepts about the usage of this topic. Sunplus also supports some powerful libraries about general operation, mathematics, communication, etc. The purpose of library is to help user speed up development. The current application notes and libraries related to PWM are listed below.

Application Note:

Title

Application Note Series Number

I/O simulate melody output	AN_O0319.DOC
Using the Timer0 CCP module	AN_O0331.DOC
Using the Timer1 CCP module	AN_O0332.DOC
Using the Timer2 CCP module	AN_O0333.DOC
Using the Timer3 CCP module	AN_O0334.DOC
Using the Timer4 CCP module	AN_O0335.DOC
Using the Timer5 CCP module	AN_O0336.DOC

Library:

Title	Application Note Series Number
SPMC65X series software basics manual	AN_O0100.DOC
SPMC65X series software advances 1 manual (data processing operation)	AN_O0101.DOC
SPMC65X series software advances 2 manual (mathematical operation)	AN_O0102.DOC

12.7 Revision History

Revision	Date	Remark
V0.1	03/31/2005	First edition

13 A/D Converter

13.1 Description

SPMC65X family's devices are built with a maximum 10-bit 9-channel ADC with one top reference voltage. It is used for many applications like touch panel, battery power detection, etc. Figure 13-1 shows A/D converter block diagram. The channel inputs of ADC are shared with PA [7:0], and PB7 that is also shared with top reference voltage.

The analog reference voltage can be selected from external input voltage on PB7 or internal voltage VDD by setting ADVRT bit in P_AD_Ctrl0 register. The ADC module has three registers that are P_AD_Ctrl0, P_AD_Ctrl1 and P_AD_Ctrl2. P_AD_Ctrl0 controls the operation of AD converter module and the pre-scale of AD clock source. Be care to choose AD clock source, it should be lower than 1.4 MHZ to satisfy AD hardware requirement. The processing of conversion is start when the STARTB bit is set to "0". P_AD_Ctrl1 configured PA[7:0] pin as analog pin for AD use or digital pin for I/O use. P_AD_Ctrl2 is used to select current channel for conversion. For a step-by-step procedure on how to set up the A/D converter interrupt, see Section16.3 in detail.

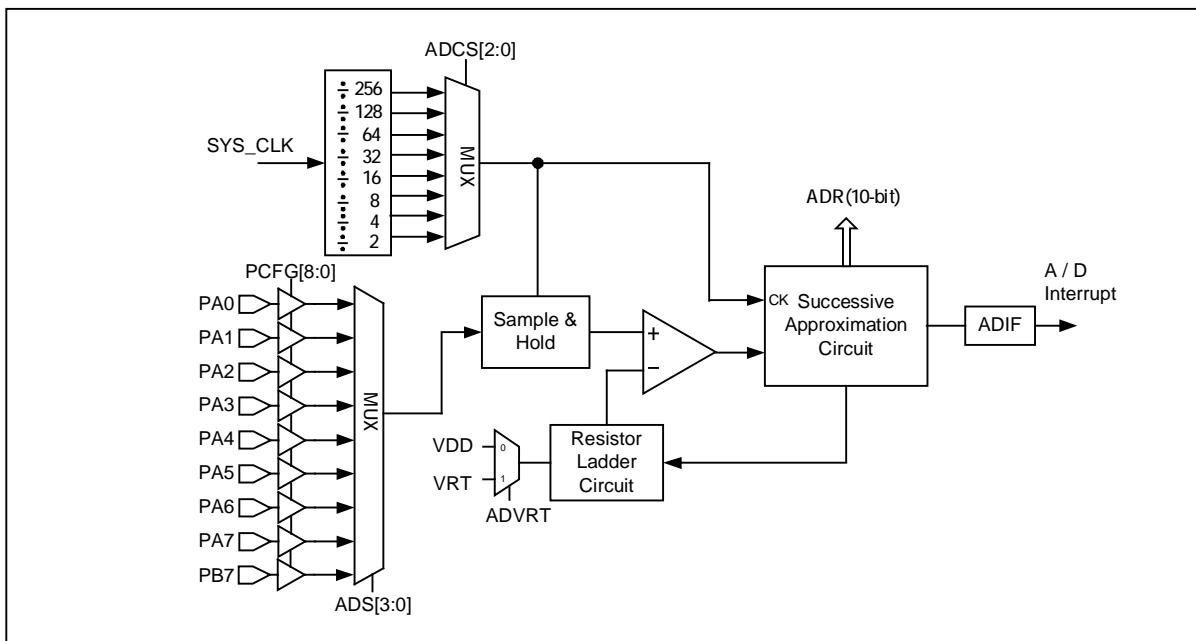


Figure 13-1 AD Converter Block Diagram

13.2 Control Register

13.2.1 ADC Control Register 0 (P_AD_Ctrl0, \$28) (R/W)

	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
R	ADEN	ADVRT	-	-	ADCS2	ADCS1	ADCS0	ADRDY
W	ADEN	ADVRT	-	-	ADCS2	ADCS1	ADCS0	STARTB
	0	0	0	0	0	1	1	0

Bit 7 **ADEN:** ADC Enable bit

0 = ADC function is disabled

1 = ADC function is enabled

Bit 6 **ADVRT:** ADC top reference voltage source selection bit

1 = External voltage (PB7) as top reference voltage

0 = Vdd as top reference voltage

Bit [5:4] Reserved

Bit [3:1] **ADCS[2:0]** : ADC Clock Selection bits

111= $F_{SYS} \div 256$

110= $F_{SYS} \div 128$

101= $F_{SYS} \div 64$

100= $F_{SYS} \div 32$

011= $F_{SYS} \div 16$

010= $F_{SYS} \div 8$

001= $F_{SYS} \div 4$

000= $F_{SYS} \div 2$

F_{SYS} : Frequency of System Clock

Bit 0 **ADRDY/STARTB:** ADC status or start control bit

Read: ADC ready status bit

1: ADC is ready and wait for next conversion.

0: ADC is busy, ie. ADC is converting data.

Write: ADC start bit

1: no effect

0: ADC start conversion

13.2.2 ADC Control Register 1 (P_AD_Ctrl1, \$29) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
PCFG7	PCFG6	PCFG5	PCFG4	PCFG3	PCFG2	PCFG1	PCFG0
R/W							
0	0	0	0	0	0	0	0

Bit [7:0] **PCFG[7:0]:** ADC channel configuration control bits

These bits are used to configure A/D channel as analog pins of digital pins.

PCFG7:

1= Analog input (AN7)

0= Digital input (PA7)

PCFG6:

1= analog input (AN6)

0= Digital input (PA6)

PCFG5:

1= analog input (AN5)

0= Digital input (PA5)

PCFG4:

1= analog input (AN4)

0= Digital input (PA4)

PCFG3:

1= analog input (AN3)

0= Digital input (PA3)

PCFG2:

1= analog input (AN2)

0= Digital input (PA2)

PCFG1:

1= analog input (AN1)

0= Digital input (PA1)

PCFG0:

1= analog input (AN0)

0= Digital input (PA0)

13.2.3 ADC Control Register 2 (P_AD_Ctrl2, \$2A) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
ADCE	ADS3	ADS2	ADS1	ADS0	-	-	PCFG8
R/W	R/W	R/W	R/W	R/W	-	-	R/W
0	0	0	0	0	0	0	0

Bit 7 **ADCE**: ADC Power control bit. This bit is used to enable the bias circuit of AD converter.

Bit [6:3] **ADS[3:0]**:ADC current conversion channel selection bits

0000= Select channel 0 (AN0).

0001= Select channel 1 (AN1)

0010= Select channel 2 (AN2)

0011= Select channel 3 (AN3)

0100= Select channel 4 (AN4)

0101= Select channel 5 (AN5)

0110= Select channel 6 (AN6)

0111= Select channel 7 (AN7)

1000= Select channel8 (AN8)

Otherwise = Reserved

Bit [2:1] Reserved

Bit 0 **PCFG8:**

1= analog input (AN8/Avref)

0= Digital input (PB7)

Note: While setting this bit, user must take care about the ADVRT bit of ADCON0 to make correct setting as AN8 or analog top reference voltage.

13.2.4 ADC Conversion High Data Register (P_AD_DataHi, 2B)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
ADDData9	ADDData8	ADDData7	ADDData6	ADDData5	ADDData4	ADDData3	ADDData2
R/W							
0	0	0	0	0	0	0	0

Bit [7:0] **ADDData [9:2]:**ADC conversion High Data, These bits are the most significant 8 bits of converted 10 bits.

13.2.5 ADC Conversion Low Data Register (P_AD_DataLo, \$2C) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
ADDData1	ADDData0						
R/W	R/W	-	-	-	-	-	-
0	0	0	0	0	0	0	0

Bit [7:0] **ADDData [1:0]:**ADC conversion High Data, These bits are the least significant 2 bits of converted 10 bits

13.3 Operation

In the SPMC65X family's devices, the P_AD_DataHi and P_AD_DataLo contain the 10-bit result of A/D conversion. There are two kinds of methods to get A/D conversion data correctly , such as polling and interrupt methods. By using one of two methods, when AD conversion is complete, the result is loaded into this A/D result register pair (P_AD_DataHi: P_AD_DataLo). If polling method is adopted and after starting conversion, the data of conversion is not loaded into result register until ARDY bit is set.

If A/D interrupt enable bit ADIE in P_INT_Ctrl0 is set, when A/D conversion is complete, the ARDY bit and A/D interrupt flag bit, ADIF will be set, and an A/D converter interrupt will be occurred.

The A/D conversion time per bit is defined as T_{AD} , In the SPMC65X family's devices, the A/D conversion requires 14 T_{AD} per 10-bit conversion. The source of the A/D conversion clock can be selected with eight stages by setting ADCS[2:0] in the P_AD_Ctrl0 register. To let A/D conversion correct, the source of the A/D conversion clock (T_{AD}) should be lower than 1.4MHZ to satisfy AD hardware requirement. Note that it must delay about 5 msec to stable AD power at first time. Figure 13-2 shows the process of AD module operation.. For a step-by-step procedure on how to set up the AD converter interrupt, see Section13.4 in detail.

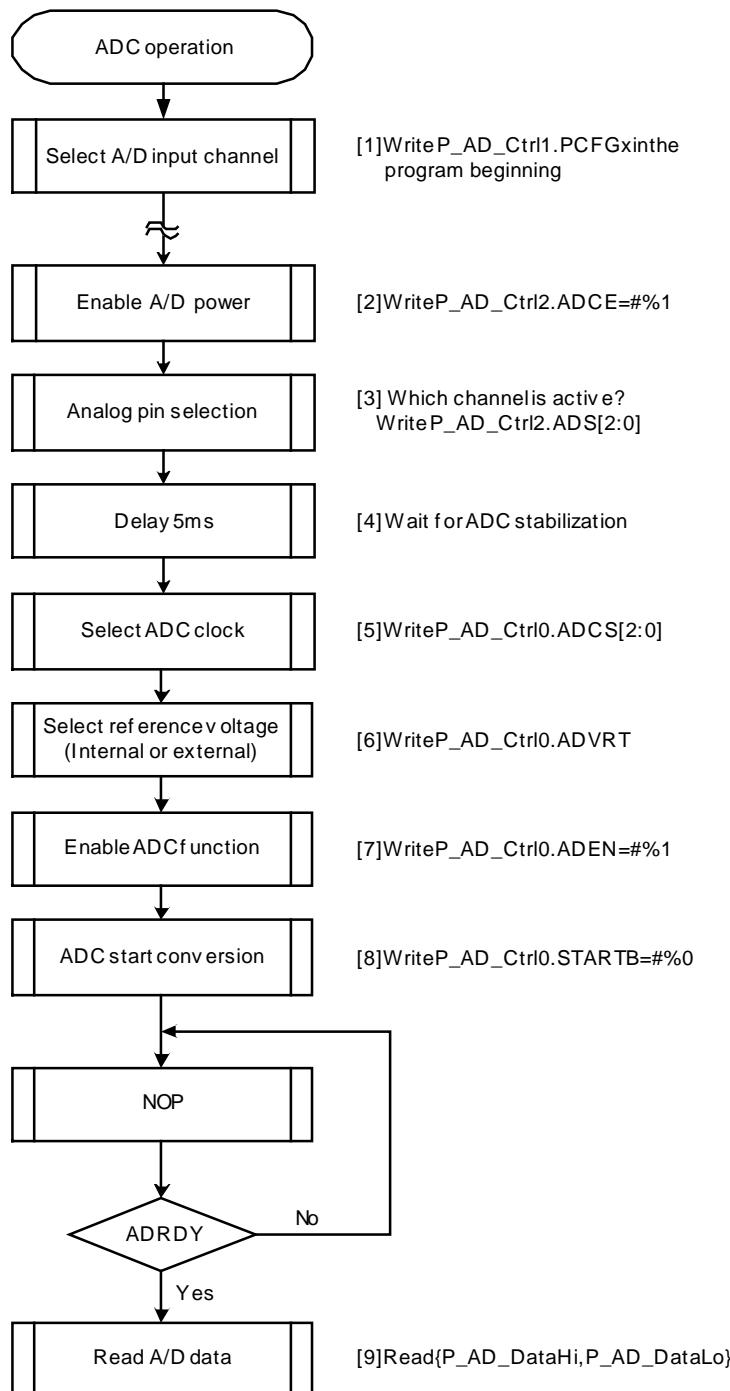


Figure 13-3 AD software flowchart

[Example 13-1]: Enable AD Converter

```

lde      #C_AD_Pin7          ; Set analog PIN, PC[7] as AN[7]
sta      P_AD_Ctrl1
lde      #(C_AD_CE+C_AD_Ch7) ; Select AN7
sta      P_AD_Ctrl2
jsr      Delay5ms           ;Delay 5msec
lde      #(C_AD_EN+C_AD_CS_8+C_AD_RDY)
                           ; Enable ADC function, ADC clock= Fsys(8MHz)/8= 1MHz(max)
sta      P_AD_Ctrl0
lde      P_AD_Ctrl0
and     #11111110B          ; Start convert
sta      P_AD_Ctrl0

```

13.4 ADC Interrupt

Interrupts can come from many sources. There is an interrupt source related with A/D converter.

n A/D converter Interrupt

There are several steps for enabling A/D converter interrupt.

1. Interrupt Disable flag (I-flag) is set to “1” by using “SEI ” instruction.
2. Configured some registers related for A/D converter operation.
3. Start Conversion.
4. Enabled A/D converter interrupt bit in P_INT_Ctrl0 register.
5. Using ‘CLI’ instruction to enable all interrupt function.
6. Now when A/D conversion is complete, the A/D converter interrupt will be occurred.

【Example 13-2】 : AD Converter by interrupt

```

lde      #C_AD_CE          ; Enable ADC power
sta      P_AD_Ctrl2
lde      #(C_AD_EN+C_AD_CS_8+C_AD_RDY)
                           ; Enable ADC function, ADC clock= Fsys(8MHz)/8= 1MHz(max)
jsr      Delay5ms           ;Delay 5msec
sta      P_AD_Ctrl0
lde      #(C_AD_Pin0+C_AD_Pin1+C_AD_Pin2+C_AD_Pin3

```

```
+C_AD_Pin4+C_AD_Pin5+C_AD_Pin6+C_AD_Pin7)
; Set analog PIN, PC[7:0] as AN[7:0]

sta      P_AD_Ctrl1
lda      #(C_AD_CE+C_AD_Ch1)    ; select AN1
sta      P_AD_Ctrl2
lda      P_AD_Ctrl0
and      #11111110B           ; start convert
sta      P_AD_Ctrl0
lda      #$FF
sta      P_INT_Flag0          ; clear INT request flag
lda      #C_INT_ADIE          ; Enable AD INT.
sta      P_INT_Ctrl0
cli      ; enable INT
```

13.5 Design Tips

[Example 13-3] : Shows how to initialize the AD converter based on SPMC65P2404A, Please refer to an example of Fortis IDE.

- AD Converter by polling
- Set ADC clock source as Fcs/8
- PA0 is analog input

```
.SYNTAX 6502                      ; process standard 6502 addressing syntax
.LINKLIST                         ; generate linklist information
.SYMBOLS                            ; generate symbolic debug information

.INCLUDE    spmc65p2404a.inc

        .PAGE0                   ; define values in the range from $00 to $FF
G_AD0_Buf      DS      2          ; AN0 ADC buffer
G_Tempbuf1     DS      1
;
        .DATA                    ; define data storage section
;
        .CODE
;
*****                                 *
;*
;* Power on Reset Process - Main Program *
;*
*****                                 *
V_Reset:
        sei                     ; Disable interrupt
        ldx      #C_STACK_BOTTOM ; Initial stack pointer at $00FF
```

```

txs                                     ; Transfer to stack point
;

lda      #$FF                         ; Clear Reset flag
sta      P_SYS_Ctrl
sta      P_SYS_Ctrl
;

jsr      F_InitIOPort                ; initial IO ports
lda      #C_AD_Pin0                  ; PA0 is analog input
sta      P_AD_Ctrl1

lda      #(C_AD_CE+ C_AD_Ch0)        ; Select AN0
sta      P_AD_Ctrl2                  ; select channel 0
jsr      Delay5ms                   ;Delay 5msec
lda      #(C_AD_EN + C_AD_CS_8)
sta      P_AD_Ctrl0                  ; ADC enable, ADC clock = Fsys / 8

;

lda      P_AD_Ctrl0
and      #11111110B                 ; start convert
sta      P_AD_Ctrl0
;

L_Main:
L_PollADC:
    lda      P_INT_Flag0
    and      #C_INT_ADIF
    beq      L_PollADC
;

    set      P_INT_Flag0, CB_INT_ADIF

    lda      P_AD_DataLo
    sta      G_AD0_Buf

    lda      P_AD_DataHi
    sta      G_AD0_Buf + 1
;

    lda      P_AD_Ctrl0
    and      #11111110B               ; start convert again
    sta      P_AD_Ctrl0
    jmp      L_Main
;

VECTOR:          .SECTION
DW      V_NMI                         ; may download program emulated either
DW      V_Reset                        ; in internal memory or external memory

```

```

DW      V_IRQ           ; dw define two bytes interrupt vector
;
; ****
; *
; *      End Of Interrupt Vector Table
; *
; *
; ****
;
.END    ; end of program

```

13.6 Related Application Notes and Libraries

This section lists some application notes and libraries about various usage of SPMC65X family. The application notes may help user develop quickly and offer the concepts about the usage of this topic. Sunplus also supports some powerful libraries about general operation, mathematics, communication, etc. The purpose of library is to help user speed up development. The current application notes and libraries related to A/D Converter are listed below.

Application Note:

File name	Application Note Series Number
I/O simulate A/D convert with RC	AN_O0316.DOC
Using ADC module	AN_O0341.DOC

Library:

File name	Application Note Series Number
SPMC65X series software basics manual	AN_O0100.DOC
SPMC65X series software advances 1 manual (data processing operation)	AN_O0101.DOC
SPMC65X series software advances 2 manual (mathematical operation)	AN_O0102.DOC

13.7 Revision History

Revision	Date	Remark
V0.1	03/31/2005	First edition

14 D/A Converter

14.1 Description

SPMC65X family are built with a maximum 10-bit Digital to Analog converter with output current 3.3mA. The setting time of D/A converter is 80u sec. Once the digital data write into DAC data register, SPMC65X family will automatically transfer this digital data, which are stored into P_DA_DataHiand P_DA_DataLo registers to correspondent analog current. The converted current that is dependent on digital data outputs through PE6 Pin. The maximum current is 3.3 mA by writing “11111111” into P_DA_DataHi and “11000000” into P_DA_DataLo. User may add one series resistor, approximate 500 ohm, to emulate linear analog voltage, which is often useful for a lot of applications like home appliance control. Figure 14-1 shows DAC block diagram.

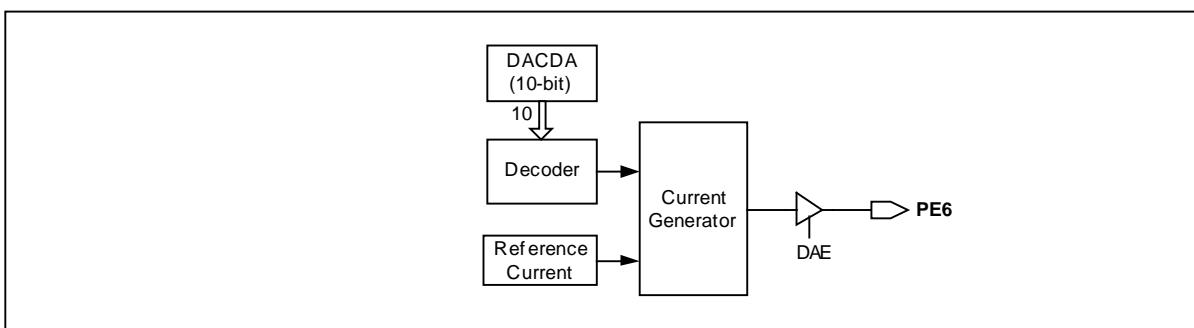


Figure 14-1 DAC Block Diagram

14.2 Control Register

14.2.1 DAC Control Register (P_DA_Ctrl, \$55) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
DAEN	-	-	-	-	-	-	-
R/W	-	-	-	-	-	-	-
0	0	0	0	0	0	0	0

Bit 7 **DAEN:** DA converter enable bit.

1=Enable DA converter

0=Disable DA converter

Bit [6:0] Reserved

14.2.2 DAC Conversion Low Data Register (P_DA_DataLo, \$56) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
DADData1	DADData0	-	-	-	-	-	-
R/W	R/W	-	-	-	-	-	-
0	0	0	0	0	0	0	0

Bit [7:6] **DADData[1:0]:** DA conversion low data, The least significant 2 bits of 10 bits.

Bit [5:0] Reserved

14.2.3 DAC Conversion High Data Register (P_DA_DataHi, \$57) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
DAData9	DAData8	DAData7	DAData6	DAData5	DAData4	DAData3	DAData2
R/W							
0	0	0	0	0	0	0	0

Bit [7:0] **DAData[9:2]**: DA conversion high data., The most significant 8 bits of 10 bits.

14.3 Operation

When DAC converter is configured for operation, the DAEN bit in P_DA_Ctrl register should be set to "1" and then analog current would be outputted and the quantity of output current are proportion to DAC data registers. The DAC data register is divided into two registers. One is P_DA_DataHi register that is stored the most significant 8 bits of converted data and the other is P_DA_DataLo register that is stored the least significant 2 bits of converted data. Setting time of DAC converter is about 80u sec and user must wait the setting time for next converter to ensure DAC conversion accurate.

14.4 DAC Interrupt

There is no interrupt source for DAC converter use.

14.5 Design Tips

[Example 14-1] : Shows how to initialize the DA converter. Please refer to an example of Fortis IDE.

n Generate a wave range from 0 to 1023

```

.SYNTAX 6502 ; process standard 6502 addressing syntax
.LINKLIST ; generate linklist information
.SYMBOLS ; generate symbolic debug information

.INCLUDE     Spmc65.inc
.PAGE0        ; define values in the range from $00 to $FF
G_DAC_Data   DS      2 ; DAC data, range = 0 ~ 1023
G_Temp       DS      1
;
.DATA         ; define data storage section
;
.CODE
; ****
;*
;* Power on Reset Process - Main Program *
;
```

```

;*****  

;  

V_Reset:  

    sei          ; Disable interrupt  

    ldx #C_STACK_BOTTOM ; Initial stack pointer at $00FF  

    txs          ; Transfer to stack point  

;  

    lda #$FF      ; Clear Reset flag  

    sta P_SYS_Ctrl  

    sta P_SYS_Ctrl  

;  

    jsr F_InitIOPort ; initial GPIO port  

;  

    set P_DA_Ctrl, CB_DA_EN ; enable DAC module  

;  

    lda #$00  

    sta G_DAC_Data  

    sta G_DAC_Data + 1  

;  

L_Main:  

    inc G_DAC_Data  

    beq L_inc_dachigh  

    jmp L_updata_DAC  

;  

L_inc_dachigh:  

    inc G_DAC_Data + 1  

    lda G_DAC_Data + 1  

    and #$03  

    sta G_DAC_Data + 1      ; High(G_DAC_Data) = 0 ~ $03  

    jmp L_updata_DAC  

;  

L_updata_DAC:  

    ; generate a wave range from 0 to 1023  

    lda G_DAC_Data  

    asl A  

    asl A  

    asl A  

    asl A  

    asl A  

    asl A  

    sta P_DA_DataLo        ; DAData[1:0]
;

```

```

        lda      G_DAC_Data
        ror      A
        ror      A
        and      #$3F           ; DAData[2:7]
        sta      G_Temp
;

        lda      G_DAC_Data + 1
        asl      A
        asl      A
        asl      A
        asl      A
        asl      A
        asl      A           ; DAData[9:8]

        adc      G_Temp          ; DAData[9:2]
        sta      P_DA_DataHi
;

        jmp      L_Main

; F_InitIOPort:
;

;     /// Initial Port A      ///
;

        lda      #00000000B
        sta      P_IOA_Data
        lda      #00000000B
        sta      P_IOA_Attrib
        lda      #11111111B
        sta      P_IOA_Dir
;

        rts

; *****
; *          *
; *      Interrupt Vector Table      *
; *          *
; *****
;

VECTOR:      .SECTION
        DW      V_NMI           ; may download program emulated either
        DW      V_Reset          ; in internal memory or external memory
        DW      V_IRQ            ; dw define two bytes interrupt vector
;

; *****

```

```
/*
;          *
; End Of Interrupt Vector Table      *
;          *
; ****
; .END                                ; end of program
```

14.6 Related Application Notes and Libraries

This section lists some application notes and libraries about various usage of SPMC65X family. The application notes may help user develop quickly and offer the concepts about the usage of this topic. Sunplus also supports some powerful libraries about general operation, mathematics, communication, etc. The purpose of library is to help user speed up development. The current application notes and libraries related to D/A Converter are listed below.

Application Note:

Title	Application Note Series Number
I/O simulate D/A convert with PWM	AN_O0317.DOC

Library:

Title	Application Note Series Number
SPMC65X series software basics manual	AN_O0100.DOC
SPMC65X series software advances 1 manual (data processing operation)	AN_O0101.DOC
SPMC65X series software advances 2 manual (mathematical operation)	AN_O0102.DOC

14.7 Revision History

Revision	Date	Remark
V0.1	03/31/2005	First edition

15 Comparator

15.1 Description

SPMC65X family' devices are built with maximum two channels of voltage comparator. The comparator pins (CMPIN1/CMPIN0) are shared with I/O Pins through multiplexers. The multiplexers can be controlled easily by setting comparator control register shown in Section15.2. The comparator module can compare the external voltage input coming from CMPIN1/CMPIN0 with the external voltage reference input on PE4/PE2, or with the internal voltage reference (1.2V). The comparator can be enabled with the setup of comparison criterion and the reference source, and selectable interrupt input for event service. The input operating range of the comparator is 0.2V to (VDD-0.2) V. Figure 15-1 shows comparator block diagram.

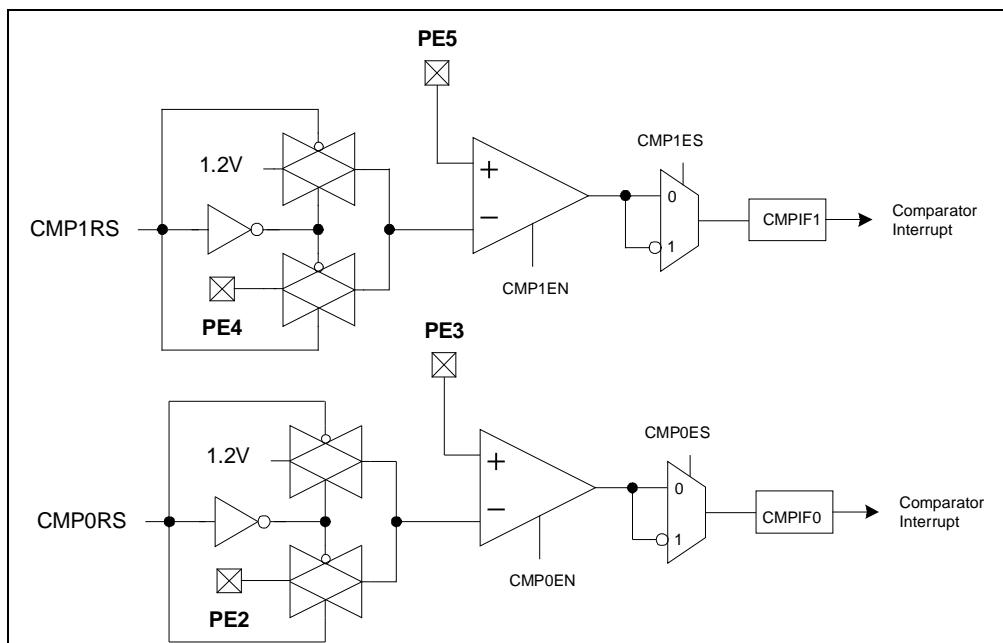


Figure 15-1 Comparator Block Diagram

15.2 Control Register

15.2.1 Comparator Control Register (P_CMP_Ctrl, \$2E) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
CMP1EN	CMP1RS	CMP1LOG	CMP1OUT	CMP0EN	CMP0RS	CMP0LOG	CMP0OUT
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit 7 **CMP1EN:** Comparator1 function enable bit.

1=enable comparator1

	0=disable comparator1
Bit 6	CMP1RS: Comparator1 Reference source selection bit. 1=External reference input (PE4) 0=Internal reference input (1.2v)
Bit 5	CMP1LOG: Comparator1 Logic event direction selection bit. 1=Vcmp1in > Vcmp1ref, it will generate flag 0=Vcmp1in < Vcmp1ref, it will generate flag
Bit 4	CMP1OUT: Comparator1 Result bit. 1=Vcmp1in>Vcmp1ref 0=Vcmp1in<Vcmp1ref
Bit 3	CMP0EN: Comparator0 function enable bit. 1=enable comparator0 0=disable comparator0
Bit 2	CMP0RS: Comparator0 Reference source selection bit. 1=External reference input (PE2) 0=Internal reference input (1.2v)
Bit 1	CMP0LOG: Comparator0 Logic event direction selection bit. 1=Vcmp0in > Vcmp0ref, it will generate flag 0=Vcmp0in < Vcmp0ref, it will generate flag
Bit 0	CMP0OUT: Comparator0 Result bit. 1=Vcmp1in>Vcmp0ref 0=Vcmp1in<Vcmp0ref

15.3 Operation

Two channels of voltage Comparators are built in the comparator module. Each one owned its independent control bits. In Comparator Control Register, Bit7 to Bit4 are for Comparator1 use and Bit3 to Bit0 are for Comparator0 use. When comparator operation is enabled, the CMPxEN(x=0~1) should be set to '1'. In addition, the comparator module also allows the selection of two kinds of Comparators reference sources that are external voltage reference input on PE4/PE2, or the internal voltage reference(1.2V). These two kinds of reference sources are determined by CMPxRS(x=0~1) bit in the P_CMP_Ctrl register.

The Comparator also supports interrupt functions. CMPxLOG(x=0~1) bit is determined to select what conditions the interrupt would be occurred. For example, if Comparator interrupt enable bit COMxE(x=0~1) in P_INT_Ctrl2 is set and CMPxLOG(x=0~1) is set to '1', when the analog input in comparator is greater than comparator reference, the comparator interrupt will be occurred and CMPxOUT(x=0~1) bit used to indicate the result of compare will be set to '1'. For a step-by-step procedure on how to set up the Comparator interrupt, see Section 15.4 in detail.

15.4 Comparator Interrupt

Interrupts can come from many sources. There are two interrupt sources related with Comparator interrupt.

- | Comparator0 Interrupt
- | Comparator1 Interrupt

There are several steps for enabling Comparator interrupt.

1. Interrupt Disable flag (I-flag) is set to "1" by using "SEI" instruction.
2. Configured some registers related for Comparator operation.
3. Enabled Comparator interrupt bit in P_INT_Ctrl2 register.
4. Using 'CLI' instruction to enable all interrupt function.
5. Now, when the result of Comparator is the same as CMP1LOG / CMP2LOG condition, Comparator interrupt will be occurred.

15.5 Design Tips

[Example 15-1]: Shows how to initialize Comparatorter 0 by interrupt. Please refer to an example of Fortis IDE.

- n Enable comparator 0
- n Internal 1.2V as reference
- n Vcmp_in > Vcmpref will generate interrupt

```
.SYNTAX 6502 ; process standard 6502 addressing syntax
.LINKLIST ; generate linklist information
.SYMBOLS ; generate symbolic debug information

.INCLUDE      Spmc65.inc

.PAGE0          ; define values in the range from $00 to $FF
;
.DATA           ; define data storage section
;
.CODE
; ****
;*
;* Power on Reset Process - Main Program *
;*
; ****
```

```

V_Reset:
    sei                      ; Disable interrupt
    ldx #C_STACK_BOTTOM      ; Initial stack pointer at $00FF
    txs                      ; Transfer to stack point
;
    lda #$FF                 ; Clear Reset flag
    sta P_SYS_Ctrl
    sta P_SYS_Ctrl
;
    jsr F_InitIOPort        ; initial GPIO port
;
    lda #(C_CMP0_EN + C_CMP0_LOG)
    sta P_CMP_Ctrl          ; comparator 0 enable, internal 1.2V as reference
                           ; Vcmp_in > Vcmp_ref will generate the flag
;
    set P_INT_Ctrl2, C_INT_CMP0IE; enable CMP0 interrupt
;
    cli                      ; enable INT
;
L_Main:
    nop
    jmp L_Main
;
*****
;*
;*      *                         *
;*      IRQ Interrupt Service Routine      *
;*      *                         *
;*****


V_IRQ:
    pha                      ; push A register
    txa                      ; transfer X to A
    pha                      ; push A register (ie. push X)
;
; Time Base iterval timer interrupt ?
;
    lda P_INT_Flag2
    and #C_INT_ITVALIF
    beq L_exit_irq
;
    set P_INT_Flag2, CB_INT_ITVALIF
;
    lda P_IOA_Data
    eor #$FF
    sta P_IOA_Data           ; toggle P_IOA_Data
;
```

```

;

L_exit_irq:
    pla                      ; pop A register
    tax                      ; transfer A to X
    pla                      ; pop A register (ie. pop X)
;
    rti

; *****
;*          *
;*      Interrupt Vector Table      *
;*          *
;****

VECTOR:      .SECTION
    DW      V_NMI           ; may download program emulated either
    DW      V_Reset          ; in internal memory or external memory
    DW      V_IRQ            ; dw define two bytes interrupt vector
;
; *****
;*          *
;*      End Of Interrupt Vector Table      *
;*          *
;****

.END          ; end of program

```

15.6 Related Application Notes and Libraries

This section lists some application notes and libraries about various usage of SPMC65X family. The application notes may help user develop quickly and offer the concepts about the usage of this topic. Sunplus also supports some powerful libraries about general operation, mathematics, communication, etc. The purpose of library is to help user speed up development. The current application notes and libraries related to Comparator are listed below.

Application Note:

Title	Application Note Series Number
No associated application notes at this time.	

Library:

Title	Application Note Series Number
SPMC65X series software basics manual	AN_O0100.DOC
SPMC65X series software advances 1 manual (data processing operation)	AN_O0101.DOC
SPMC65X series software advances 2 manual (mathematical operation)	AN_O0102.DOC

15.7 Revision History

Revision	Date	Remark
V0.1	03/31/2005	First edition

16 Interval Timer and Buzzer

16.1 Description

The SPMC65X family have an 8-bit time base Interval timer. It is used to produce a reference period needed for system control on the chip. As many as fifteen-stage can be optional to change interval timer period.

The 8-bit basic interval timer is increased every system clock pulse (F_{SYS}) and it will cause an interrupt when counter matched interval timer period setting by user.

SPMC65X family' devices also support one channel Buzzer output. A 50% duty pulse can be outputted using the buzzer output circuit, which is useful for buzzer drive. The buzzer output frequency can be selected by setting BZFS[3:0] in P_BUZ_Ctrl register. Detailed registers setting about interval and buzzer are shown in Section 16.2.

16.2 Control Register

16.2.1 Interval Timer and Buzzer Control Register (P_BUZ_Ctrl, \$2D) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
INTIMS3	INTIMS2	INTIMS1	INTIMS0	BZFS3	BZFS2	BZFS1	BZFS0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit [7:4] **INTIMS[3:0]**: Interval Timer Clock or Period Selection bits.

0000 = Function disable

0001 = $F_{SYS} /128$

0010 = $F_{SYS} /256$

0011 = $F_{SYS} /512$

0100 = $F_{SYS} /1024$

0001 = $F_{SYS} /2048$

0110 = $F_{SYS} /4096$

0111 = $F_{SYS} /8192$

1000 = $F_{SYS} /16384$

1001 = $F_{SYS} /32768$

1010 = $F_{SYS} /65535$

1011 = $F_{SYS} /131072$

1100 = $F_{SYS} /262144$

1101 = $F_{SYS} /524288$

1110 = $F_{SYS} /2^{21}$

1111 = $F_{SYS} /2^{23}$

Bit [3:0] **BZFS[3:0]**: Buzzer Clock or Frequency Selection bits

0000 = Function disable

0001 = $F_{SYS} /64$

0010 = $F_{SYS} /128$

0011 = $F_{SYS} /256$

0100 = $F_{SYS} /512$

0101 = $F_{SYS} /1024$

0110 = $F_{SYS} /2048$

0111 = $F_{SYS} /4096$

1000 = $F_{SYS} /8192$

1001 = $F_{SYS} /4$

1010 = $F_{SYS} /8$

1011 = $F_{SYS} /16$

1100 = $F_{SYS} /32$

1101 = $F_{SYS} /32$

1110 = $F_{SYS} /32$

1111 = $F_{SYS} /32$

16.3 Operation

16.3.1 Interval Timer

The interval timer is used as a time-base timer. The period can be selected by setting INTIMS [3:0] in P_BUZ_Ctrl register. Table 16.1 listed Interval Timer Period selection. Fifteen-stages can be optional to change interval timer period. Figure 16-1 shows the timing of time base.

Table 16-1 Time Base Interval Timer List (if the System clock is $F_{SYS} = 8.0\text{MHz}$)

INTIMS [3:0]	Interval Timer period	
	Divisor	$F_{T0} = F_{SYS} / \text{Divisor}$
0000	-	Function disable
0001	$2^7 = 128$	16us
0010	$2^8 = 256$	32us
0011	$2^9 = 512$	64us
0100	$2^{10} = 1024$	128us
0101	$2^{11} = 2048$	256us
0110	$2^{12} = 4096$	512us
0111	$2^{13} = 8192$	1.024ms
1000	$2^{14} = 16384$	2.048ms
1001	$2^{15} = 32768$	4.096ms
1010	$2^{16} = 65535$	8.192ms
1011	$2^{17} = 131072$	16.384ms
1100	$2^{18} = 262144$	32.768ms
1101	$2^{19} = 524288$	65.535ms
1110	2^{21}	262.144ms
1111	2^{23}	1s

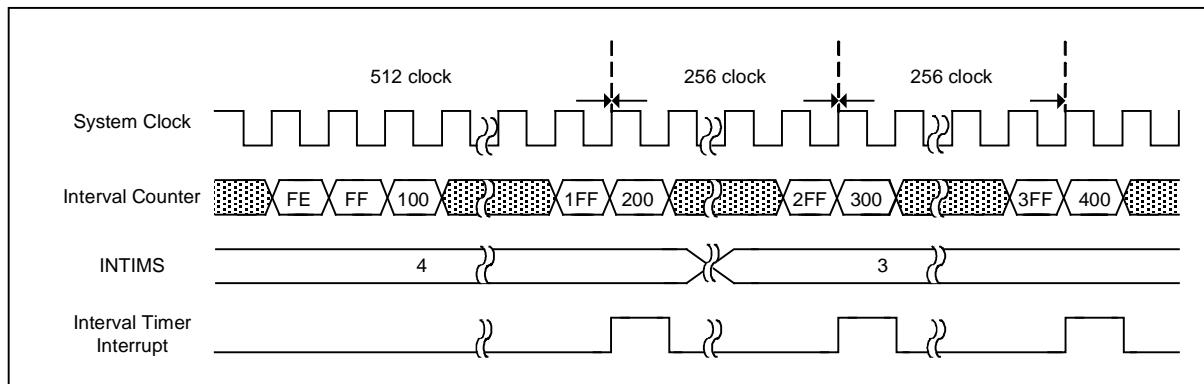


Figure 16-1 Time Base Interval Timer Operating Waveform

[Example 16-1]: Generate 512us Time Base Interval Timer.

lda	#C_TBASE_Div_4k ; Store \$60 to P_BUZ_Ctrl register
sta	P_BUZ_Ctrl

16.3.2 Buzzer

A square-wave about 50% duty pulse can be outputted when buzzer driver is enabling. The frequency of buzzer output can be ranged from 976hz to 2Mhz by setting BZFS[3:0] in the P_BUZ_Ctrl (\$2D) register. Table 16-2 listed Buzzer output selection.

Table 16-2 Buzzer Output List

BZFS [3:0]	BZO (Buzzer output) rate ($F_{SYS} = 8.0\text{MHz}$)	
	Divisor	$F_{T0} = F_{SYS} / 1^*$
0000		Function disable
0001	64	8us
0010	128	16us
0011	256	32us
0100	512	64us
0101	1024	128us
0110	2048	256us
0111	4096	512us
1000	8192	1.024ms
1001	4	0.5us
1010	8	1us
1011	16	2us
1100	32	4us
1101	32	4us
1110	32	4us
1111	32	4us

To enable the Interval Timer or Buzzer, user only configured the P_BUZ_Ctrl register. When interval timer needed to enable, user could set the most significant 4 bits of P_BUZ_Ctrl according to period selection. The Interval Timer also supports interrupt function. By enabling Interval timer bit (ITVALIE) in the P_INT_Ctrl2 register and setting interval timer register, the Interval Timer interrupt was issued on condition that counter matched interval timer period setting by user. The frequency of buzzer output is controlled by setting the least 4 bits of P_BUZ_Ctrl register. Once the buzzer function is enable, the 50% duty pulse will output through PB6 pin.

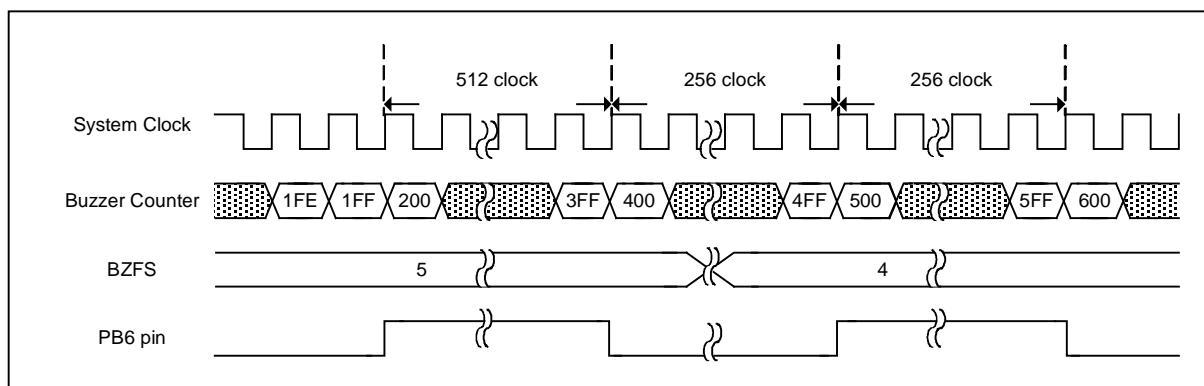


Figure 16-2 Buzzer Operating waveform

[Example 16-2]: Generate pulse output using Buzzer function, which period is 128us and duty is 50%.

lda	#C_BUZ_Div_1k ; store \$05 to accumulator
sta	P_BUZ_Ctrl

16.4 Time base interval Interrupt

[Example 16-3]: Generate 512us interrupt using Time Base Interval Timer.

lda	#C_TBASE_Div_4k ; Store \$60 to P_BUZ_Ctrl register
sta	P_BUZ_Ctrl
set	P_INT_Ctrl2, CB_INT_ITVALIE ; Enable Interval Timer interrupt
cli	; enable INT

16.5 Design Tips

【Example 16-4】: Shows how to initialize Interval Timer based on SPMC65P2404A. Please refer to an example of Fortis IDE.

- Enable Interval Timer
- Interval Timer frequency is 1.953 KHz

```

.SYNTAX 6502           ; Process standard 6502 addressing syntax
.LINKLIST              ; Generate linklist information
.SYMBOLS               ; Generate symbolic debug information

.INCLUDE    SPMC65P2404A.inc      ; Define all hardware, Registers and ports.

.PAGE0                 ; define values in the range from $00 to $FF

.DATA                  ; define data storage section

.CODE
; ****
;*
;* Power on Reset Process - Main Program *
;*
;****

.PUBLIC     V_Reset
V_Reset:
    sei          ; Disable interrupt
    ldx #C_STACK_BOTTOM ; Initial stack pointer at $00FF
    txs          ; Transfer to stack point
;
    lda #$FF      ; Clear Reset flag
    sta P_SYS_Ctrl
    sta P_SYS_Ctrl
;
    jsr F_InitIOPort ; initial GPIO port
;
    lda #C_TBASE_Div_4k
    sta P_BUZ_Ctrl      ; time base clock = 8.0MHz / 4096 = 1.953 KHz
    set P_INT_Ctrl2, CB_INT_ITVALIE; enable time base interrupt
    cli
;
L_MainLoop:
    nop
    jmp L_MainLoop
;

F_InitIOPort:

```

```

;           /// Initial Port A      ///
;

    lda      #00000000B
    sta      P_IOA_Data
    lda      #00000000B
    sta      P_IOA_Attrib
    lda      #11111111B
    sta      P_IOA_Dir

;

    rts

; *****
; *
; *      IRQ Interrupt Service Routine      *
; *
; *****
;

V_IRQ:
    pha          ; push A register
    txa          ; transfer X to A
    pha          ; push A register (ie. push X)

;

; Time Base iterval timer interrupt ?
;

    lda      P_INT_Flag2
    and      #C_INT_ITVALIF
    beq      L_exit_irq

;

    set      P_INT_Flag2, CB_INT_ITVALIF

;

    lda      P_IOA_Data
    eor      #$FF
    sta      P_IOA_Data      ; toggle P_IOA_Data

;

L_exit_irq:
    pla          ; pop A register
    tax          ; transfer A to X
    pla          ; pop A register (ie. pop X)

;

    rti

; *****
; *
; *      Interrupt Vector Table      *
; *
; *****
;
```

```

VECTOR      .SECTION
DW          V_NMI           ; Non-mask interrupt vector(no use)
DW          V_Reset          ; Reset vector
DW          V_IRQ            ; IRQ interrupt vector

; *****
;*
;*      End Of Interrupt Vector Table   *
;*
;****

.END        ; end of program

```

16.6 Related Application Notes and Libraries

This section lists some application notes and libraries about various usage of SPMC65X family. The application notes may help user develop quickly and offer the concepts about the usage of this topic. Sunplus also supports some powerful libraries about general operation, mathematics, communication, etc. The purpose of library is to help user speed up development. The current application notes and libraries related to Interval Timer and Buzzer are listed below.

Application Note:

Title	Application Note Series Number
Using Time Base Interval Timer	AN_O0337.DOC

Library:

Title	Application Note Series Number
SPMC65X series software basics manual	AN_O0100.DOC
SPMC65X series software advances 1 manual (data processing operation)	AN_O0101.DOC
SPMC65X series software advances 2 manual (mathematical operation)	AN_O0102.DOC

16.7 Revision History

Revision	Date	Remark
V0.1	03/31/2005	First edition

17 Watch Dog Timer

17.1 Description

The purpose of a watchdog timer is to monitor if the system operates normally. Within a certain period, watchdog counter must be cleared. For example, if the watchdog is not cleared within a setting period, CPU will issue a watchdog interrupt if WDT interrupt enable bit is set to '1'. Once watchdog interrupt are persisted for 8 times without clearing watchdog counter, CPU assumes the program has been running in an abnormal condition and therefore, CPU will reset the system to the initial state and start running the program from beginning. It protects the system from incorrect code execution by launching a CPU reset when the watchdog timer overflows as a result of failure of software to clear the timer within selection time. For SPMC65X family's devices, watchdog function can be enabled or disabled by setting in Fortis IDE as Figure 5-7. The Watchdog Timer Block Diagram is shown in Figure 17-1.

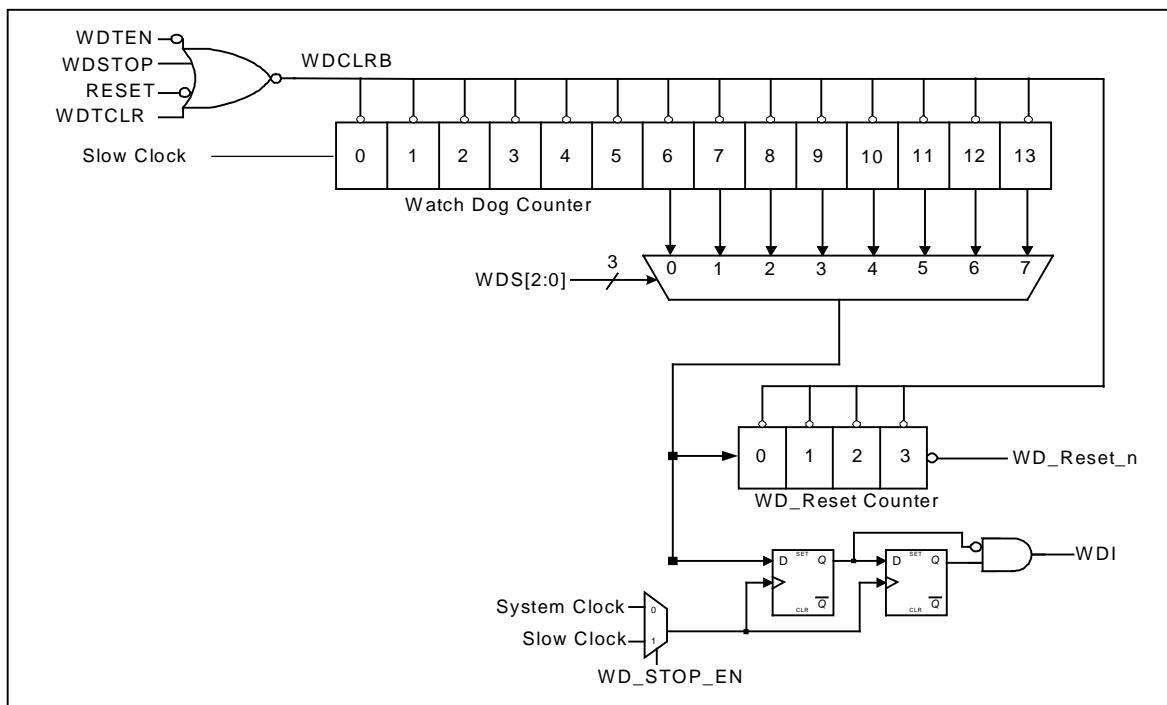


Figure 17-1 Watchdog Timer Block Diagram

17.2 Control Register

17.2.1 Watch Dog Timer Control Register (P_WDT_Ctrl, \$32) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
SCKEN	WDSCS2	WDSCS1	WDSCS0	-	-	-	-
R/W	R/W	R/W	R/W	-	-	-	-
1	1	1	1	-	-	-	-

- Bit 7 **SCKEN:**Slow Clock Enable bit in Stop mode
1= Enable Slow Clock in stop mode
0= Disable Slow Clock in stop mode
- Bit [6:4] **WDCS:** Watch Dog interrupt Clock rate Selection bits
000 = f_slow/128
001 = f_slow/256
010 = f_slow/512
011 = f_slow/1024
100 = f_slow/2048
101 = f_slow/4096
110 = f_slow/8192
111 = f_slow/16384
f_slow: internal build-in RC oscillator, frequency is 25kHz

WDS[2:0]	Watch Dog Reset (Hz)	Watch Dog Interrupt Clock (Hz)
000	195/8	195
001	97/8	97
010	48/8	48
011	24/8	24
100	12/8	12
101	6/8	6
110	3/8	3
111	1.5/8	1.5

Bit [3:0] Reserved

Note: All of above bits need write twice to set corresponding ones.

17.2.2 Watch Dog Timer Clear Register (P_WDT_Clr, \$10) (W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
-	-	-	-	-	-	-	-
R/W							
0	0	0	0	0	0	0	0

Bit [7:0] **WDT_CLR[7:0]:**_Watchdog clear command

Write:

55= Clear watchdog counter

Otherwise = No action

Read:

Always \$00

Note1: While using watchdog timer, user can select suitable frequency in P_WDT_Ctrl (\$32).

Note2: It is used to clear watchdog counter by writing '\$55' to this register.

17.3 Operation

Watchdog function can be enabled or disabled by device configuration registers setting in Figure 17-1. When watchdog is enabled, a watchdog time out would generate system RESET. Within a certain period decided by P_WDT_Ctrl.WDCS[2:0], watchdog counter must be cleared by writing C_WDT_Clr (\$55) to P_WDT_Clr register. If the watchdog is not cleared, CPU will issue a watchdog interrupt on condition that P_INT_Flag0.WDIF bit is set to '1' in P_INT_Flag0 register. Once watchdog interrupt are persisted for 8 times without clearing watchdog counter, CPU will reset automatically. The clock source of watchdog is derived from an on-chip RC oscillator that does not require any external component. Typically, the on-chip RC oscillator frequency is about 25Khz. User may select Watch Dog interrupt Clock rate to produce desired period by setting P_WDT_Ctrl.WDCS[2:0] bits in the P_WDT_Ctrl register. Note that enabling RC oscillator may cause a little power consumption in stop mode. For more information about power saving operation, please refer chapter 21.

For a step-by-step procedure on how to set up the Watch Dog interrupt, see Section 17.4 in detail.

[Example 17-1]: Set Watch Dog Timer as 1.5Hz.

lda	#C_WDT_Div_16384	; WDI= Fslow(25KHz)/16384= 1.5Hz
sta	P_WDT_Ctrl	
sta	P_WDT_Ctrl	

17.4 Watch Dog Interrupt

Interrupts can come from many sources. There is an interrupt source related with Watch Dog.

I Watch Dog Interrupt

There are several steps for enabling Watch Dog interrupt.

1. Watchdog function is enabled by setting in Fortis IDE as Figure 5-7.
2. Interrupt Disable flag (I-flag) is set to "1" by using "SEI" instruction.
3. Configured some registers related for Watch Dog operations such as WDT interrupt enable bit in the P_INT_Ctrl0 register and WDT period in the P_WDT_Ctrl register.
4. Using 'CLI' instruction to enable all interrupt function.
5. Now If Watch Dog timer overflows, Watch Dog interrupt will be occurred.

[Example 17-2]: Enable Watch Dog Interrupt

lda	#\$FF	
sta	P_INT_Flag0	; clear INT request flag
set	P_INT_Ctrl0, CB_INT_WDIE	; Enable WDT INT

cli ; enable INT

[Example 17-3] : Disable slow clock (25Khz) in stop mode using bit operation for purposes of power saving consideration.

clr P_WDT_Ctrl,7
clr P_WDT_Ctrl,7

17.5 Design Tips

[Example 17-4] : Shows how to initialize Watchdog based on SPMC65P2404A. Please refer to an example of Fortis IDE.

- Set Watchdog interrupt clock rate as F_slow/512 (48.828 Hz).

.SYNTAX 6502 ; process standard 6502 addressing syntax
.LINKLIST ; generate linklist information
.SYMBOLS ; generate symbolic debug information
.INCLUDE spmc65p2404a.inc
.PAGE0 ; define values in the range from \$00 to \$FF
;
.DATA ; define data storage section
;
.CODE
; ****
; *
; * Power on Reset Process - Main Program *
; *
; ****
V_Reset:
sei ; Disable interrupt
Idx #C_STACK_BOTTOM ; Initial stack pointer at \$00FF
txs ; Transfer to stack point
;
ldx #\$FF ; Clear Reset flag
sta P_SYS_Ctrl
sta P_SYS_Ctrl
;
jsr F_InitIOPort ; initial GPIO port
;
ldx #C_WDT_Div_512 ; Watchdog interrupt clock rate = F_slow/512 = 48.828 Hz
sta P_WDT_Ctrl
sta P_WDT_Ctrl

```

lda      #$FF
sta      P_INT_Flag0           ; clear INT request flag
sta      P_INT_Flag1
set      P_INT_Ctrl0, CB_INT_WDIE ; Enable WDT INT
cli      ; enable INT
;
L_Main:
nop
jmp      L_Main
;
F_InitIOPort:
;
;      /// Initial Port A      ///
;
lda      #00000000B
sta      P_IOA_Data
lda      #0000000B
sta      P_IOA_Attrib
lda      #1111111B           ; PortA as input with pull-low
sta      P_IOA_Dir
;
rts
; ****
;*
;*      IRQ Interrupt Service Routine      *
;*
; ****
V_IRQ:
pha      ; push A register
txa      ; transfer X to A
pha      ; push A register (ie. push X)
;
; WDT timer overflow interrupt ?
;
lda      P_INT_Flag0
and      #C_INT_WDIF
beq      L_exit_irq
;
set      P_INT_Flag0, CB_INT_WDIF
;
lda      P_IOA_Data
eor      #$FF
sta      P_IOA_Data           ; toggle P_IOA_Data
;
```

```

;
L_exit_irq:
    pla                      ; pop A register
    tax                      ; transfer A to X
    pla                      ; pop A register (ie. pop X)
;
    rti
;
; *****
; *          *
; *      Interrupt Vector Table      *
; *          *
; *****
;
VECTOR:      .SECTION
    DW      V_NMI           ; may download program emulated either
    DW      V_Reset          ; in internal memory or external memory
    DW      V_IRQ            ; dw define two bytes interrupt vector
;
; *****
; *          *
; *      End Of Interrupt Vector Table      *
; *          *
; *****
;
.END          ; end of program

```

17.6 Related Application Notes and Libraries

This section lists some application notes and libraries about various usage of SPMC65X family. The application notes may help user develop quickly and offer the concepts about the usage of this topic. Sunplus also supports some powerful libraries about general operation, mathematics, communication, etc. The purpose of library is to help user speed up development. The current application notes and libraries related to Watch Dog Timer are listed below.

Application Note:

Title	Application Note Series Number
Using Watchdog timer	AN_O0340.DOC

Library:

Title	Application Note Series Number
SPMC65X series software basics manual	AN_O0100.DOC



SPMC65X series software advances 1 manual (data processing operation) AN_O0101.DOC

SPMC65X series software advances 2 manual (mathematical operation) AN_O0102.DOC

17.7 Revision History

Revision	Date	Remark
V0.1	03/31/2005	First edition

18 SPI

18.1 Description

SPI function is the abbreviation of Serial Peripheral Interface. The SPMC65X family devices include the four-pin SPI module. The SPI is a high-speed synchronous serial I/O that allows a serial of bit stream to be transmitted out or received into the device at a programmable transfer rate. The SPI supports full-duplex synchronous transfer between a master device and a slave device. The SPMC65X family support both master and slave modes. The parameters related to SPI such as operation mode, clock frequency, clock phase, and clock polarity are programmable. The SPI module provides the following features:

- Four external pins:
 - SDO: data output pin (shared with PC3)
 - SDI: data input pin (shared with PC2)
 - SCK: clock input/output pin (shared with PC1)
 - SSB: Slave select pin (shared with PC0)
- Supports full-duplex synchronous transfer
- Two operation modes: master and slave
- Baud rate: 8 programmable transfer rate / Max. 2Mbps at 8 MHz CPU clock
- Data word length: 8-bit
- Programmable clock phase and clock polarity settings
- Selectable data strobe time: input data bit sampled at the middle/end of data output time
- SPI TX/RX buffer is only one byte

Following is a function diagram of SPI module.

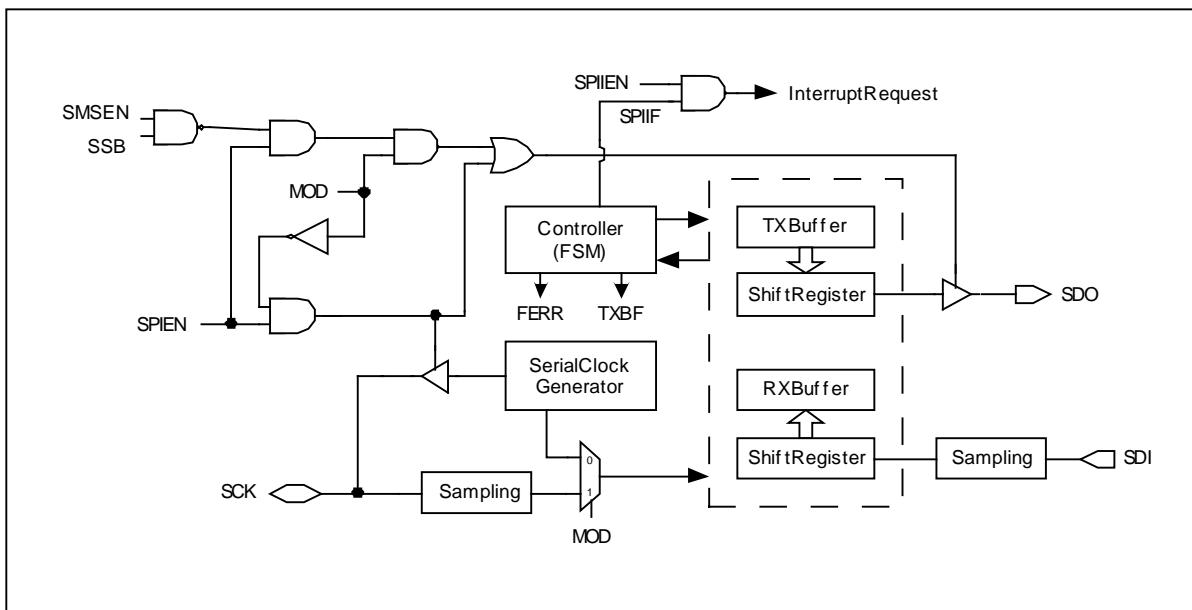


Figure 18-1 Function Diagram of SPI Module

18.2 Application Circuit

The SPI interface is usually used to communicate with EEPROM. The Figure 18-2 shows the application circuit between 93C46 and SPMC65X family MCU.

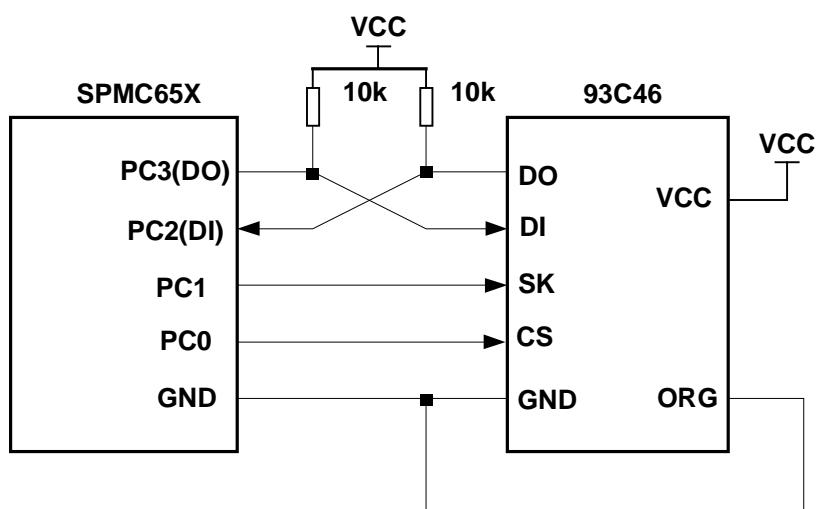


Figure 18-2 Application Circuit For SPI Operation

18.3 Control Register

18.3.1 SPI Control Register0 (P_SPI_Ctrl0, \$38)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
SPIEN	MOD	SCKPHA	SCKPOL	SMS	SCKSEL2	SCKSEL1	SCKSEL0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit 7 **SPIEN:** SPI enable bit. Once this bit is set to 1, PC[3:0] become SPI interface.

1=Enable SPI

0=Disable SPI

Bit 6 **MOD:** SPI mode

1=Slave mode

0=Master mode

Bit 5 **SCKPHA:** SPI clock phase. SPI clock phase select, see SPI Master Mode Timing

Bit 4 **SCKPOL:** SPI clock polarity. SPI clock polarity select, see SPI Master Mode Timing

Bit 3 **SMS:** Sample mode selection bit for master mode

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 [2:0] **CKSEL [2:0]:** Master mode clock selection bit

111= $F_{SYS} \div 128$

110= $F_{SYS} \div 128$

101= $F_{SYS} \div 128$

100= $F_{SYS} \div 64$

011= $F_{SYS} \div 32$

010= $F_{SYS} \div 16$

001= $F_{SYS} \div 8$

000= $F_{SYS} \div 4$

F_{SYS} : Frequency of System Clock

18.3.2 SPI Control Register1 (P_SPI_Ctrl1, \$39)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
SMSEN	SWRST	-	-	-	-	SPISPCLK1	SPISPCLK0
R/W	R/W	-	-	-	-	R/W	R/W
0	0	0	0	0	0	1	0

Bit 7 **SMSEN:** SPI Slave mode Selection input

1=PC0 become SSB input pin

0=PC0 is GPIO

SSB: Slave mode Selection, low active

Bit 6	SWRST: SPI software reset
	Write: 1= generate one pulse to reset SPI module except register setting 0=No action.
	Read: Always 0
Bit [5:2]	Reserved
Bit [1:0]	SPISPCLK [1:0]: Sampling clock selection bits
	11= $F_{SYS} \div 4$
	10= $F_{SYS} \div 2$
	01= F_{SYS}
	00= No sampling
	F_{SYS} : Frequency of System Clock

Note: The purpose of sampling clock is to prevent received data from glitch noise, but lower sampling clock would affect the speed of communication

18.3.3 SPI State Status Register (P_SPI_Status, \$3A)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
SPIIF	SPIIEN	TXBF	0	0	0	0	BUFFull
R/W	R/W	R	-	-	-	-	R
0	0	0	0	0	0	0	0

Bit 7	SPIIF: SPI interrupt flag
	Read: 1=SPI interrupt flag is active 0=No flag
	Write: 1= Clear flag 0=No action.
Bit 6	SPIIEN: SPI interrupt enable bit
	1=Enable 0=Disable
Bit 5	TXBF: Transmission buffer full flag. 1= Transmission buffer is full. 0=Transmission buffer is empty
Bit [4:1]	Reserved
Bit 0	BUFFull: Buffer full and overwrite 1=Overwrite 0=buffer is under normal condition

18.3.4 SPI Transmission Buffer Register0 (P_SPI_TxData, \$3B)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
SPITXDATA7	SPITXDATA6	SPITXDATA5	SPITXDATA4	SPITXDATA3	SPITXDATA2	SPITXDATA1	SPITXDATA0
R/W							
0	0	0	0	0	0	0	0

Bit [7:0] **SPITXDATA:** SPI Transmit data

Read: Always is #00

Write: Transmission data

18.3.5 SPI Receive Buffer Register0 (P_SPI_RxData, \$3C)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
SPIRXDATA7	SPIRXDATA6	SPIRXDATA5	SPIRXDATA4	SPIRDATA3	SPIRXDATA2	SPIRXDATA1	SPIRXDATA0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit [7:0] **SPIRXDATA:** SPI Receive data

Read: SPI Receive data

Write: No Action

18.4 Operation

18.4.1 Master Mode

In Master mode operation, the shifting clock (SCK) is generated by SPI module. There are two control bits to control the clock phase (SCKPHA) and polarity (SCKPOL) in the P_SPI_Ctrl0 register. The transmission starts immediately when data is written to the P_SPI_TxBuf register. In addition, the SDI pin (PC2) can be programmed as high-impedance if user wants to receive the data from slave device. SPMC65X family's devices support receiving and transmitting interrupt for SPI module. The interruption can be enabled simultaneously by setting SPIIEN bit to '1' in P_SPI_Status register. Similarity, SPIIF bit has to be clear after receiving or transmitting interrupt service routine execution.

After one byte data is wrote in P_SPI_TxBuf register, the data is latched into its internal transmission buffer. If the shift register is empty, the data will be loaded to the shift register and start transmitting at the next SCK phase. On the other hand, if the shift register is busy in shifting data (TXBF flag is set in P_SPI_Status register), the new data will not be loaded until the present byte has been shifted out.

The SPI shifts the data from MSB to LSB through the SDO pin. The 8-bit data is shifted out after eight SCK cycles. At the same time, the data is also shifted in through SDI pin. When each 8-bit transfer is completed, the SPIIF bit in P_SPI_Status register will be set; besides, a SPI interrupt will be generated if the SPIIEN bit is set to '1' in P_SPI_Status register.

In contrast, while SPI interface is received one byte successfully, the received data will be latched into received buffer. At that time, SPIIF bit in P_SPI_Status register will be set and a SPI interrupt will be issued to CPU if the SPIIEN bit in the P_SPI_Status register is set.

The following diagram depicts the timing scheme on SPI master mode for different operation types (polarity control bit equals “1” or “0”, phase control bit equals “1” or “0”, and sample strobe control bit equals “1” or “0”).

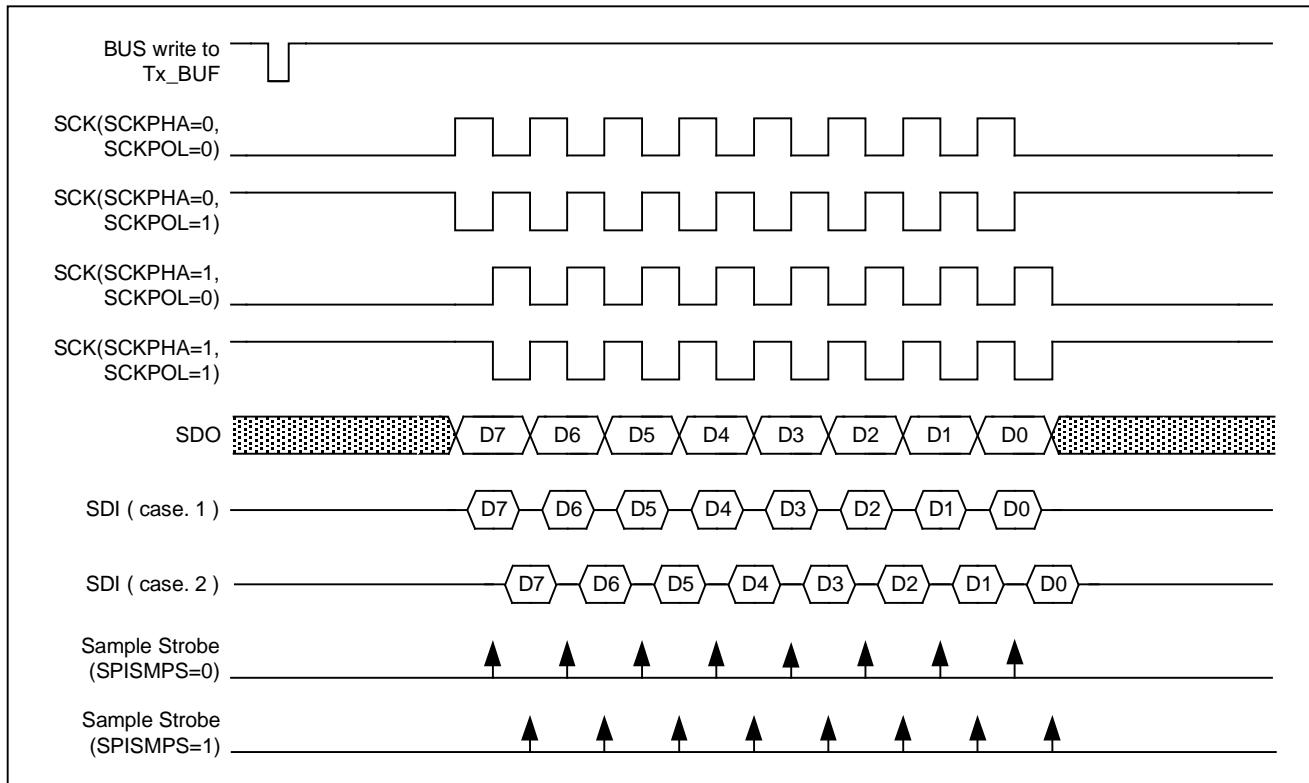


Figure 18-3 SPI Master mode timing

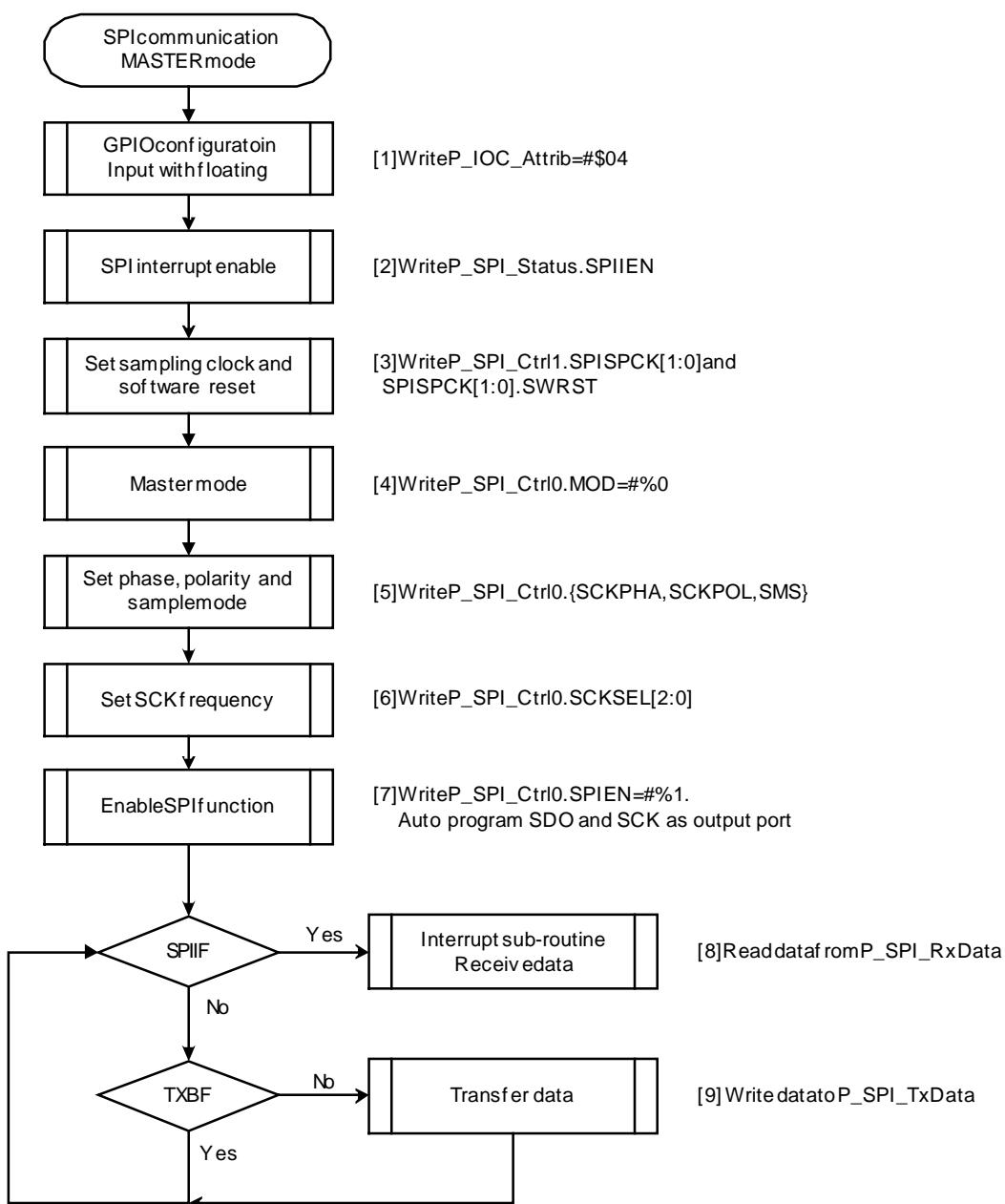


Figure 18-4 SPI Master mode software flowchart

[Example 18-1]: SPI operation on Master Mode (initial)

```

lda      #00000000B
sta      P_IOC_Data
lda      #00000100B          ; PC2 as input with float(SDI)
sta      P_IOC_Attrib
lda      #11111011B          ;
sta      P_IOC_Dir
ldy      #00

```

```

    lda      #(C_SPI_INTEN + C_SPI_INTIF)
    sta      P_SPI_Status           ; enable SPI interrupt, clear interrupt flag

    lda      #(C_SPI_SWRST + C_SPISPC_Div_4)
    sta      P_SPI_Ctrl1          ; sampling clock = Fsys/1, software reset

    lda      #(C_SPI_EN + C_SPI_SCKPHA + C_SPICS_Div_32 + C_SPI_SPISMPS)
    sta      P_SPI_Ctrl0          ; Fsys/32, PHA = 1, master mode

```

18.4.2 Slave Mode

In slave mode, the shifting clock, SCK, comes from external SPI master, so the transmission starts from the first external SCK event. To transmit, the user should write the data to its transmitting buffer before the first SCK comes from the master. Both master and slave devices must be programmed with the same SCK phase and polarity for transmitting and receiving data.

If the clock phase bit (SCKPHA) is “1”, the first data bit to be shifted out starts right after the command written to P_SPI_TxBUF register. If the clock phase bit (SCKPHA) is “0”, the first data bit to be shifted will start after first SCK edge.

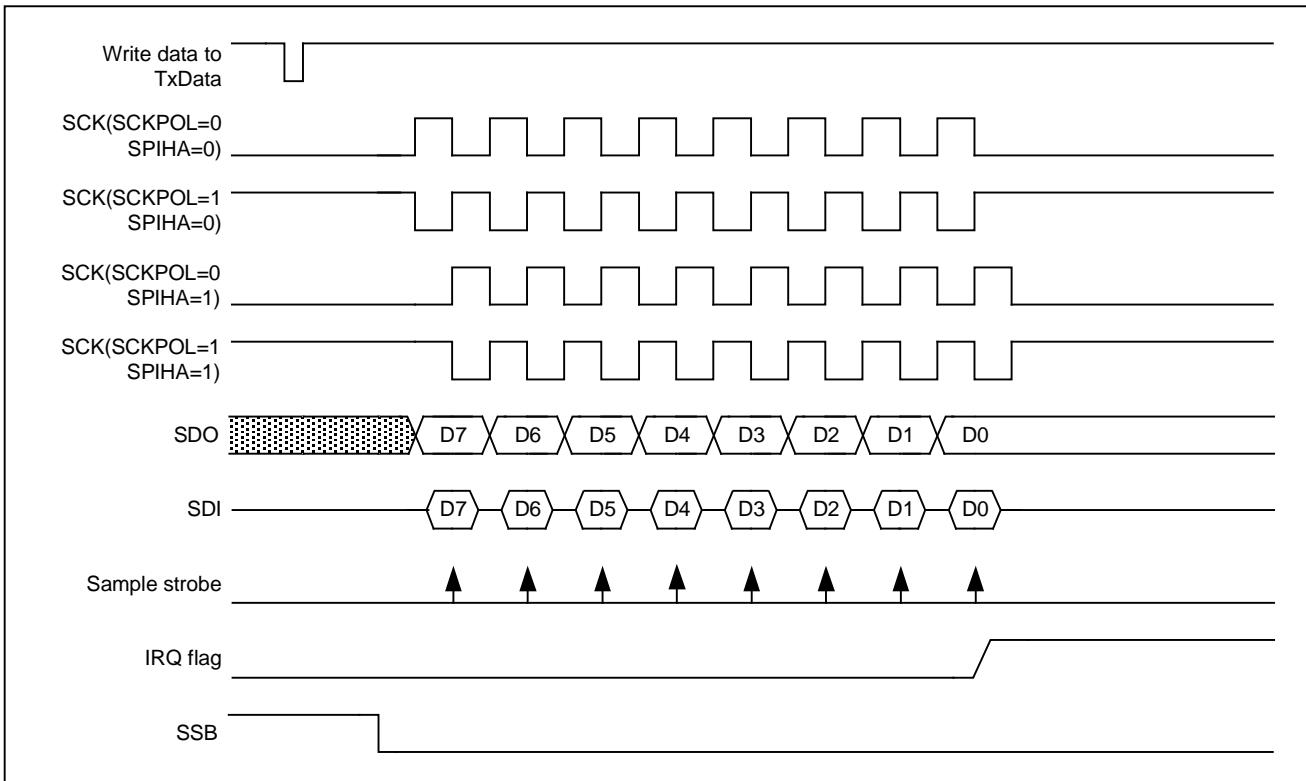


Figure 18-4 SPI Slave mode timing

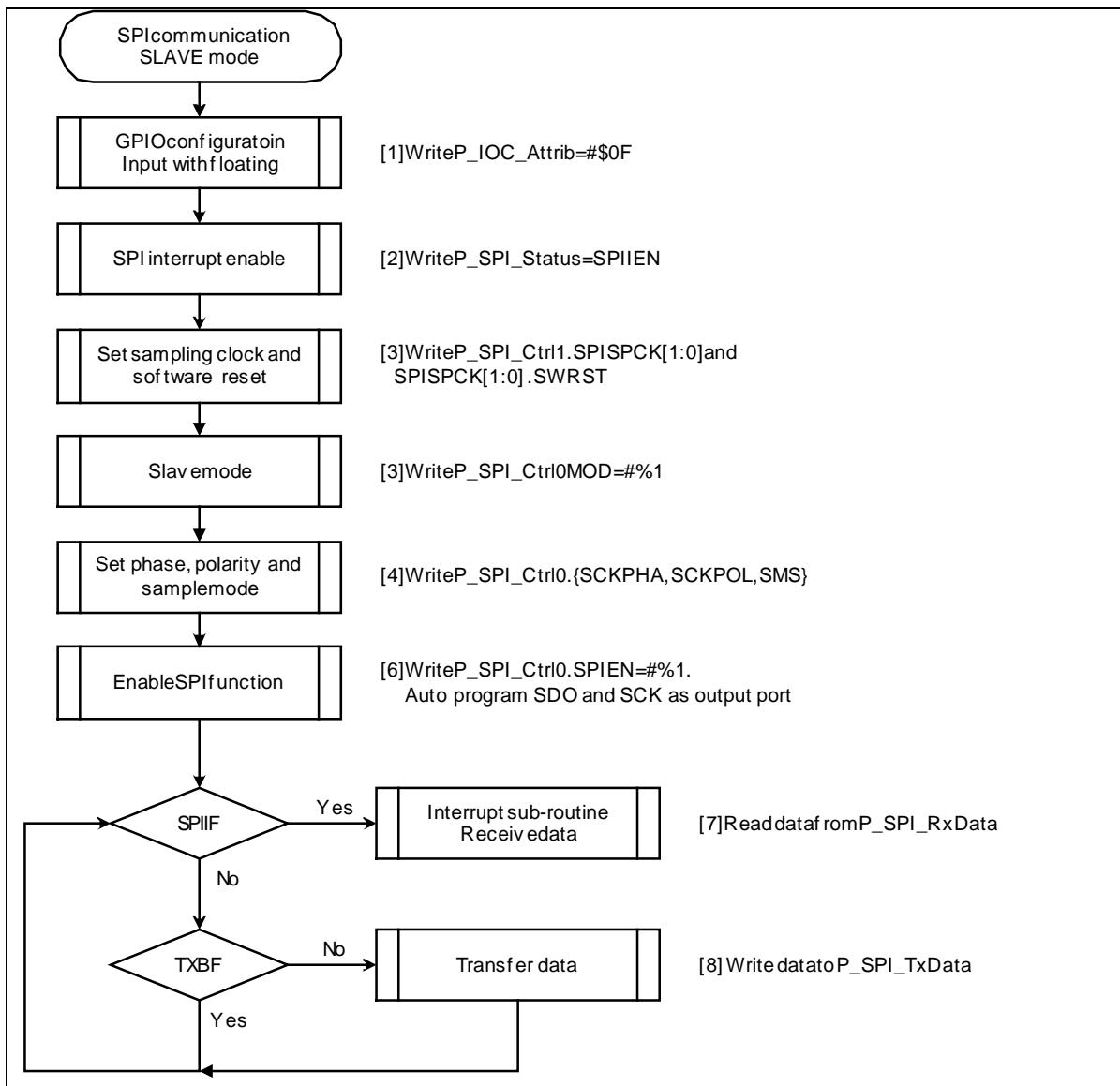


Figure 18-5 SPI Master mode software flowchart

[Example 18-2]: SPI operation on Slave Mode (initial)

```

lda      #00000000B
sta      P_IOC_Data
lda      #00001111B      ; PC0~PC3 as input with float
sta      P_IOC_Attrib
lda      #11110000B      ;
sta      P_IOC_Dir
ldy      #00
    
```

```

lDa      #(C_SPI_INTEN + C_SPI_INTIF)
sta      P_SPI_Status           ; enable SPI interrupt, clear interrupt flag

lDa      #(C_SPI_SWRST + C_SPISPC_Div_4)
sta      P_SPI_Ctrl1            ; sampling clock = Fsys/1, software reset

lDa      #(C_SPI_EN + C_SPI_MOD + C_SPI_SCKPHA + C_SPICS_Div_32 + C_SPI_SPISMPS)
sta      P_SPI_Ctrl0             ; Fsys/32, PHA = 1, slave mode

```

18.5 SPI Interrupt

There are two interrupt sources related with SPI. The SPI transmit interrupt and receive interrupt are occurred in the same time because the SPI is full-duplex function. So there are shared with same interrupt flag SPIIF.

- | Transmit Interrupt
- | Receive Interrupt

There are several steps for enabling SPI interrupt.

1. Interrupt Disable flag (I-flag) is set to "1" by using "SEI" instruction.
2. Enable SPI module and configured some registers related SPI operation.
3. Enable SPI interrupt enable bit SPIIEN in the P_SPI_Status(\$3A) register..
4. Using 'CLI' instruction to enable all interrupt function.
5. Now SPI interrupt will be accepted.

18.6 Design Tips

[Example 18-3]: SPI Formula for Sampling clock selection and I/O setting

1.SPI clock formula

Sampling clock must follow below formula in order to ensure SPI can get data correctly.

Sampling clock > 4 x SPI clock (**Master mode**)

System clock > 4 x SPI clock x SPISPCLK (**Slave mode**)

For example:

SPI Clock =1MHz(PC1), F_{SYS}=8MHz

Sampling clock >= 4X1MHz = 4MHz

Sampling clock must greater than 4Mhz to ensure that slave can receive data correctly.

So, Sampling clock selection bits (SPISPCLK [1:0]) may set to 2.

Sampling clock=8MHz/2=4Mhz

2.Port C setting when SPI operation in Master mode

PC0(SS):Output

PC1(CLK):Output
PC2(SDI):Input with float
PC3(SDO):Output

3.Port C setting when SPI operation in Slave mode

PC0(SS):Input with float
PC1(CLK):Input with float
PC2(SDI): Input with float
PC3(SDO): Input with float

[Example 18-4] : Shows how to initialize SPI as master by polling flag based on SPMC65P2404A, please refer to an example of Fortis IDE.

- Set SPI as master mode
- Send data by Polling .

```

.SYNTAX 6502           ; process standard 6502 addressing syntax
.LINKLIST               ; generate linklist information
.SYMBOLS                ; generate symbolic debug information

.INCLUDE     Spmc65.inc

.PAGE0          ; define values in the range from $00 to $FF
G_Tempbuf1      DS    1    ;
G_Tempbuf2      DS    1    ;
G_MWorkReg1     DS    1    ;
G_MWorkReg2     DS    1    ;
G_TXbuf         DS    7    ;
;
.DATA           ; define data storage section
;
.CODE
; ****
;*
;* Power on Reset Process - Main Program *
;*
; ****

V_Reset:
    sei           ; Disable interrupt
    Idx          #C_STACK_BOTTOM ; Initial stack pointer at $00FF
    txs          ; Transfer to stack point

```

```

;

    lda      #$FF          ; Clear Reset flag
    sta      P_SYS_Ctrl
    sta      P_SYS_Ctrl

;

;     /// Initial Port C      ///

;

    lda      #00000000B
    sta      P_IOC_Data
    lda      #00000100B      ; PC2 as input with float(SDI)
    sta      P_IOC_Attrib
    lda      #11111011B
    sta      P_IOC_Dir
    ldy      #00

    lda      #C_SPI_INTIF
    sta      P_SPI_Status    ; Disable SPI interrupt, clear interrupt flag

    lda      #(C_SPI_SWRST + C_SPISPC_Div_4)
    sta      P_SPI_Ctrl1      ; sampling clock = Fsys/1

    lda      #(C_SPI_EN + C_SPI_SCKPHA + C_SPICS_Div_32 + C_SPI_SPISMPS)
    sta      P_SPI_Ctrl0      ; Fsys/32, PHA = 1, master mode
;

L_Main:
L_TestSPI3LInit:
    ldy      #0

L_TestSPI3L2:
    lda      #$00          ; chip enable
    sta      P_IOC_Data
    lda      #C_SPI_INTIF   ; clear SPI INT flag
    sta      P_SPI_Status
    tya
    sta      P_SPI_TxData   ; send data

L_TestSPI3L3:
    lda      P_SPI_Status
    and      #C_SPI_INTIF   ; SPI INT(TX)send finish ?
    beq      L_TestSPI3L3   ; no
    lda      P_SPI_RxData   ; read data
    sta      G_TXbuf
    sta      P_IOB_Data
    lda      P_IOC_Data      ; chip disable
    eor      #$01

```

```

sta      P_IOC_Data
cpy      #0          ; fisrt data ?
beq      L_TestSPI3L4    ; yes
pla
cmp      G_TXbuf      ; RX data = Previous TX data ?
beq      L_TestSPI3L4    ; yes
lda      #$AA         ; error then break
sta      P_IOA_Data
jmp      L_TestSPI3LInit

L_TestSPI3L4:
tya
pha
jsr      F_Delay100ms
iny
cpy      #00          ; end ?
bne      L_TestSPI3L2    ; no
pla
jmp      L_TestSPI3LInit
;

F_Delay100ms:
lda      #C_WDT_Clr    ; 2c Clear WDT
sta      P_WDT_Clr     ; 3c

F_Delay100ms1:
lda      #$FF          ; 2c (224ms)
sta      G_Tempbuf1    ; 3c
jmp      L_Delay1S_L3

F_Delay1sec:
lda      #C_WDT_Clr    ; 2c Clear WDT
sta      P_WDT_Clr     ; 3c
lda      #$F9          ; 2c (1.55s)
sta      G_Tempbuf1    ; 3c

L_Delay1S_L3:
idx      #0

L_Delay1S_L2:
clc
lda      #0          ; 2c
;

L_Delay1S_L1:
nop
nop
adc      #1          ; 2c
bne      L_Delay1S_L1    ; 3c ((9*256+ 9)*256+ 10)*7+ 5
inx
;
```

```

bne      L_Delay1S_L2          ; 2c
inc      G_Tempbuf1           ; 5c
bne      L_Delay1S_L3          ; 2c
rts

F_Delay50us:
Idx      #200                ; 2c
L_Delay50_L1:
nop                  ; 2c
nop                  ; 2c
inx                  ; 2c
bne      L_Delay50_L1          ; 2c
rts

; ****
; *
; *      Interrupt Vector Table
; *
; *
; ****

VECTOR:   .SECTION
DW       V_NMI                 ; may download program emulated either
DW       V_Reset               ; in internal memory or external memory
DW       V_IRQ                 ; dw define two bytes interrupt vector
; ****
; *
; *      End Of Interrupt Vector Table
; *
; *
; ****

.END      ; end of program

```

18.7 Related Application Notes and Libraries

This section lists some application notes and libraries about various usage of SPMC65X family. The application notes may help user develop quickly and offer the concepts about the usage of this topic. Sunplus also supports some powerful libraries about general operation, mathematics, communication, etc. The purpose of library is to help user speed up development. The current application notes and libraries related to SPI are listed below.

Application Note:

Title	Application Note Series Number
I/O simulate SPI communication	AN_O0310.DOC
Using SPI module	AN_O0342.DOC

Library:



Title	Application Note Series Number
SPMC65X series software basics manual	AN_O0100.DOC
SPMC65X series software advances 1 manual (data processing operation)	AN_O0101.DOC
SPMC65X series software advances 2 manual (mathematical operation)	AN_O0102.DOC

18.8 Revision History

Revision	Date	Remark
V0.1	03/31/2005	First edition

19 UART

19.1 Description

UART function is the abbreviation of Universal Asynchronous Receiver - Transmitter. The UART can be configured as a full duplex asynchronous system that can communicate with peripheral devices, such as personal computers, or it can be configured as a half duplex synchronous system to communicate with peripheral device, such as AD or D/A, etc. The UART module built in SPMC65X family performs serial-to-parallel conversion on data received from an external device and it also performs parallel-to-serial conversion on data transmitted to the external device. The UART Block Diagram is shown in figure 19-1. For a step-by-step procedure on how to set up the UART interrupts, see Section 19.5 in detail. This module provides the following features:

- Two external pins:
 - RXD: data reception pin (shared with PC5)
 - TXD: data transmission pin (shared with PC4)
- Provides standard asynchronous, full-duplex communication
- Programmable trans-receive baud rate
- Parity can be even, odd or disabled for generation and detection
- Stop bit width can be 1 or 2 bits
- Support transmitting interrupt
- Support receiving interrupt
- High noise rejection for bit receiving (majority decision of 3 consecutive samples in the middle of received bit time)
- Framing, Parity error detection during reception.
- Overrun detection
- Programmable baud rate from 2400 bps to 38400 bps @ 8MHz

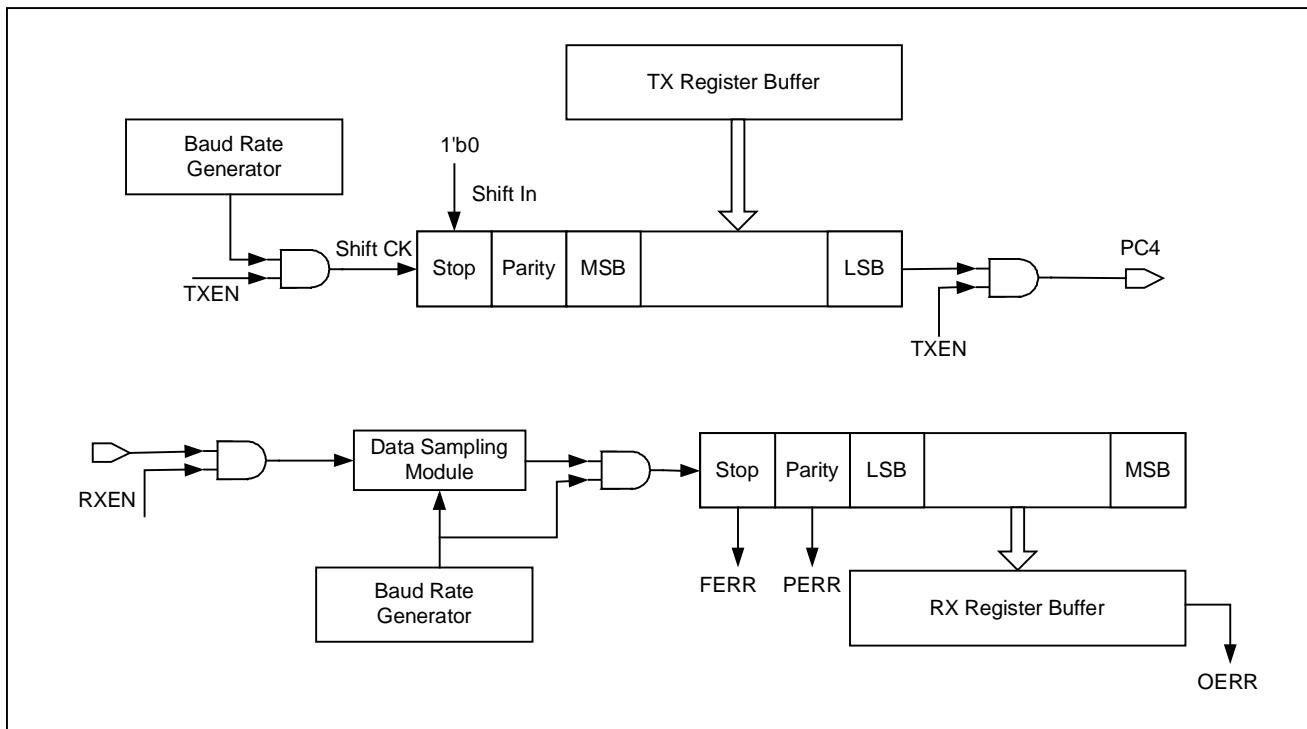


Figure 19-1 UART Block Diagram

Following diagram shows the data format for UART.

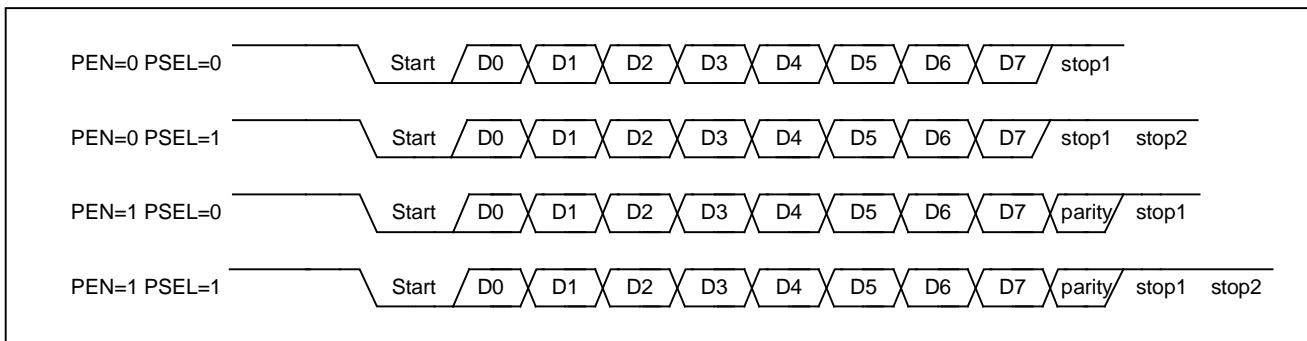


Figure 19-2 Data Format Of UART

19.2 Application Circuit

The UART interface is usually used to communicate with PC computer. The following Figure 19-3 show the application circuit between PC and SPMC65X family MCU by using ICL232 chip.

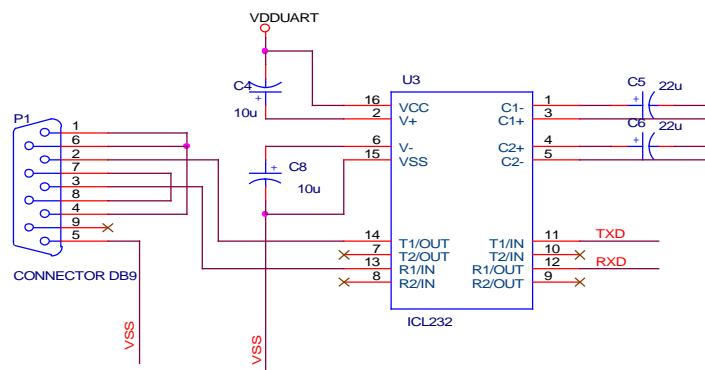


Figure 19-3 UART interface to host application circuit

19.3 Control Register

19.3.1 UART Control Register (P_UART_Ctrl, \$46) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
RXIE	TXIE	RXEN	TXEN	SOFTRST	STOPSEL	PSEL	PEN
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit 7 **RXIE:** Receive Interrupt enable bit

1=Enable receive interrupt request

0=Disable receive interrupt request

Bit 6 **TXIE:** Transmit Interrupt enable bit

1=Enable transmit interrupt request

0=Disable transmit interrupt request

Bit 5 **RXEN:** UART Receive function enable bit

1=Enable receive interrupt request

0=Disable receive interrupt request

Bit 4 **TXEN:** UART transmit function enable bit

1=Enable UART transmit function

0=Disable UART transmit function

Bit 3 **SOFTRST:** Software reset

Write:

1= Reset all UART module

0= No action

Bit 2 **STOPSEL:** Stop bit length selection bit

1=2 Stop bits

0=1 Stop bit

Bit 1 **PSEL:** Parity type selection bit

1=Even parity

0=Odd parity

Bit 0 **PEN:** Parity enable bit

1=Enable parity check or generation

0=Disable parity check or generation

19.3.2 UART Baud Rate Divider Register (P_UART_Baud, \$47) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
BaudRate7	BaudRate6	BaudRate5	BaudRate4	BaudRate3	BaudRate2	BaudRate1	BaudRate0
R/W							
0	0	0	0	0	0	0	0

Bit [7:0] BaudRate[7:0]: UART Baud rate divisor

$$\text{Baud Rate} = \text{F}_{\text{SYS}} / [16 \times (256 - \text{UARTBAUD})]$$

The formula for BAUD Rate is derived from above formula and shown as following:

$$\text{UARTBAUD} = 256 - \text{F}_{\text{SYS}} / (16 \times \text{Baud Rate})$$

Baud Rate (bps)	Suggestion value at 16MHz
2400	\$30
4800	\$98
9600	\$CC
19200	\$E6
38400	\$F3

Note: The resolution can't be tolerated when baud rate is larger than 38400; thus, Keep the baud rate below 38400.

19.3.3 UART Status Register (P_UART_Status, \$48) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
RXIF	TXIF	BUSY	-	-	OERR	PERR	FERR
R	R	R	-	-	R	R	R
0	0	0	0	0	0	0	0

Bit 7 **RXIF**: Receive Interrupt flag bit

1=RX data is ready(full)

0=RX data is not read

Bit 6 **TXIF**: Transmit Interrupt flag bit

1=TX buffer is ready (empty)

0=TX buffer is full

Bit 5 **BUSY**: UART transmission in progress flag bit

1=Transmission is progress

0=Transmission is finished and UART wait for next transmission.

Bit [4:3] Reserved

Bit 2 **OERR**: Overrun Error flag bit

Read

1=Overrun error occurred

0=No overrun error

Bit 1 **PERR**: Parity Error flag bit

Read:

1=Parity error occurred

0=No Parity error

Bit 0 **FERR**: Frame Error flag bit

Read:

1=Frame error occurred

0=No frame error

19.3.4 UART Transmit or Receive Data Register (P_UART_Data, \$49) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
DATA7	DATA6	DATA5	DATA4	DATA3	DATA2	DATA1	DATA0
R/W							
0	0	0	0	0	0	0	0

Bit [7:0] **UARTDATA** [7:0]: UART transmit/receive data

Read: Get received data and clear RXIF

Write: Push the transmission data to buffer and clear TXIF

19.4 Operation

The UART function has Transmit/receive status in P_UART_Status register.. While UART interface is received one byte successfully, the received data will be latched into received buffer. At that time, RXIF bit in P_UART_Status register will be set and a UART interrupt will be issued to CPU if the RXIE bit in the P_UART_Ctrl register is set. RXIF bit in P_UART_Status register would be cleared automatically when reading received data. TXIF flag is set to '1' means the transfer is complete and a UART interrupt will be issued to CPU if the TXIE bit in the P_UART_Ctrl register is set. TXIF bit in P_UART_Status register would be cleared automatically when writing data to P_UART_Data register. BUSY flag is a read only flag. When Busy flag is set to '1' means CPU is busy in transmission. If the transfer is complete, the BUSY flag bit is clear. OERR, PERR and FERR are error bits for user to check whether some communication errors occurred. Each error flag will be cleared automatically when the error doesn't occur on next communication. Figure19-4 and Figure19-5 show parity and Frame timing. Note that, the received data byte must be read first from P_UART_Data before reading the corresponding error status from P_UART_Status. This read sequence cannot be reversed since the status register P_UART_Status is updated only when a read operation is performed on P_UART_Data control register.

There exists a baud rate register and an 8-bit timer to generate the baud rate. Each time the timer increases from its maximum count (\$FF), 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 with the value in baud rate register.

$$\text{Baud Rate} = \text{F}_{\text{sys}} / [16 \times (256 - \text{UARTBAUD})]$$

The value in baud rate register is taken as an 8-bit unsigned number. To derive the required baud rate register values from a known baud rate, use the equation:

UARTBAUD=256 - $F_{SYS} / (16 \times \text{Baud Rate})$

The UART begins transmitting after the first the rollover of the divide-by-16 counter after the software writes to the P_UART_Data register. The UART transmits data on the TX pin in the following order: start bit, 8 data bits (LSB first), parity bit (Parity Enable mode only) and stop bit. The TXIF bit in P_UART_Status register is set after 2 CPU clock cycles when the stop bit is transmitted.

Reception begins at the falling edge of a start bit received on RX pin, when enabled by the RXEN bit in P_UART_Ctrl register. For this purpose, RX is sampled 16 times per bit for any baud rate. When a falling edge of a start bit is detected, the divide-by-16 counter used to generate the receiving clock is reset to align the counter rollover to the bit boundaries.

For noise rejection, the serial port establishes the content of each received bit by a majority decision of 3 consecutive samples in the middle of each bit time. This is especially true for the start bit. If the falling edge on RX is not verified by a majority decision of 3 consecutive samples logic low level, then the serial port stops reception and waits for another falling edge on RX.

At the middle of the stop bit time, the serial port checks for the following conditions:

RXIF = 0 and RXIE = 1. If the above conditions are met, the serial port then writes the received byte to the P_UART_Data register and set the RXIF bit. If the above condition are not met, the received data is lost and would not be loaded into the P_UART_Data . After the middle of the stop bit time, the serial port waits for another high-to-low transition on the RX pin.

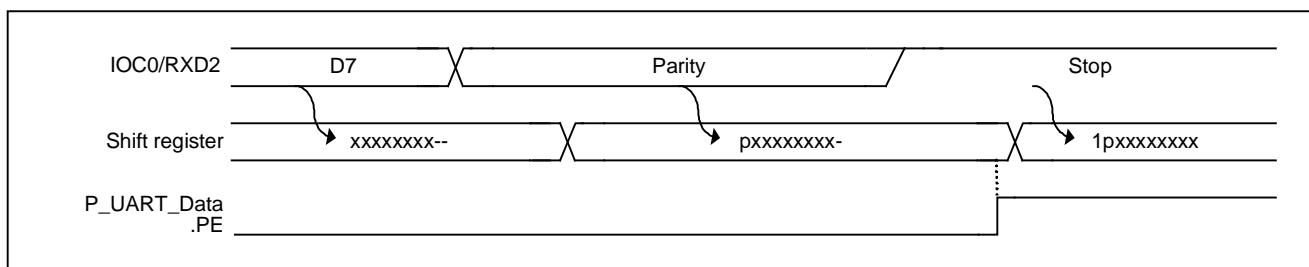


Figure19-4 Parity error timing

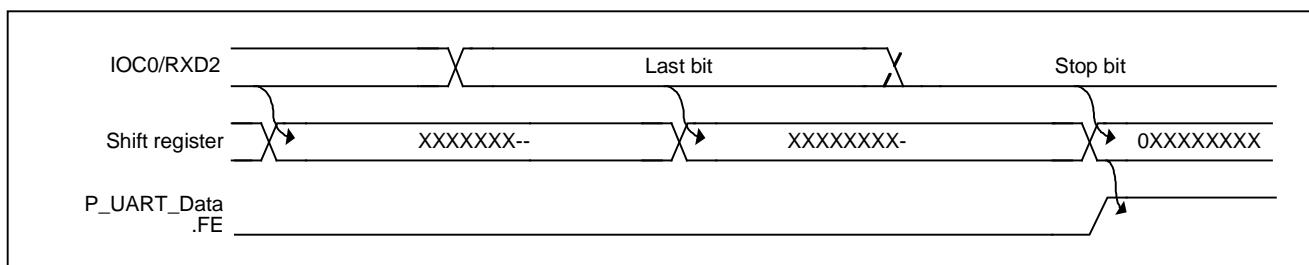


Figure19-5 Frame error timing

19.5 UART Interrupt

Interrupts can come from many sources. There are two interrupt sources related with UART.

- | Transmit Interrupt
- | Receive Interrupt

There are several steps for enabling UART interrupt.

1. Interrupt Disable flag (I-flag) is set to “1” by using “SEI ” instruction.
2. Configured some registers related for UART operation.
3. Deciding Transmit/Receive interruptions would be used and then enabling its interrupt enable bits in the P_UART_Ctrl(\$46) register.
4. Using ‘CLI’ instruction to enable all interrupt function.
5. Now Transmit /Receive interrupt will be accepted.

[Example 19-1] Initialize UART (Transceiver) by interrupt

```

lda      #C_UART_SOFTRST
sta      P_UART_Ctrl           ; perform software reset for UART
lda      #(C_UART_TXEN + C_UART_TXIE)
sta      P_UART_Ctrl           ; UART transmission enable and enable TXINT
lda      #(C_UART_OERR + C_UART_PERR + C_UART_FERR)
sta      P_UART_Status         ; clear UART error flag

```

19.6 Design Tips

[Example 19-2] : Shows how to initialize UART(Transceiver) by polling flag based on SPMC65P2408A, please refer to an example of Fortis IDE.

- n Send data by polling method

```

.SYNTAX 6502                  ; process standard 6502 addressing syntax
.LINKLIST                         ; generate linklist information
.SYMBOLS                           ; generate symbolic debug information

.INCLUDE    Spmc65.inc

        .PAGE0                 ; define values in the range from $00 to $FF
;
        .DATA                  ; define data storage section
;
        .CODE
;
*****
```

```

;*
;* Power on Reset Process - Main Program *
;*
;. ****
;V_Reset:
    sei          ; Disable interrupt
    ldx #C_STACK_BOTTOM ; Initial stack pointer at $00FF
    txa          ; Transfer to stack point
;
    lda #$FF      ; Clear Reset flag
    sta P_SYS_Ctrl
    sta P_SYS_Ctrl
;
    jsr F_InitIOPort ; initial GPIO port
;
    lda #$00
    sta P_IOC_Data
    sta P_IOC_Attrib
;
    lda #11011111B ; PC5(Rx) as input, PC4(Tx) as output
    sta P_IOC_Dir
;
    lda #C_UART_SOFTRST
    sta P_UART_Ctrl ; perform software reset for UART
    lda #C_UART_TXEN
    sta P_UART_Ctrl ; UART transmission enable
;
    lda #(C_UART_OERR + C_UART_PERR + C_UART_FERR)
    sta P_UART_Status ; clear UART error flag
;
    lda #$CC
    sta P_UART_Baud ; P_UART_Baud $CC -> 9600 bps
;
    ldx #0
;

L_TxReady:
    lda P_UART_Status
    and #C_UART_TXIF ; check if tx ready
    beq L_TxReady
;
    lda Hello, X
    beq L_exitTx
;
    sta P_UART_Data
;
```

```

inx
lda    #(C_UART_OERR + C_UART_PERR + C_UART_FERR)
sta    P_UART_Status           ; clear UART error flag
jmp    L_TxReady              ; loop again
;
L_exitTx:
L_Main:
nop
jmp    L_Main
;
Hello: .BYTE    'Hello World!', 0Ah, 0Dh, 00h

; *****
; *          *
; *      Interrupt Vector Table      *
; *          *
; *****
VECTOR:   .SECTION
DW      V_NMI                 ; may download program emulated either
DW      V_Reset                ; in internal memory or external memory
DW      V_IRQ                  ; dw define two bytes interrupt vector
; *****
; *          *
; *      End Of Interrupt Vector Table  *
; *          *
; *****
.END

```

19.7 Related Application Notes and Libraries

This section lists some application notes and libraries about various usage of SPMC65X family. The application notes may help user develop quickly and offer the concepts about the usage of this topic. Sunplus also supports some powerful libraries about general operation, mathematics, communication, etc. The purpose of library is to help user speed up development. The current application notes and libraries related to UART are listed below.

Application Note:

Title	Application Note Series Number
I/O simulate UART communication	AN_O0312.DOC
Using UART module	AN_O0344.DOC

Library:



Title	Application Note Series Number
SPMC65X series software basics manual	AN_O0100.DOC
SPMC65X series software advances 1 manual (data processing operation)	AN_O0101.DOC
SPMC65X series software advances 2 manual (mathematical operation)	AN_O0102.DOC

19.8 Revision History

Revision	Date	Remark
V0.1	03/31/2005	First edition

20 IIC

20.1 Description

IIC function is the abbreviation of Inter-Integrated Circuit. The multi-master I2C-bus controller provides a mechanism to communicate between bus masters and peripheral devices by using two signals, a serial data line (SDA) and a serial clock line (SCL). To avoid all possibilities of confusion, data loss and blockage of information, the master and slave devices must have a defined protocol. In multi-master I2C-bus mode, multiple microprocessors can receive or transmit serial data from slave devices. The master that initiates a data transfer over the I2C-bus is responsible for terminating the transfer. It is possible to combine several masters, in addition to several slaves onto an I2C-bus to form a multi-master system. If more than one master simultaneously tries to control the line, an arbitration procedure decides which master gets priority. The maximum number of devices connected to the bus is dictated by the maximum allowable capacitance on the lines, 400 pF, and the protocol's addressing limit of 16 K.

Two external pins:

SDA: Data Input/Output pin (shared with PC7)

SCL: Clock pin (shared with PC6)

- | Master transmit/receive mode supported.
- | Slave transmit/receive mode supported.
- | Detection of bus arbitration fail.
- | Interrupt generation.
- | Programmable ACK generation.
- | Programmable clock speed in master mode.

20.2 Application Circuit

The IIC interface is usually used to communicate with EEPROM. The Figure 20-1 shows the application circuit between 24C01 and SPMC65X family MCU.

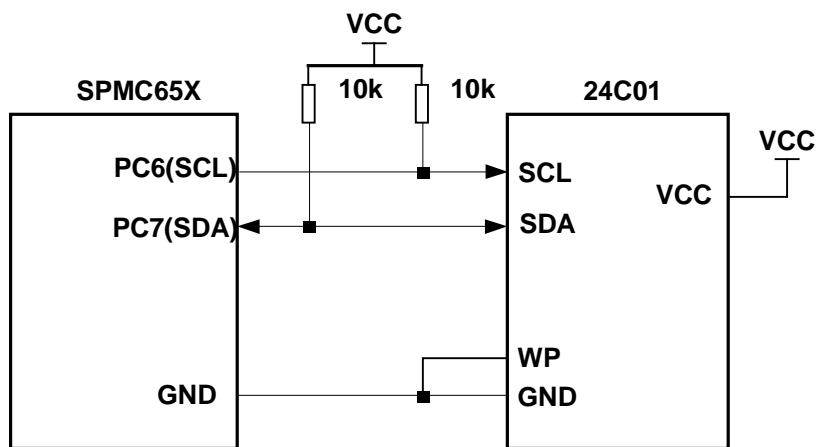


Figure 20-1 Application Circuit for IIC Operation

20.3 Control Register

20.3.1 IIC Bus Control Register (P_IIC_Ctrl, \$4A) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
IICEN	-	TXRXEN	IICIEN	ACKEN	SCK2	SCK1	SCK0
R/W	-	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit 7 **IICEN**: IIC series interface enable bit

1=Enable IIC series interface

0=Disable IIC series interface

Bit 6 Reserved

Bit 5 **TXRXEN**: IIC bus data output enable bit

1=Enable

0=Disable

Bit 4 **IICIEN**: IIC interrupt bit

1=Enable

0=Disable

Bit 3 **ACKEN**: IIC acknowledge enable bit

1=Enable ACK generation

0=Disable ACK generation

Bit [2:0] **SCK[2:0]:** IIC clock rate selection bit

111= $F_{SYS} \div 1024$

110= $F_{SYS} \div 512$

101= $F_{SYS} \div 256$

100= $F_{SYS} \div 128$

011= $F_{SYS} \div 64$

010= $F_{SYS} \div 32$

001= $F_{SYS} \div 16$

000= $F_{SYS} \div 8$

F_{SYS} : Frequency of System Clock

20.3.2 IIC Bus Status Register (P_IIC_Status, \$4B) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
MXT	TXRXSEL	BUSY/SIGEN	IICIF	ARBITRAT	AAS	AZERO	LRB
R/W	R/W	R/W	R/W	R/W	R	R	R
0	0	0	0	0	0	0	0

Bit 7 **MXT:** IIC Master/Slave mode selection bit

1=Master mode

0=Slave mode

Bit 6 **TXRXSEL:** TX/RX selection bit

1=Transmitter mode

0=Receiver mode

Bit 5 **BUSY/SIGEN:** Bus busy bit or start/stop signal generation bit

Read

1=Bus is busy

0=Bus is ready

Write

1=START signal generation

0=STOP signal generation

Note:

1.While writing MXT and TXRXSEL, keep this bit =0.

2.While clearing INTIF flag, i.e. IICF=1, device will automatically ignore this bit setting.

Bit 4 **IICIF:** IIC interrupt flag bit

1=IIC interrupt flag on

0=IIC interrupt flag off

Bit 3	ARBITRAT: IIC arbitration procedure status flag bit 1=Bus arbitration status failed during communication 0=Bus arbitration status okay
Bit 2	AAS: IIC address-as-slave status flag bit 1=Received slave address matches the address value in the IIC address register. 0=START/STOP condition was generated
Bit 1	AZERO: IIC address zero status flag bit 1=Received slave address is "\$00" 0=START/STOP condition was generated
Bit 0	LRB: IIC last-received bit status flag bit 1=Last-received bit is "1" (ACK was not received) 0=Last-received bit is "0" (ACK was received)

20.3.3 IIC Bus Data Register (P_IIC_Data, \$4C) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
DATA7	DATA6	DATA5	DATA4	DATA3	DATA2	DATA1	DATA0
R/W							
0	0	0	0	0	0	0	0

Bit [7:0] **IICDATA [7:0]:** IIC Bus data register

Read: Receiving data

Write: Transmission data

20.3.4 IIC Bus Address Register (P_IIC_Address, \$4D) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
ADDR7	ADDR6	ADDR 5	ADDR 4	ADDR 3	ADDR 2	ADDR 1	ADDR 0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit [7:0] **IICADDRESS [7:0]:** IIC Bus address register

Slave mode: Slave address

20.4 Operation

The I²C-bus interface I of each device has an address. When a master wishes to initiate a data transfer, first ,it transmits the address of the device that is wishes to “talk” to. All device “listen” to see if this is their address. Within this address, a bit specifies if the master wishes to read-from/write-to the slave device. The master and slave are always in opposite mode (transmitter/receiver) operation during a data transfer. That is they can be thought of as operating in either of these two relations:

- | Master –transmitter and Slave-receiver
- | Slave-transmitter and Master-receiver

In both cases the master generates the clock signal.

The output stages of the clock (SCL) and data (SDA) lines are open-drain in order to perform the wired-AND function in the bus. External pull-up resistors are used to ensure a high level when no device is pulling the line down.

The below protocol is normal used for 24C01:

Start Bit	Device Address	R/W	ACK	Data	ACK	...	Stop Bit
		← 7 bit →	1 bit	1 bit	← 8 bit →	1 bit	

Figure 20-2 24C01 protocol

20.4.1 Initializing and Terminating Data Transfer

During time of no data transfer (idle time), both the clock line (SCL) and the data line (SDA) are pulled high through the external pull-up resistors. The START and STOP signals determine the start and stop of data transmission. The START signal is defined as a high to low transition of the SDA when the SCL is high. The STOP signal is defined as a low to high transition of the SDA when the SCL is high.

Figure 20-3 is shown the START and STOP signals. The master generates these signals for starting and terminating data transfer. Due to definition of the START and STOP signals, when data is being transmitted, the SDA line can only change state when the SCL line is low.

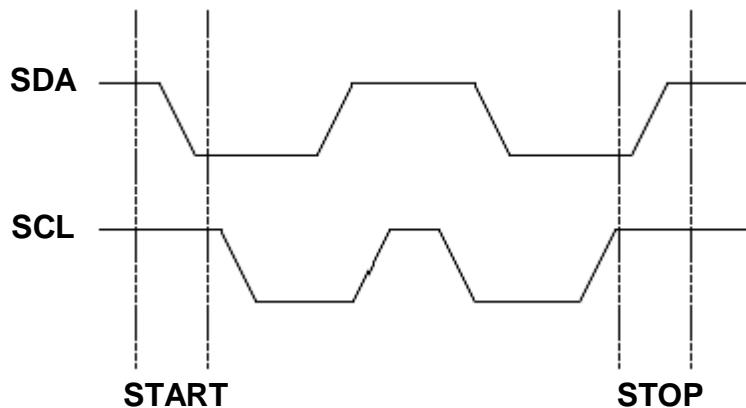


Figure 20-3 START and STOP signals

20.4.2 Addressing IIC Devices

The SPMC65x family devices only support the 7-bit address format with a R/W bit.

Figure 20-4 is shown the 7-bit address format.

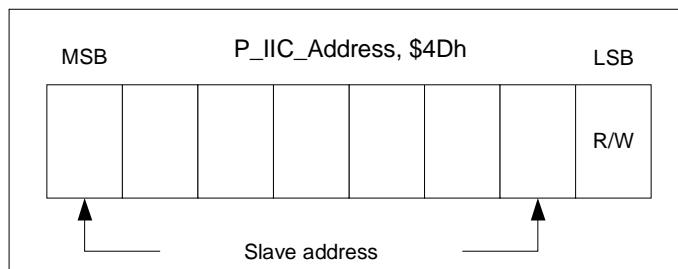


Figure 20-4 7-bit address format

20.4.3 Transfer Acknowledge

All data must be transmitted per byte, with no limit to the number of bytes transmitted per data transfer. After each byte, the slave-receiver generates an acknowledge signal (ACK).

When a slave-receiver doesn't acknowledge the slave address or received data, the master must abort the transfer. The slave must leave SDA high so that the master can generate the STOP signal.

Figure 20-5 is shown IIC bus ACK signal.

If the master is receiving the data (master-receiver), it generates an acknowledge signal for each received byte of data, except for the last byte. To output the end of data to the slave-transmitter, the master does not generate an acknowledge signal (not ACK). The slave then releases the SDA line so the master can generate the STOP signal. The master can also generate the STOP signal during the acknowledge pulse for valid termination of the data transfer.

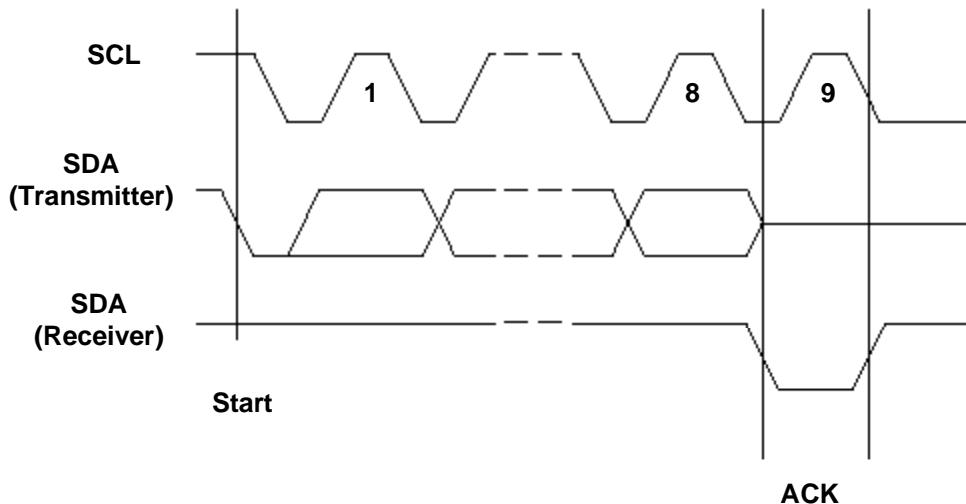


Figure 20-5 IIC bus ACK signal

20.4.4 General Call Address Support

The address procedure for the IIC bus is such that the first byte after the START signal usually determines which device will be the slave addressed by the master. The exception is the general call address which can address all device. When this address is used, all devices should, in theory, respond with an acknowledge signal.

The general call address is one of eight addresses reserved for specific purpose by the IIC protocol. It consists of all "0" with R/W=0.

The general call address is recognized automatically. When recognized the general call address that then set the AZERO bit.

20.4.5 Bus Arbitration

Multi-Master mode is achieved by bus arbitration. When the master outputs address/data bits onto the SDA pin, arbitration takes place when the master outputs a '1' on SDA by letting SDA float high and another master asserts a '0'. Then the ARBITRAT flag bit will be set.

20.4.6 Slave mode

In slave mode operation, the SCL (PC6) and SDA (PC7) pins must be configured as inputs. The IIC module will over-ride the input state with the output data when required (slave-transmitter).

When address is matched with the P_IIC_Address register or the data transfer after an address match is received, the IIC module automatically will generate the acknowledge signal by setting ACKEN= 1, and then load received data in the P_IIC_Data register.

The setting slave mode sequence as blow:

- I Set PC6 and PC7 as input with float.
- I Enable IIC module (IICEN= 1)

- | Enable IIC data output enable bit (TXRXEN= 1)
- | Set slave-receiver mode (MXT= 0, TXRXSEL= 0)

A typical receiving sequence would go as follows:

- | Set slave device address in P_IIC_Data.
- | Wait one byte data received ready by checking IICIF bit.
- | Load the received data from P_IIC_Data when master device is set write mode by TXRXSEL=0.
- | Set transmitted data in P_IIC_Data when master device is set read mode by TXRXSEL=1.
- | Clear IICIF bit.

20.4.7 Master mode

In master mode operation, the SCL (PC6) and SDA (PC7) pins must be configured as inputs. The IIC module will over-ride the input state with the output data when required (master-transmitter).

The IIC clock (SCL) is generated from Master device, so SPMC65x family devices support 8 programmable clock rates and up to 1MHz speed @ 8MHz by SCK[2:0].

The setting master mode sequence as blow:

- | Set PC6 and PC7 as input with float.
- | Enable IIC module (IICEN= 1)
- | Enable IIC data output enable bit (TXRXEN= 1)
- | Set master-transmitter mode (MXT= 1, TXRXSEL= 1)

A typical transmit sequence would go as follows:

- | Set slave device address in P_IIC_Data.
- | Waiting IIC bus is become ready by checking BUSY flag.
- | Generate a START signal by setting the SIGGEN bit. (Begin to send the P_IIC_Data.)
- | Wait one byte data transmitted ready by checking IICIF bit.
- | Load the next data in P_IIC_Data.
- | Clear IICIF bit.
- | Generate a STOP signal by setting the SIGGEN bit.

[Example 20-1]: Set IIC as Master Mode

```

lda      #(C_IIC_IICEN + C_IIC_INTEN + C_IIC_TXRXEN + C_IIC_ACKEN + C_IIC_SCK_128)
sta      P_IIC_Ctrl           ; I2C enable, data output enable and ACK generation
                           ; I2C clock = Fsys / 128
lda      #(C_IIC_MXT + C_IIC_TXRXSEL + C_IIC_INTIF)
sta      P_IIC_Status        ; master mode, TX mode, clear IICIF flag

```

20.5 IIC Interrupt

The following events will cause the IIC interrupt flag bit, IICIF, to be set:

- | Data transfer byte transmitted
- | Data transfer byte received

There are several steps for enabling IIC interrupt.

1. Interrupt Disable flag (I-flag) is set to "1" by using "SEI" instruction.
2. Configured some registers related for IIC operation.
3. Enable IIC interrupt enable bit IICEN in the P_IIC_Ctrl(\$4A) register.
4. Using 'CLI' instruction to enable all interrupt function.
5. Now IIC interrupt will be accepted.

20.6 Design Tips

[Example 20-3] : Shows how to initialize IIC with 24C02, please refer to an example of Fortis IDE.

- n Send data to by polling method
- n Data are (\$AA,\$A5,\$5A)

```
.SYNTAX 6502 ; process standard 6502 addressing syntax
.LINKLIST ; generate linklist information
.SYMBOLS ; generate symbolic debug information
.INCLUDE Spmc65.inc
.PAGE0 ; define values in the range from $00 to $FF
G_Tempbuf1 DS 1
G_Tempbuf2 DS 1
G_TXbuf DS 7
;
.DATA ; define data storage section
;
.CODE
; ****
;*
;* Power on Reset Process - Main Program *
;*
; ****
V_Reset:
```

```

sei                                ; Disable interrupt
Idx      #C_STACK_BOTTOM          ; Initial stack pointer at $00FF
txs                                ; Transfer to stack point
;
lda      #$FF                      ; Clear Reset flag
sta      P_SYS_Ctrl
sta      P_SYS_Ctrl
;
; I2C master mode setup
lda      #00000000B
sta      P_IOC_Data
lda      #11000000B
sta      P_IOC_Attrib
lda      #00000000B                ; PC6(SDL), PC7(SDA) as input float
sta      P_IOC_Dir
;
lda      #(C_IIC_IICEN + C_IIC_TXRXEN + C_IIC_ACKEN + C_IIC_SCK_128)
sta      P_IIC_Ctrl               ; I2C enable, data output enable and ACK generation
                                ; I2C clock = Fsys / 128
;
lda      #(C_IIC_MXT + C_IIC_TXRXSEL + C_IIC_INTIF)
sta      P_IIC_Status             ; master mode, TX mode, clear IICIF flag
;
;
L_Main:
L_I2C_Func:
    lda      #%"10100000
    sta      P_IIC_Data            ; send slave address
;
L_TestI2CL20:
    lda      P_IIC_Status
    and      #C_IIC_BUSY          ; bus is busy ?
    bne      L_TestI2CL20        ; yes
;
    lda      P_IIC_Status
    ora      #%"00100000
    sta      P_IIC_Status         ; make 'START' signal
;
    idx      #0
L_TestI2CL21:
    lda      P_IIC_Status
    and      #C_IIC_INTIF        ; INTIF come ?
    beq      L_TestI2CL21        ; no
;

```

```

L_TestI2CL22:
    lda      T_I2C_Data, X           ; Sending data
    sta      P_IIC_Data

    lda      P_IIC_Status
    ora      #C_IIC_INTIF          ; clear INTIF(sending)
    sta      P_IIC_Status
    inx
    cpx      #4                   ; send 4 bytes finish ?
    bne      L_TestI2CL21         ; no
;

L_TestI2CL3:
    lda      P_IIC_Status
    and      #C_IIC_INTIF          ; INTIF come ?
    beq      L_TestI2CL3          ; no

L_TestI2CL31:
    lda      #%11000000          ; make 'STOP' signal
    sta      P_IIC_Status
    jsr      F_Delay100ms         ; wait for eeprom write time (10ms)

    lda      #(C_IIC_IICEN + C_IIC_TXRXEN + C_IIC_ACKEN + C_IIC_SCK_128)
    sta      P_IIC_Ctrl

    lda      #(C_IIC_MXT + C_IIC_TXRXSEL + C_IIC_INTIF)
    sta      P_IIC_Status

    lda      #$A0                 ; slave address
    sta      P_IIC_Data

    lda      #%11100000          ; write, start signal
    sta      P_IIC_Status
    nop

L_TestI2CL4:
    lda      P_IIC_Status
    and      #C_IIC_INTIF          ; INTIF come ?
    beq      L_TestI2CL4          ; no
;
    lda      T_I2C_Data           ; I2C address
    sta      P_IIC_Data
    lda      P_IIC_Status
    ora      #C_IIC_INTIF          ; clear INTIF(sending)
;
```

```

sta      P_IIC_Status
L_TestI2CL41:
    lda      P_IIC_Status
    and     #C_IIC_INTIF          ; INTIF come ?
    beq     L_TestI2CL41          ; no

    lda      #%10100000          ; I2C slave address
    sta      P_IIC_Data

    lda      #%10100000          ; read, start signal
    sta      P_IIC_Status

L_TestI2CL42:
    lda      P_IIC_Status
    and     #C_IIC_INTIF          ; INTIF come ?
    beq     L_TestI2CL42          ; no
;

    lda      P_IIC_Status
    ora     #C_IIC_INTIF          ; clear INTIF(sending)
    sta      P_IIC_Status

    idx      #0

L_TestI2CL43:
    lda      P_IIC_Status
    and     #C_IIC_INTIF          ; INTIF come ?
    beq     L_TestI2CL43          ; no
;

    lda      P_IIC_Data
    sta      G_TXbuf,X
;

    inx
    cpx      #3                  ; received end ?
    beq     L_TestI2CL45          ; yes
    cpx      #2                  ; received last one ?
    bne     L_TestI2CL44          ; no
    lda      #(C_IIC_IICEN + C_IIC_TXRXEN)
    sta      P_IIC_Ctrl           ; I2C enable, no ACK, output enable
;

L_TestI2CL44:
    lda      P_IIC_Status
    ora     #C_IIC_INTIF          ; clear INTIF(sending)
    sta      P_IIC_Status
    jmp     L_TestI2CL43          ;

```

```
L_TestI2CL45:  
    lda      #%"10000000          ; stop signal  
    sta      P_IIC_Status  
;  
    jmp      L_I2C_Func  
;  
  
T_I2C_Data:  
    DB      #$52                ; Word address  
    DB      #$AA                ; data 1  
    DB      #$A5                ; data 2  
    DB      #$5A                ; data 3  
;  
.*  
;*****  
;*      Interrupt Vector Table      *  
;*  
;*****  
;VECTOR:      .SECTION  
    DW      V_NMI               ; may download program emulated either  
    DW      V_Reset              ; in internal memory or external memory  
    DW      V_IRQ                ; dw define two bytes interrupt vector  
;  
.*  
;*****  
;*      End Of Interrupt Vector Table      *  
;*  
;*****  
.END           ; end of program  
;
```

20.7 Related Application Notes and Libraries

This section lists some application notes and libraries about various usage of SPMC65X family. The application notes may help user develop quickly and offer the concepts about the usage of this topic. Sunplus also supports some powerful libraries about general operation, mathematics, communication, etc. The purpose of library is to help user speed up development. The current application notes and libraries related to IIC are listed below.

Application Note:

Title	Application Note Series Number
I/O simulate I2C communication	AN_O0311.DOC
Using I2C module	AN_O0343.DOC

Library:

Title	Application Note Series Number
SPMC65X series software basics manual	AN_O0100.DOC
SPMC65X series software advances 1 manual (data processing operation)	AN_O0101.DOC
SPMC65X series software advances 2 manual (mathematical operation)	AN_O0102.DOC

20.8 Revision History

Revision	Date	Remark
V0.1	03/31/2005	First edition

21 Power Saving Mode

21.1 Description

In SPMC65X family, the devices provide three kinds of operation modes, which are NORMAL mode, STOP mode and HALT mode for operation. Figure 21-1 shows the Power Saving Mode Operation. When a device is power up, the system always starts from reset and operates in normal mode in beginning. User can transfer the operating mode to STOP or HALT mode from normal mode. After entering STOP or HALT mode, the system can only back to normal mode. It means if user wants to change from HALT mode to STOP mode, the system has to change from HALT mode to normal mode, then changes from normal mode to STOP, and vice versa.

When operating in normal mode, the device consumes the maximum power and all peripherals can be used. According to needs of application,

These two modes can be used to reduce the power consumption for a large amount. The wake up time on the two modes are different though. User has to choose a suitable mode for their application. To transfer into STOP mode or HALT mode, user must double-write corresponding value to Mode Control Register. Detailed information about power saving mode are discussed in the following sections.

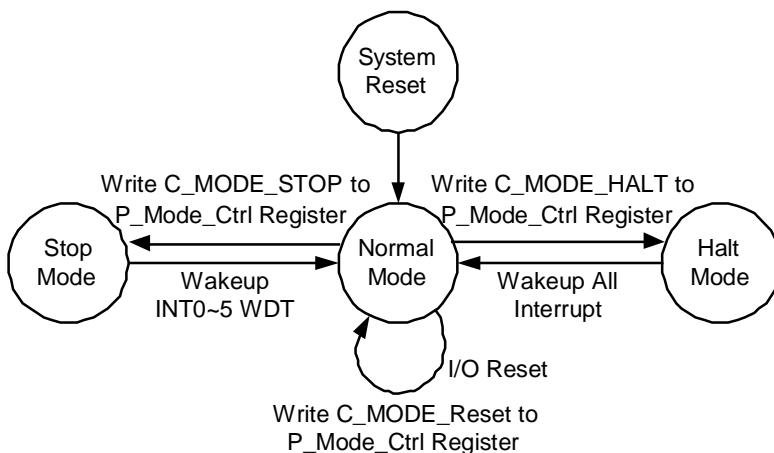


Figure 21-1 Power Saving Mode Operation

Operating Modes	Index Access Register (\$31)	CPU	Peripheral
Stop	#\$5A	OFF	OFF
Halt	#\$A5	OFF	ON
High Speed (Normal)	-	ON	ON

Table 21-1 The relationship between mode and operation

Table 21-1 shows the relationship between mode and operation.

In STOP mode, the whole system clock will be disabled, including the clock generating circuit. Once the system entered the STOP mode, only the activated specific interrupt events (from I/Os) can recover the normal operation from STOP mode. Be aware that if user set the watchdog interrupt event as wake up source, the power will be larger than disable the watchdog.

In order to wait for the clock source to stable, when the system receives the interrupt event, the system will delay for a certain time. This delay time is about 40ms in typical and will range from 20ms to 60ms. After the clock is stable, if user enable the interrupt exception (clear the interrupt flag in status register by CLI), the program will jump into exception rountin immediately, otherwise, the program will start from the next instruction of the STOP mode command. To enable the specific interrupt events could wake up the system, the corresponding interrupt enable bits must be set before entering the STOP mode.

If user uses the Watchdog interrupt to wake up the system, the on-chip watchdog RC oscillator clock must be activated before entering STOP mode. Note that enabling RC oscillator may cause a little power consumption. In stop mode, if user needn't use watch dog as wake up trigger source and has power saving consideration, it is necessary to disable RC oscillator by clearing SCKEN bit in the P_WDT_Ctrl register for saving power consumption. Detailed examples are shown in design tips. Besides, to use Watch-Dog to recover the system, user must set the Watch-Dog interrupt rate properly to prevent system reset during the clock stable time. In HALT mode, CPU and memory clock will be disabled. The periphery clock remains active so that the periphery will keep running in HALT mode.

All interrupt sources will recover the normal operation from HALT mode. If user enables the interrupt exception (clear the interrupt flag in status register by CLI), the program will jump into exception rountin immediately, otherwise, the system will run from the next instruction of the HALT mode command immediately. In Halt mode, there is no delay time from the wakeup event to CPU active.

21.2 Control Register

21.2.1 Power Saving Mode Control Register (P_Mode_Ctrl, \$31) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
Mode_Ctrl7	Mode_Ctrl6	Mode_Ctrl5	Mode_Ctrl4	Mode_Ctrl3	Mode_Ctrl2	Mode_Ctrl1	Mode_Ctrl0
R/W							
0	0	0	0	0	0	0	0

Bit [7:0] **Mode_Ctrl[7:0]: Operation Mode control register**

Read:

Data is always \$00

Write (twice):

\$5A=enter Stop mode (C_MODE_STOP = \$5A)

\$A5=enter HALT mode (C_MODE_HALT = \$A5)

\$66=reset all internal modules, except CPU (C_MODE_Reset = \$66)

Note: This byte must be double-write.

21.2.2 Watch Dog Timer Control Register (P_WDT_Ctrl, \$32) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
SCKEN	WDCS2	WDCS1	WDCS0	-	-	-	-
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
1	1	1	1	-	-	-	-

Bit 7 **SCKEN:** Slow Clock Enable bit in Stop mode

1= Enable Slow Clock in stop mode

0= Disable Slow Clock in stop mode

Bit [6:4] **WDCS:** Watch Dog interrupt Clock rate Selection bits

000 = f_slow/128

001 = f_slow/256

010 = f_slow/512

011 = f_slow/1024

100 = f_slow/2048

101 = f_slow/4096

110 = f_slow/8192

111 = f_slow/16384

f_slow: internal build-in RC oscillator, typical frequency is 25kHz

Bit [3:0] Reserved

Note: All of above bits need write twice to set corresponding ones.

21.3 Operation

21.3.1 STOP mode

In STOP mode, the on-chip oscillator is stopped, but the on-chip RAM and Control registers are held. The I/O ports maintain the status that they had before the stop instruction was executed. Oscillator stops and the system operations are all held up.

Figure 21-2 shows the timing diagram of entering STOP Mode. The STOP mode is activated by setting value of P_Mode_Ctrl by “STOP” command, “C_MODE_STOP”-- “\$5A”. Before executing “C_MODE_STOP” command, user must clear all interrupt request flags and enable interrupt sources that wanted to be served as waking up sources. Because if the interrupt request flags are set before “C_MODE_STOP” command, the MCU runs as if it doesn’t perform “C_MODE_STOP” command, even though the “C_MODE_STOP” command is executed. So

inserting instructions to clear all interrupt request flags (P_INT_Flag0, P_INT_Flag1, P_INT_Flag2) is necessary before “STOP” command.

After executing “C_MODE_STOP” command, at least two or more NOP instruction should be added for program reliability. Note that if user needn’t use watch dog as wake up source, WDTEN bit in the P_WDT_Ctrl register should be set to ‘0’ in stop mode to stop the on-chip RC oscillator for purpose of power saving.

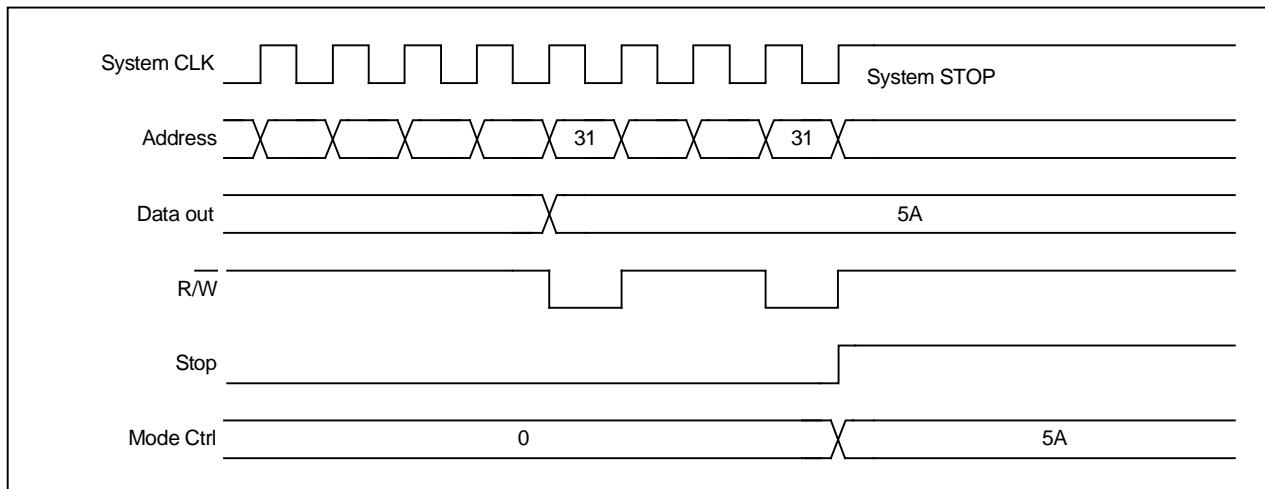


Figure 21-2 Enter STOP Mode

[]Example 21-1] : Set STOP mode

```

lда      #$FF
ста      P_INT_Flag0           ; clear INT request flag
ста      P_INT_Flag1
сет      P_INT_Ctrl0, CB_INT_IRQ0E ; enable IRQ0 INT (INT0)for wake up source
кли          ; enable INT
lда      #C_MODE_STOP         ; STOP command
ста      P_MODE_Ctrl
ста      P_MODE_Ctrl
ноп          ; Must wait 2 NOPs
ноп
    
```

21.3.2 Release the STOP mode

There are several ways to wake up the MCU from STOP mode such as hardware reset or external interrupt. The hardware reset re-defines all the control registers but does not change the on-chip RAM. External interrupts allow both on-chip RAM and control register to retain their values. To enable the specific interrupt events could wake up the system when specific interrupt comes, but the corresponding interrupt enable bits must be set before entering

the STOP mode.

After releasing from STOP mode, instruction execution is divided into two ways according to I-flag in status (P). If I-flag=0, the normal interrupt would be occurred. If I-flag=1, the chip will resume execution starting with the following the STOP instruction. In other words, MCU would not execute interrupt service routine. Detail release flow was shown in Figure 21-3.

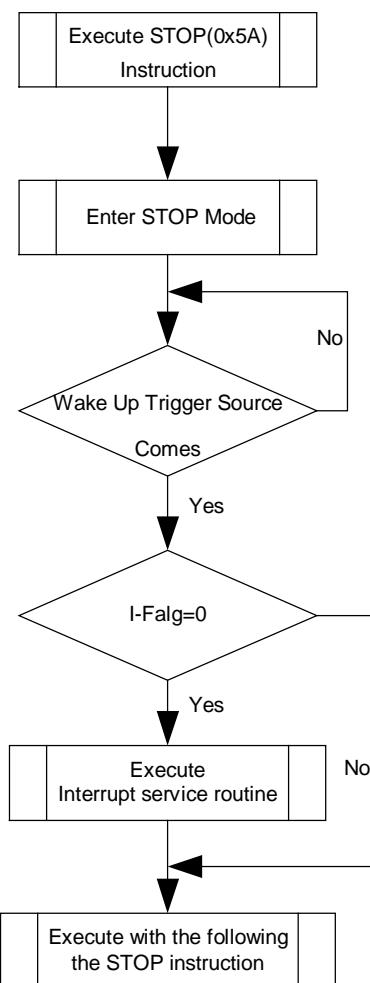


Figure 21-3 Release Flow From STOP Mode

When exiting from Stop mode by external interrupt, it must take a little time for oscillation stabilization. This delay time is about 40ms in typical and will range from 20ms to 60ms. Figure 21-4 shows the Timing of STOP Mode and Release.

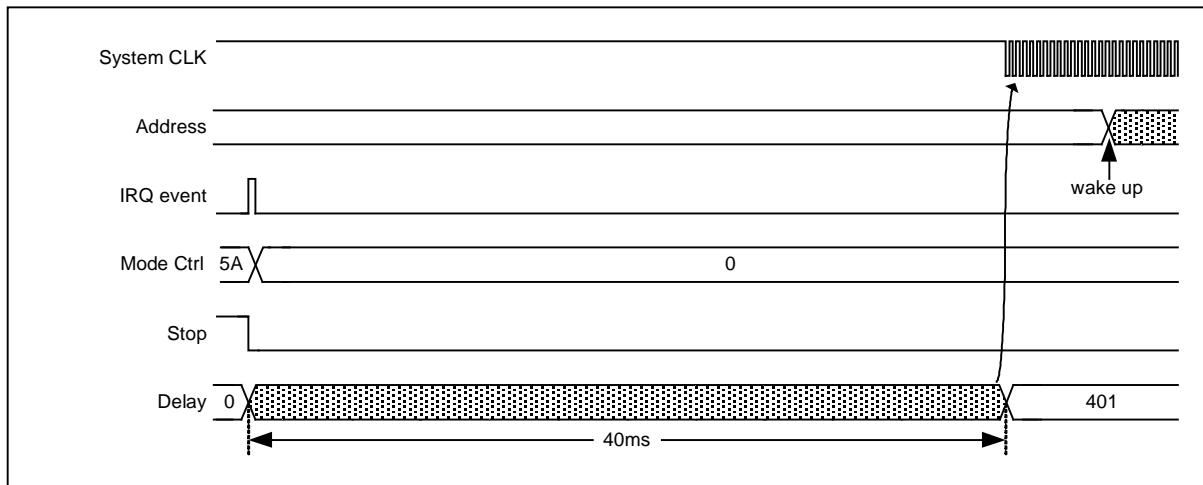


Figure 21-4 Leave STOP mode by External Interrupt

If user uses the Watchdog interrupt to wake up the system, the on-chip watchdog RC oscillator clock must be activated before entering STOP mode. Besides, to use Watch-Dog to recover the system, user must set the Watch-Dog interrupt rate properly to prevent system reset during the clock stable time, as shown in Figure 21-5.

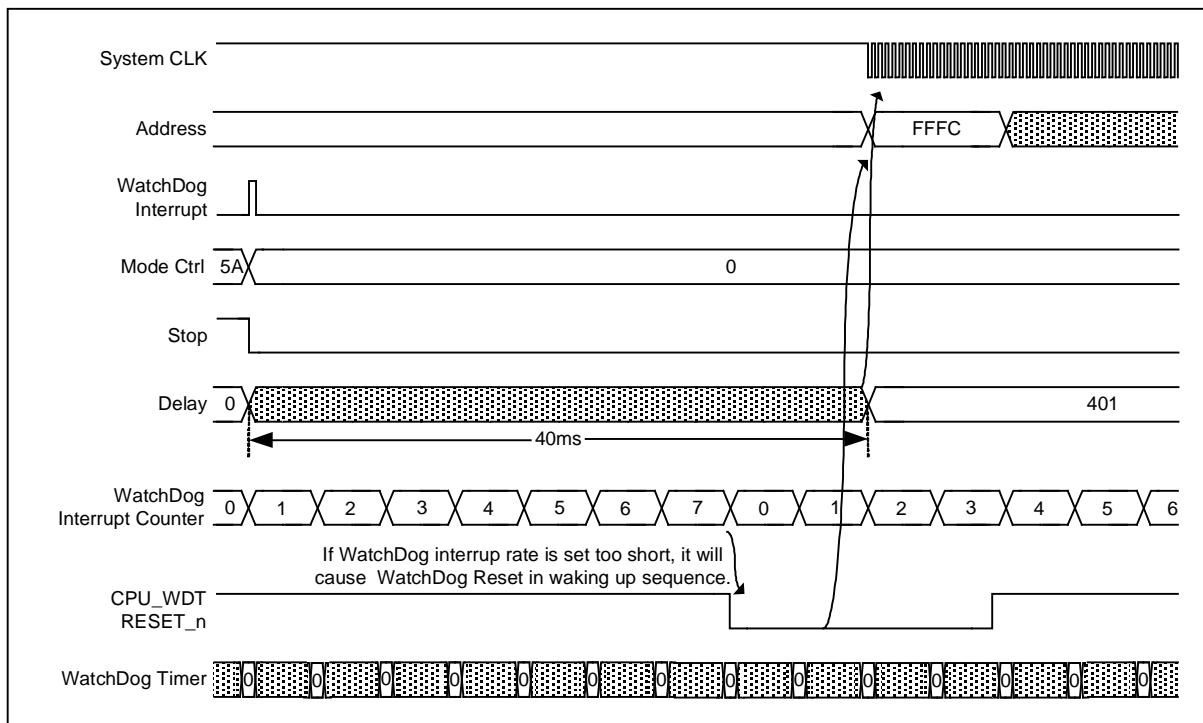


Figure 21-5 Leave STOP mode by WatchDog with error interrupt rate.

21.3.3 Halt mode

Figure 21-6 shows the timing diagram of entering HALT Mode. In Halt mode, all peripherals keep their previous state and are still operating. Only CPU is halted to decrease power consumption. The Halt mode will be entered by setting value of P_Mode_Ctrl by “Halt” command, “C_MODE_HALT”--“\$A5”. Before executing “C_MODE_HALT” command, user must clear all interrupt request flags and enable interrupt sources that wanted to be served as waking up sources. Because if the interrupt request flags are set before “C_MODE_HALT” command, the MUC runs as if it doesn’t perform “C_MODE_HALT” command, even though the “C_MODE_HALT” command is executed. So inserting instructions to clear all interrupt request flags (P_INT_Flag0, P_INT_Flag1, P_INT_Flag2) is necessary before “C_MODE_HALT” command.

After “C_MODE_HALT” command, at least two or more NOP instruction should be added for program reliability.

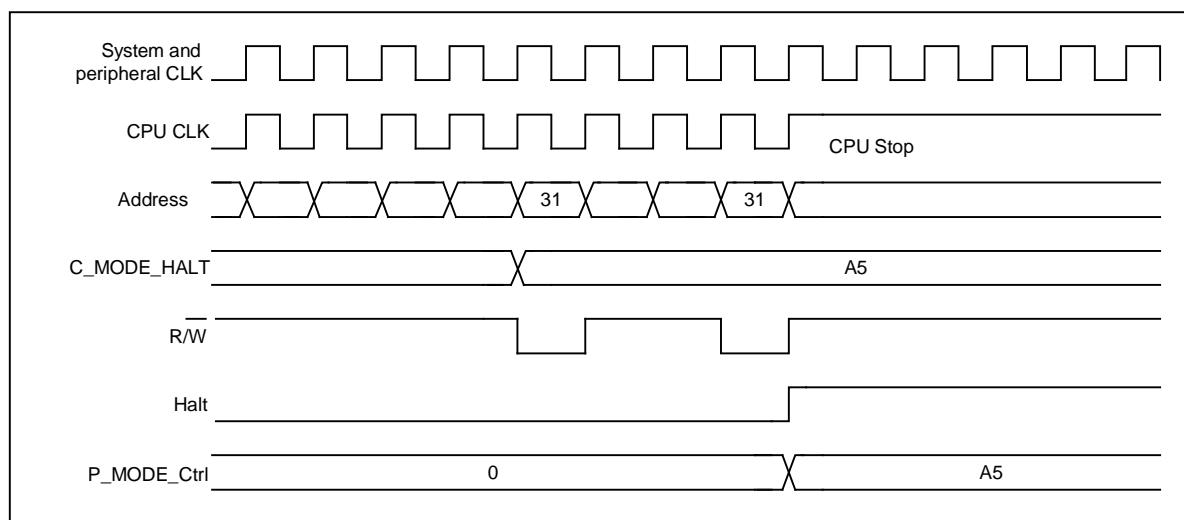


Figure 21-6 Enter HALT mode

Please refer to [Example 21-2]. It shows how to set HALT mode.

[Example 21-2] : Set HALT mode

```

    lda      #$FF          ; clear INT request flag
    sta      P_INT_Flag0
    sta      P_INT_Flag1
    set      P_INT_Ctrl0, CB_INT_IRQ0E ; enable IRQ0 interrupt
    cli      ; enable INT
;
    lda      #C_MODE_HALT
    sta      P_MODE_Ctrl
    sta      P_MODE_Ctrl      ; enter HALT mode
    nop
    nop          ; must wait 2 NOPs

```

21.3.4 Release the Halt mode

All of interrupt sources can wake up the MCU from Halt mode, such as External interrupt, Timer overflow interrupt, Watch Dog overflow interrupt, ADC converter interrupt, Capture input interrupt, SPI Interrupt and UART interrupt. Reset re-defines all the control registers but does not change the on-chip RAM. External interrupts allow both on-chip RAM and control register to retain their values. To enable the specific interrupt events could wake up the system, the corresponding interrupt enable bits must be set before entering the HALT mode.

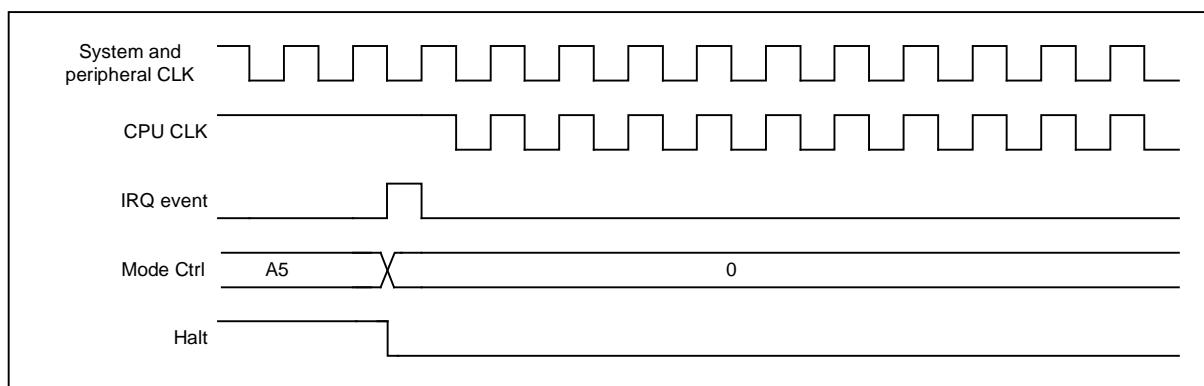


Figure 21-7 Leave HALT mode from interrupt

Refer to the timing diagram of leaving HALT mode in Figure 21-7. After releasing HALT mode, instruction execution is divided into two ways according to I-flag in status (P). If I-flag=0, the normal interrupt would be occurred. If I-flag=1, the chip will resume execution starting with the following the HALT instruction. In other words, MCU will not execute interrupt service routine. Detailed release flow is shown in Figure 21-8. When exiting from HALT mode by interruption, it must take a little time for oscillation stabilization.

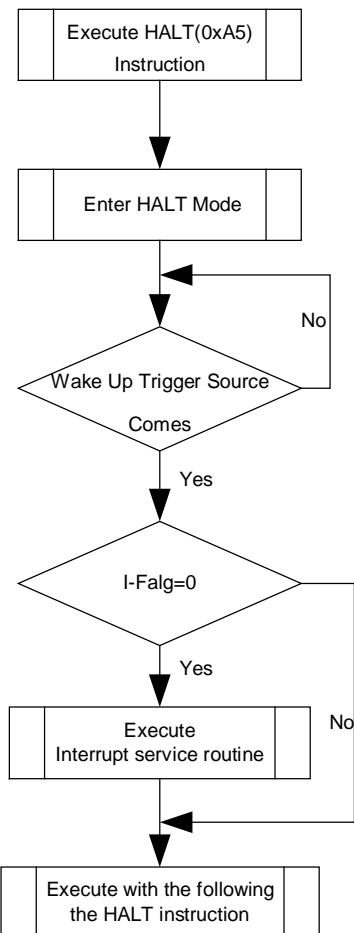


Figure 21-8 Release Flow From HALT Mode

21.3.5 Minimizing Current Consumption

The STOP mode is designed to reduce power consumption. To minimize current drawn during STOP mode, the user should turn-off output drivers that are sourcing or sinking current, if it is practical. It should be set properly that current flow through port doesn't exist. First, consider the setting port to input mode. Be sure that there is no current flow after considering its relationship with external circuit. In input mode, the pin impedance viewing from external MCU is very high that the current doesn't flow. Figure 21-89 shows application example of unused input port. But input voltage level should be Vss or Vdd. Be careful that if unspecified voltage, i.e. uncertain voltage level (not Vss or Vdd) is applied to input pin, there can be little current (max. 1mA at around 2V) flow. If it is not appropriate to set port as an input mode, then set to output mode considering there is no current flow. Setting to High or Low is decided considering its relationship with external circuit. For example, if there is external pull-up resistor then it is set to output mode, i.e. to High, and if there is external pull-down resistor, it is set to Low. Figure

21-10 shows application example of unused output and Input ports.

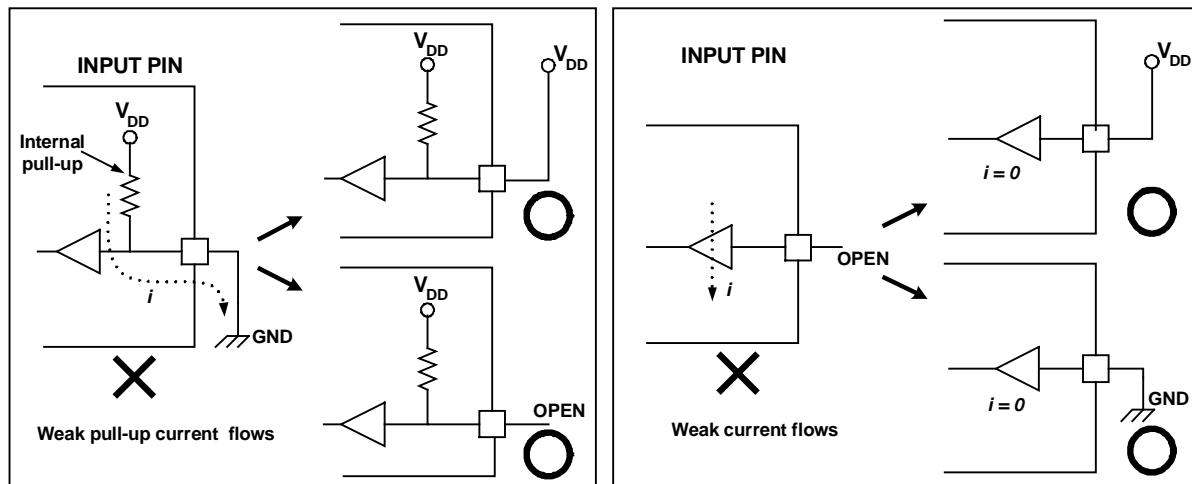


Figure 21-9 Application Example Of Unused Input Port

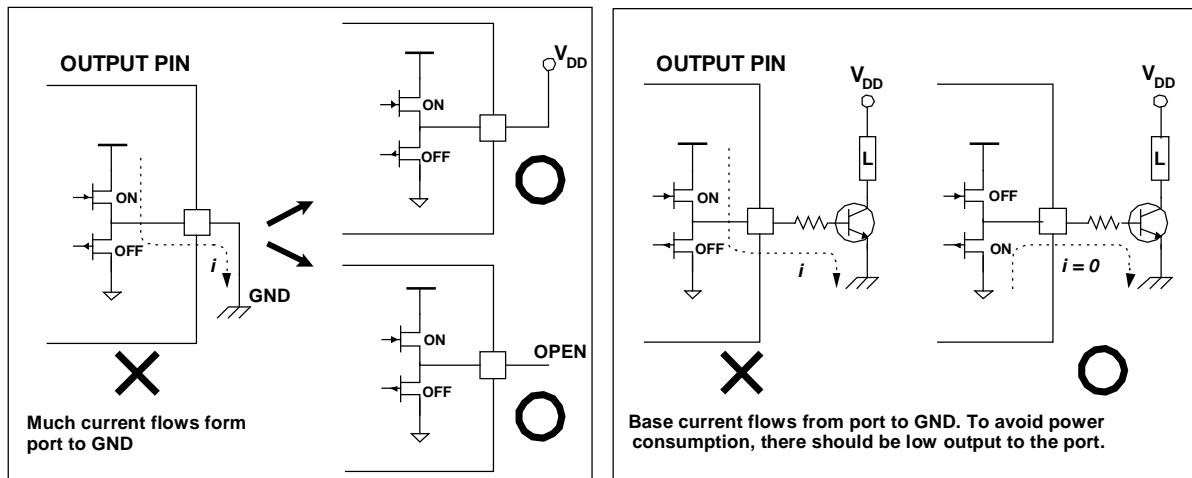


Figure 21-10 Application Example Of Unused Output Port

21.4 Power Saving Mode Interrupt

There is no interrupt source for Power Saving Mode use.

21.5 Design Tips

[Example 21-3]: Shows how to enter stop mode and using IRQ0 to wake up based on SPMC65P2404A, please

refer to an example of Fortis IDE.

- MCU entered STOP mode
- Enable IRQ0 as wake up source
- Disable Slow Clock in stop mode for purposes of power saving consideration

```
.SYNTAX 6502 ; process standard 6502 addressing syntax
.LINKLIST ; generate linklist information
.SYMBOLS ; generate symbolic debug information

.INCLUDE      spmc65p2404a.inc

.PAGE0          ; define values in the range from $00 to $FF
;
;           .DATA          ; define data storage section
;
;           .CODE
; ****
;*
;* Power on Reset Process - Main Program *
;*
; ****

V_Reset:
    sei          ; Disable interrupt
    ldx #C_STACK_BOTTOM ; Initial stack pointer at $00FF
    txa          ; Transfer to stack point
;
    lda #$FF      ; Clear Reset flag
    sta P_SYS_Ctrl
    sta P_SYS_Ctrl
;
    lda #$00      ; Set IOB0 to low
    sta P_IOB_Attrib
;
    lda #00010000B ; Set IOB1 to high
    sta P_IOB_Data
;
    lda #11101111B ; Set IOB2 to high
    sta P_IOB_Dir      ; set PB4 as input high for INT0
;
    clr P_WDT_Ctrl,7 ; Disable slow clock (25Khz) for saving power consumption
    clr P_WDT_Ctrl,7
    lda #$00
    sta P_IRQ_Opt1
    sta P_IRQ_Opt1      ; IRQ0 is falling edge trigger
```

```

;
    lda      #$FF           ; clear INT request flag
    sta      P_INT_Flag0
    sta      P_INT_Flag1
    lda      #C_INT IRQ0E   ; enable IRQ0 interrupt
    sta      P_INT_Ctrl0
    cli      ; enable INT
;
L_Main:
    lda      #C_MODE_STOP
    sta      P_MODE_Ctrl
    sta      P_MODE_Ctrl     ; enter STOP mode
    nop
    nop          ; must wait 2 NOPs

    jmp      L_Main
;
;
; *****
; *          *
; *      IRQ Interrupt Service Routine      *
; *          *
; ;
; *****

V_IRQ:
    pha      ; push A register
    txa      ; transfer X to A
    pha      ; push A register (ie. push X)
;

; IRQ0 interrupt ?
;

    lda      P_INT_Flag0
    and      #C_INT IRQ0F
    beq      L_exit_irq
;

    set      P_INT_Flag0, CB_INT IRQ0F
;

L_exit_irq:
    pla      ; pop A register
    tax      ; transfer A to X
    pla      ; pop A register (ie. pop X)
;

    rti
;
;
; *****
; *          *
;

```

```
;;
;          Interrupt Vector Table      *
;          *                                *
; ****
;
VECTOR:      .SECTION
    DW      V_NMI           ; may download program emulated either
    DW      V_Reset          ; in internal memory or external memory
    DW      V_IRQ            ; dw define two bytes interrupt vector
;
; ****
;
;          *                                *
;          End Of Interrupt Vector Table   *
;          *                                *
; ****
;
.END         ; end of program
```

[Example 21-4]: Shows how to enter stop mode based on SPMC65P2404A, please refer to an example of Fortis IDE.

- MCU entered STOP mode
- Enable Watch Dog as wake up source

```
.SYNTAX  6502           ; process standard 6502 addressing syntax
.LINKLIST                      ; generate linklist information
.SYMBOLS             ; generate symbolic debug information

.INCLUDE    spmc65p2404a.inc

.PAGE0                ; define values in the range from $00 to $FF
;

.DATA                 ; define data storage section
;

.CODE
;
; ****
;
;          *                                *
;          Power on Reset Process - Main Program  *
;          *                                *
; ****
;

V_Reset:
    sei                  ; Disable interrupt
    ldx      #C_STACK_BOTTOM ; Initial stack pointer at $00FF
    txa                  ; Transfer to stack point
;
    lda      #$FF           ; Clear Reset flag
    sta      P_SYS_Ctrl
    sta      P_SYS_Ctrl
```

```

;
    lda      #C_WDT_Div_512          ; Watchdog interrupt clock rate = F_slow/512 = 48.828 Hz
    sta      P_WDT_Ctrl
    sta      P_WDT_Ctrl
;
    lda      #$FF
    sta      P_INT_Flag0           ; clear INT request flag
    sta      P_INT_Flag1
    set      P_INT_Ctrl0, CB_INT_WDIE ; Enable WDT INT

L_Main:
    lda      #C_MODE_STOP
    sta      P_MODE_Ctrl
    sta      P_MODE_Ctrl          ; enter STOP mode
    nop
    nop                      ; must wait 2 NOPs

    jmp      L_Main
;

; *****
; *          *
; *      IRQ Interrupt Service Routine      *
; *          *
; *****
V_IRQ:
    pha      ; push A register
    txa      ; transfer X to A
    pha      ; push A register (ie. push X)
;
; WDT timer overflow interrupt ?
;
    lda      P_INT_Flag0
    and      #C_INT_WDIF
    beq      L_exit_irq
;
    set      P_INT_Flag0, CB_INT_WDIF
;
    lda      P_IOA_Data
    eor      #$FF
    sta      P_IOA_Data          ; toggle P_IOA_Data
;
L_exit_irq:
    pla      ; pop A register
    tax      ; transfer A to X
    pla      ; pop A register (ie. pop X)
;
```

```

;
      rti;
; ****
;*          *
;*      Interrupt Vector Table      *
;*          *
; ****
; VECTOR:      .SECTION
      DW      V_NMI           ; may download program emulated either
      DW      V_Reset          ; in internal memory or external memory
      DW      V_IRQ            ; dw define two bytes interrupt vector
;
; ****
;*          *
;*      End Of Interrupt Vector Table   *
;*          *
; ****
; .END          ; end of program

```

21.6 Related Application Notes and Libraries

This section lists some application notes and libraries about various usage of SPMC65X family. The application notes may help user develop quickly and offer the concepts about the usage of this topic. Sunplus also supports some powerful libraries about general operation, mathematics, communication, etc. The purpose of library is to help user speed up development. The current application notes and libraries related to Power saving mode are listed below.

Application Note:

Title	Application Note Series Number
Power saving mode	AN_O0346

Library:

Title	Application Note Series Number
SPMC65X series software basics manual	AN_O0100.DOC
SPMC65X series software advances 1 manual (data processing operation)	AN_O0101.DOC
SPMC65X series software advances 2 manual (mathematical operation)	AN_O0102.DOC

21.7 Revision History

Revision	Date	Remark
V0.1	03/31/2005	First edition

22 Development Tool

22.1 Description

Sunplus is devoted to provide the product development engineer with a high reliability and user-friendly design environment for SPMC65X family microcontrollers. This chapter describes all kinds of development tools used for SPMC65X Family MCU. The development tools include FortisIDE™, software language, in-circuit emulator (ICE), and application notes.

22.2 Fortis IDE

Sunplus FortisIDE™ provides a friendly and convenient developed environment for user, and it allows you to program, debug, and integrate SPMC65X family to emulate different processors easily . Sunplus FortisIDE™ is a SPMC65 CPU core integrated development environment that features project management, text editing, and program compile, and debug capabilities. The user-friendly interface, pull-down menu, short-cut command and fast-access command list will assist you to write and debug program easily and efficiently.

22.2.1 Key Feature

- Built-in assembler and linker
- Integrated program editor
- Project management
- Easy resource files management
- Text search / replace in an opening document / disk file
- Built-in debugger (ICE)
- Software & hardware break point
- Debug function (step into, step over, step out, run to cursor)
- Register / flag status
- Memory status
- Disassemble capability
- Environment customization

22.2.2 Installation

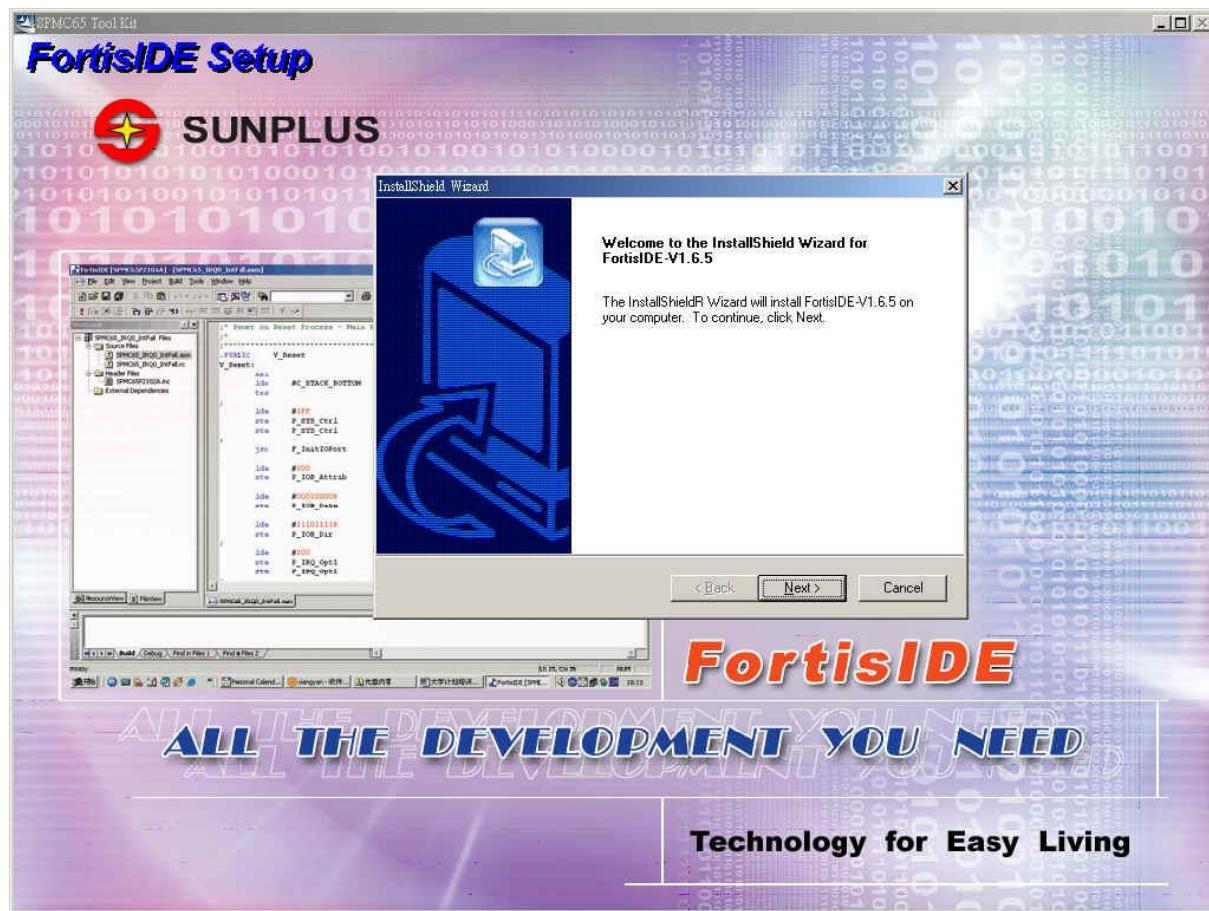
1. System Requirement

FortisIDE™ is capable of running under Windows 98® / 2000®/XP®.

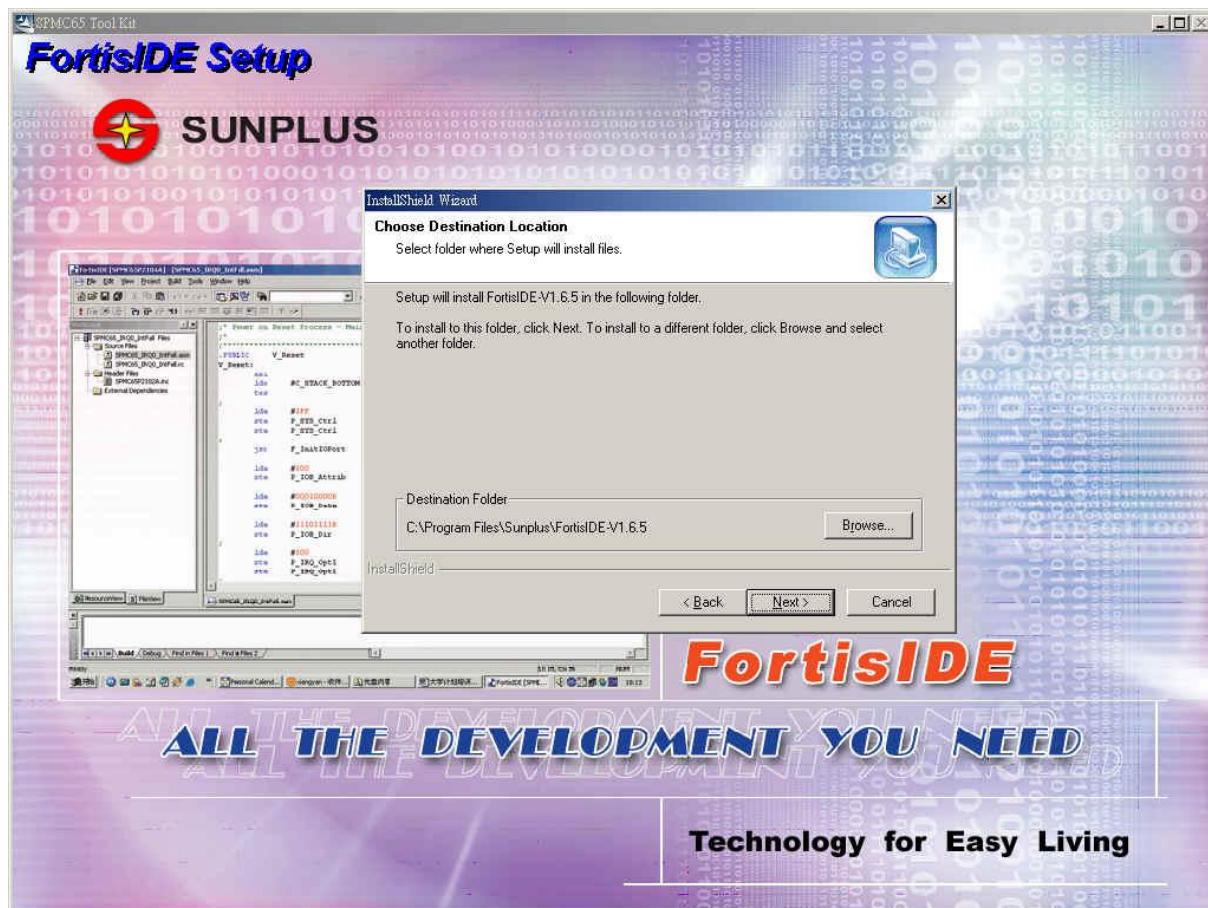
2. Installation

FortisIDE™ installation wizard is accompanied with the development system. In this section, we will guide you how to install FortisIDE™ on your computer.

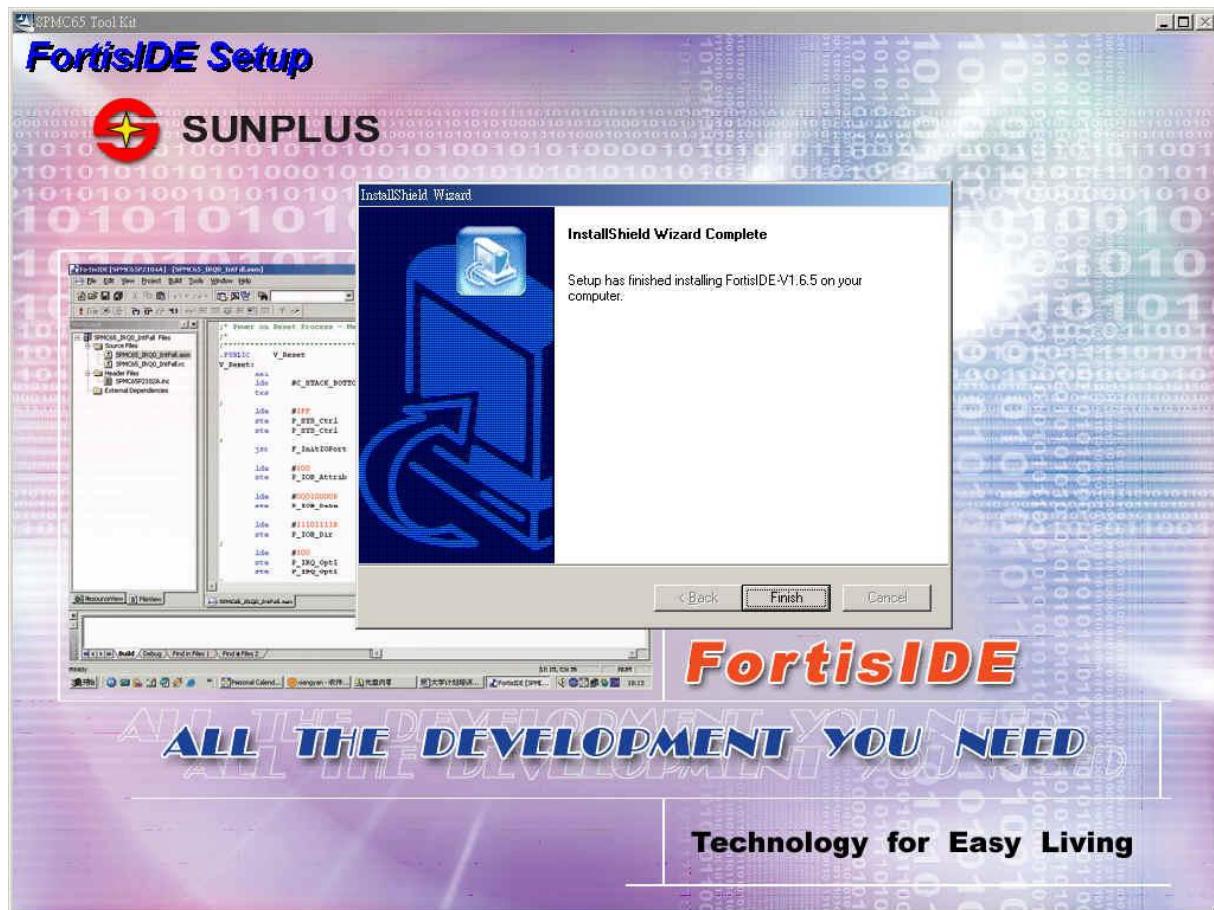
- I. Find FortisIDE software package in CD-ROM or SUNPLUS Website.
- II. Run SPMC65 Tool Kit_v1.0.0.exe and then the welcome screen is shown as follows. V1.0.0 is the release version of software installation package.

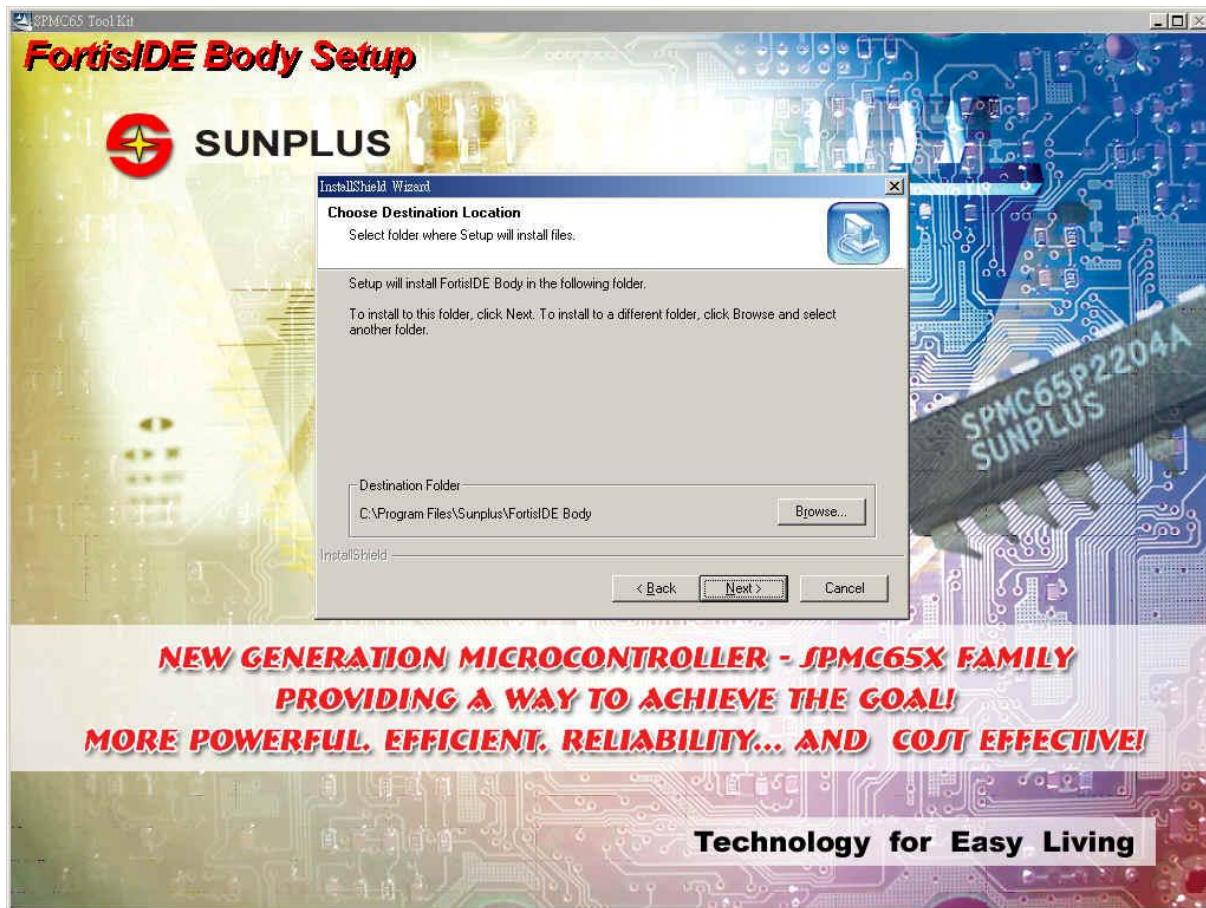


- III. First, follow the installing steps to install “FortisIDE™” software package. Sunplus suggests that user does not change the default installation path.

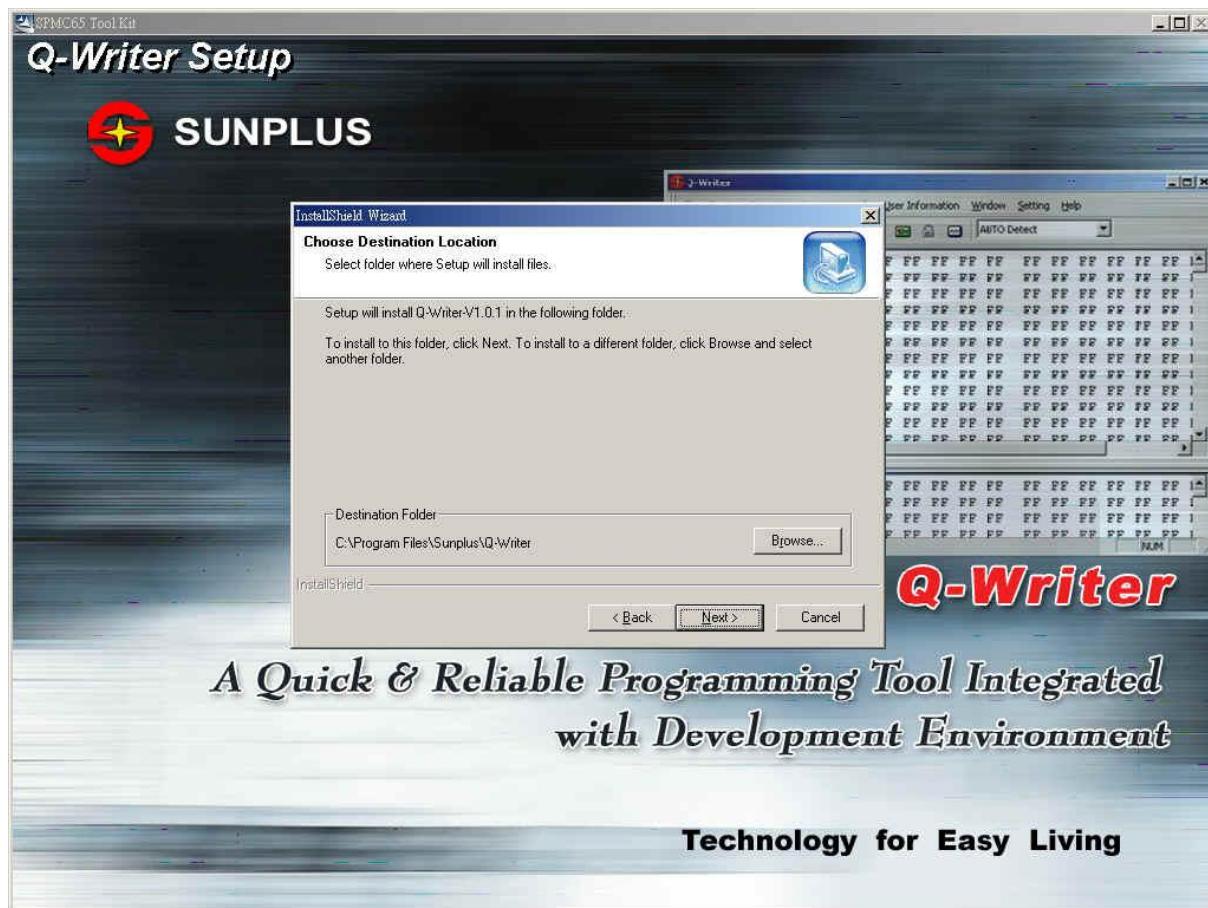


- IV. After installing FortisIDE, the InstallShield Wizard will hold about 5 second and then continue to install FortisIDE Body on your computer.

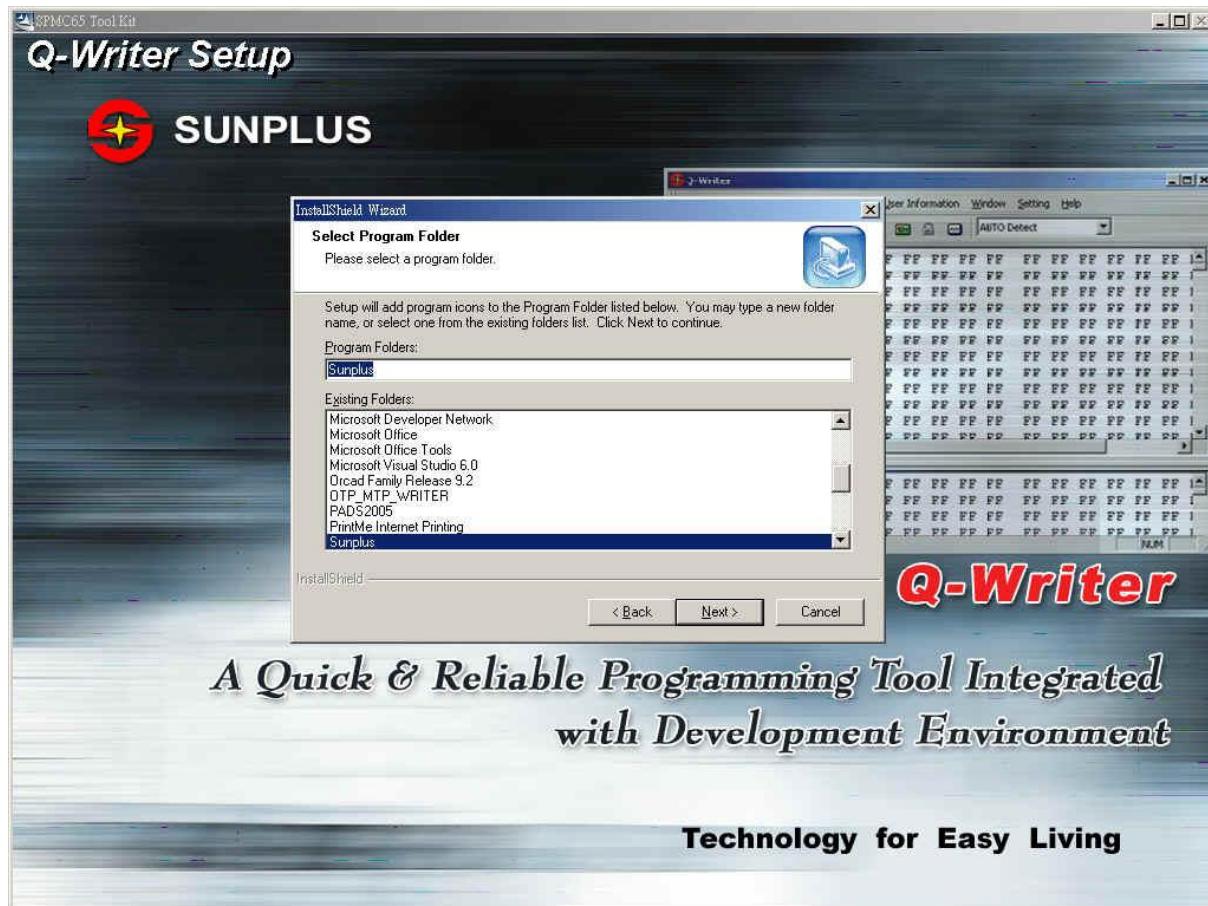




- V. Finally, the installation package will install "Q-Writer" application program on your computer.
After completing the whole installation, the installing picture will disappear.



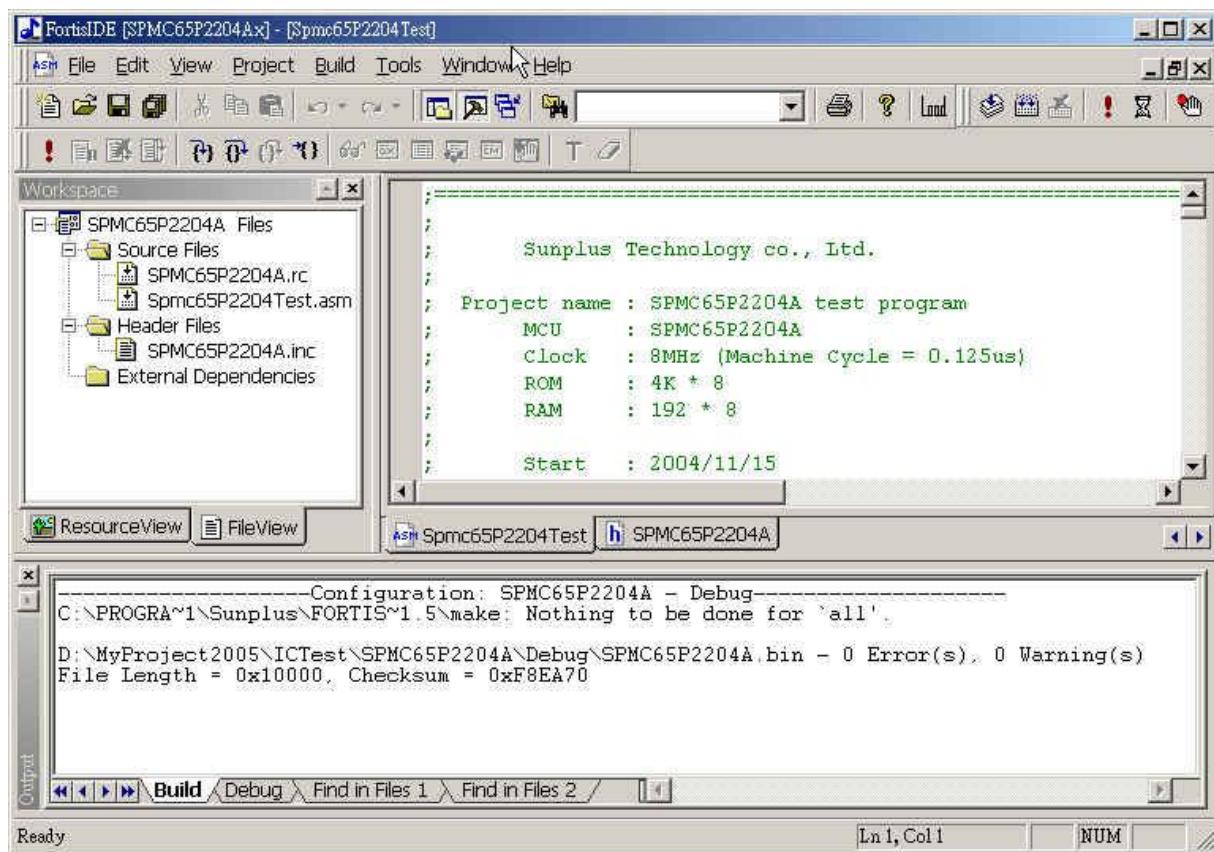
- VI. Enter the installing path in the next step. Sunplus strongly suggests that user does not change the default installation path.



VII. Select [Start Menu] → [Program] → [Sunplus] → [FortisIDE] → [FortisIDE] to run FortisIDE™.

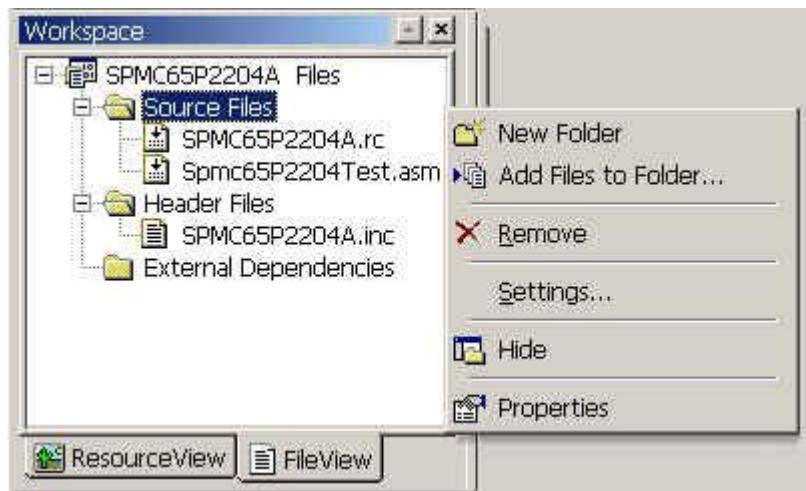
22.2.3 Interface

FortisIDE includes a series of integrated development tools that can be used to edit, compile, link and debug your program within IDE. You are allowed to open more than one window at the same time on the screen. There are three major windows you will see in FortisIDE: Workspace, Output, Editor. You are able to switch among these windows by simply click within window area.



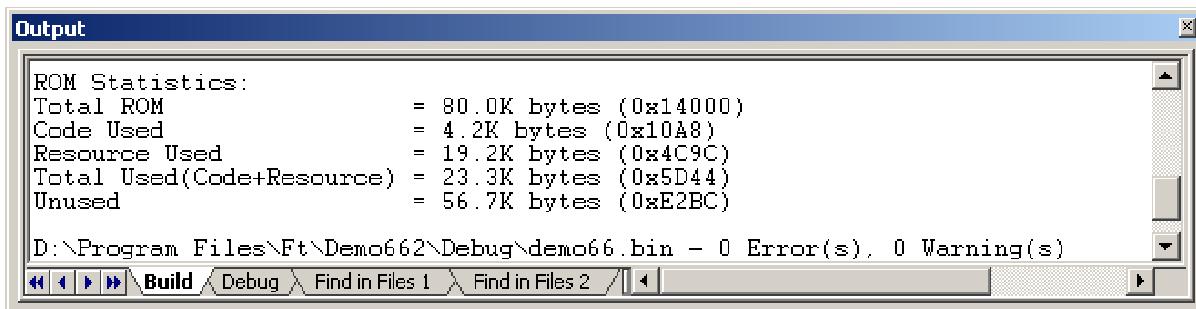
File View

FileView displays all default and user-defined file groups and files in groups. The relationship in this window is the logical relationship, not physical relationship, and therefore, do not reflect the organization of the file on your hard disk. You can right click on top-level project file / file group / files in the group to bring out the separate shortcut menu.



Output Window

Use the 'Output' window to see the status of building, debugging, and finding.



I. Build:

Shows the information of compilation. Error and warning may be given during compilation. A program is built completely and successfully when no error occurred in the report. To find the source code corresponding to an error/warning, double click the error/warning line.

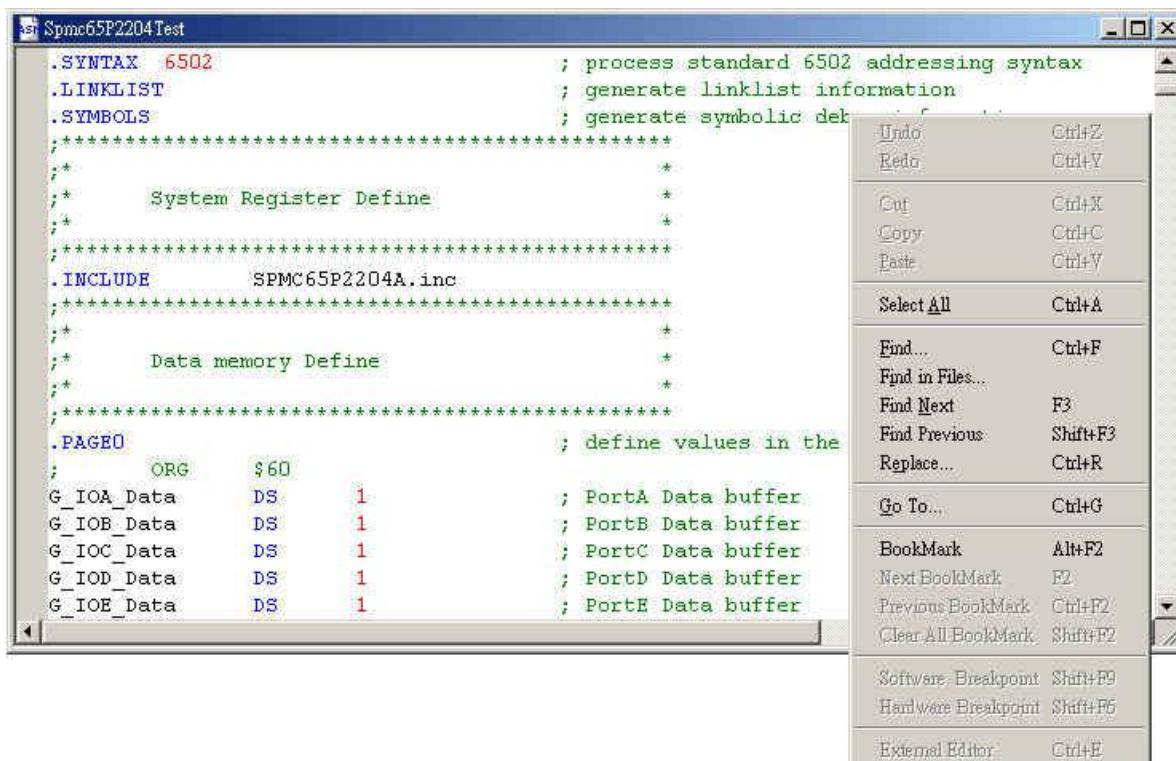
II. Debug:

Shows the information during download.

III. Find in Files:

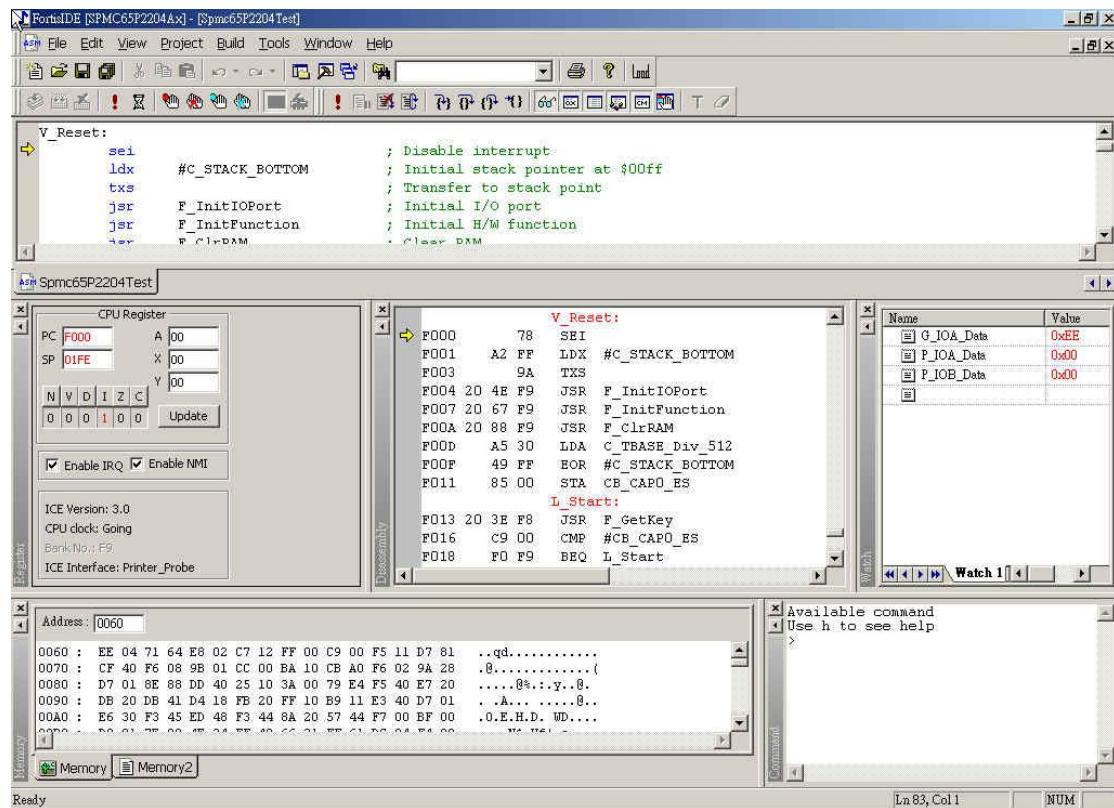
Show the result of finding text in file. By selecting the Output to pane2 in the 'Find in Files' dialog, you can set 'Find in Files2' pane to show the output, else, show the output in 'Find in Files1' pane.

Text Editor



An integrated edit window is provided to edit code. The content in the file can be shown in the window when you open a file. You can start editing the code in the Text Editor. You can also right click to bring out the shortcut menu.

Debug Window



Disassembly window

Display the disassembly contents of memory.

Register window

Display current contents of the general registers and CPU status register.

Memory window

Display the contents of memory.

Watch window

Display the contents of symbols and hex addresses.

Command window

Display the available commands.

Breakpoint window

Display the contents of breakpoints.

Trace buffer window

Display the executed instructions, status and memory contents.

22.2.4 Quick Start

1. Run FortisIDE on Start menu.
2. Click [File] → [Open Project] and select one project. To create new project, click [File] → [New] and follow the on screen direction.
3. The project window is located at the left side. You can open the source file on File view by double clicking on the filename.
4. Click [Build] → [Rebuild All] to compile source program. If there is no error, continue the following step.
5. Click [Build] → [Start Debug] → [Download] to download one program, use the menu command in Debug menu to run and debug it. You can also use [Build] → [Start Debug] → [Go] to run the program directly.

22.2.5 Create Project

A new project contains at least seven types of files: ***.spj**, ***.rc**, ***.set**, ***.env***, **prog.lik**, ***.cmd**, ***.inc**. After creating a new project, FortisIDE automatically adds these files under the project directory. ***.spj**, ***.rc**, ***.set**, ***.env** (with the same project filename) are system files; don't modify the directory. **prog.lik** informs linker how to link object files(user can modify this file directly by external editor). ***.cmd** (with the same project filename) informs loader how to download binary file(.tsk/.bin); you can modify the file to arrange the file image's distribution in ICE/EMU board. If the file is modified, the percentage of downloading bar, which depends on how many ROM sections you choose to download, may not end up with 100%. ***.inc** describes the name definition of each I/O ports and hardware registers; user should include this file in the source code.

Step:

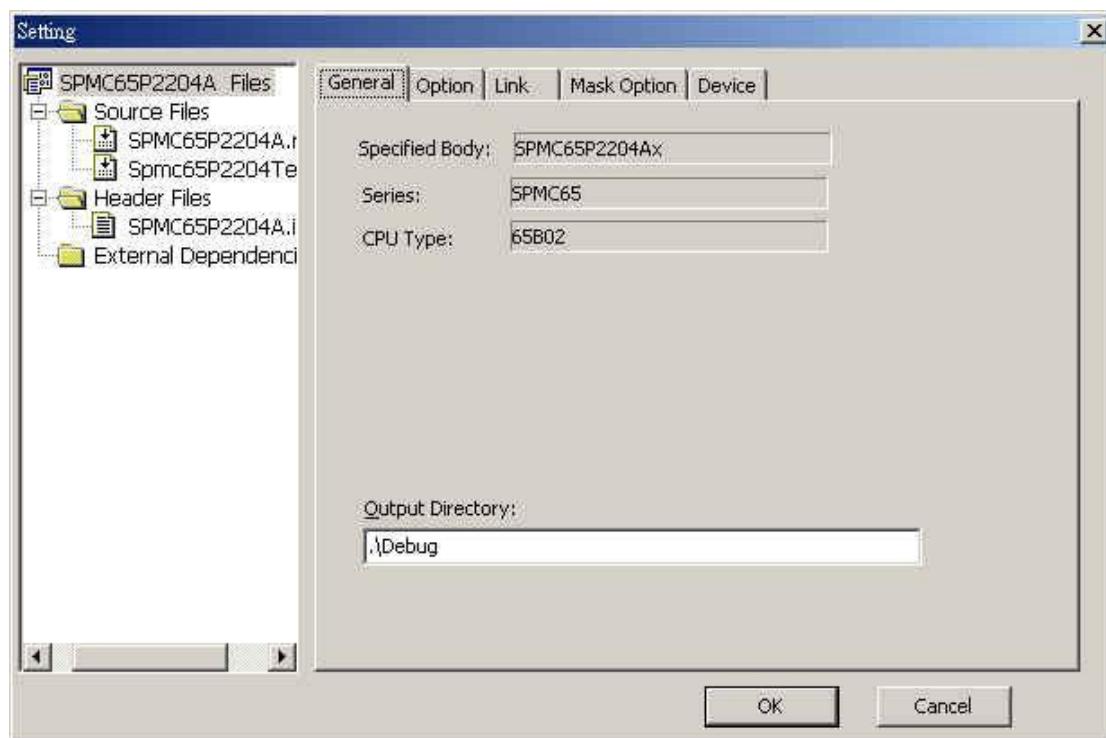
1. Click [File] → [New] → [Project] to open the create project window.
2. Input project name in File field.
3. Select/write down the project path in Location field.
4. Click [Next] to select body type.
5. Click [Finish].

22.2.6 Project Setting

Click [Project] → [Setting...] to pop up **Setting** dialog box.

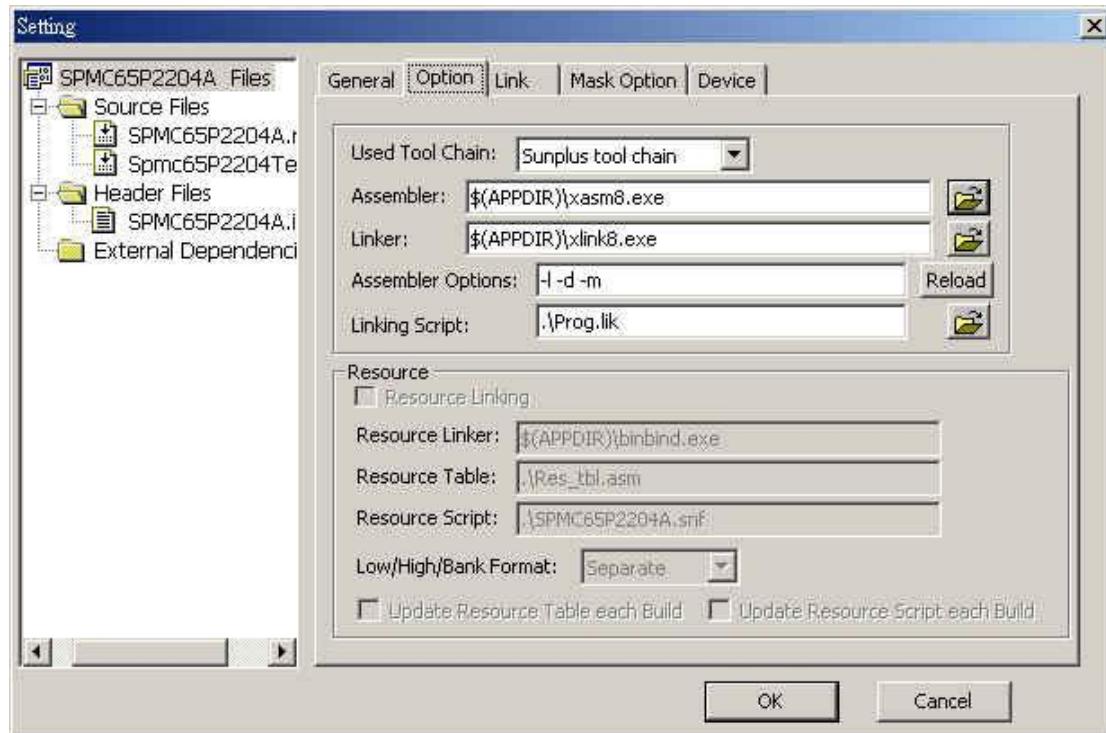
1. General page

In this page, you can find the body adopted by the project, and the corresponding series, CPU type, and can set output directory for the target file and intermediate files created during compilation.



2. Option page

In 'Option' page you can configure the tool chain and the resource. On the upper you can select Sunplus tool chain or customized tool chain.



I. Assembler and Linker

Assembler option and linking script can be specified separately in the 'Assembler Options' and 'Linking Script' field.

Four options in assembler are provided (-l -d -m -z) and user can mix them arbitrarily.

-l: generate listing file.

-d: generate source level debug information.

-m: generate macro into debug information.

-z: fill DS content by zero.

The 'Reload' button is provided to reset the default assembler option. Linking script is provided to arrange section's distribution in the file image by modifying the prog.lik directly.

II. Resource

Resource is available or unavailable, according to the type of body. If resource is available, it can be set to be enabled or disabled further by checking/un-checking the 'Resource Linking' checkbox.

Resource linking:

If resource is enabled, resource linker will be invoked after the linker executed during compilation.

Resource Table (Res_tbl.asm):

That is provided to reserve 3 byte's space (address low/high/ bank) for each resource file in the separate resource group. User can include it in the source code. If user doesn't want to let IDE update the content of resource table automatically each building project, don't put a check on the 'Update Resource Table each Build' of checkbox.

If user want to update the content of the resource table anytime he need, switch to the 'ResourceView' in the 'Workspace' window, then right click on the top-level resource to pop up one sub-menu, and select the 'Update Resource Table (Res_tbl.asm)' item.

Two formats of the 'Address low/high/bank' are supported. One is continue, another is separate.

The continue means address low/high/bank is located continuously for each resource file; the separate means address low/high/bank is located separately for each resource file. You can change the default format (continue) by re-selecting one in the 'Low/High/Bank Format' combo box.

Resource Script (.snf):

This is the input file of sunplus resource linker (binbind.exe). Whenever building project, IDE will automatically feed it to the resource linker if the resource is enabled. If user doesn't want to let IDE update the content of resource script automatically each building project, don't put a check on the 'Update Resource Script each Build' of checkbox.

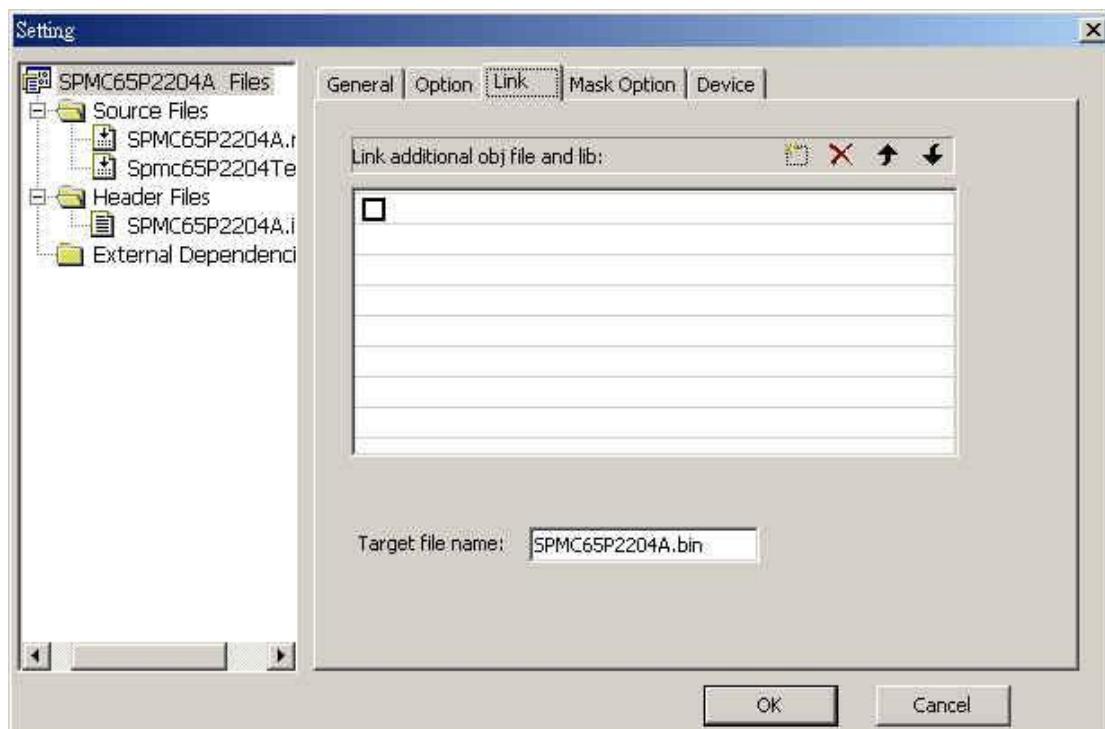
If user want to update the content of the resource script anytime he need, switch to the 'ResourceView' in the 'Workspace' window, then right click on the top-level resource to pop up one sub-menu, and select the ' Update Resource Script (*.snf)' item.

For some extra application, e.g., the body owns expanded ROM, you should use enhanced function in the resource linker. To achieve this aim, modify the .snf file user-self and un-check the 'Update Resource

Script each Build' checkbox. Please reference sunplus binbind user guide to obtain the detail spec. of resource script.

3. Link page

In **Link** page, you can select the additional object file (*.obj) and library file (*.lib) to link with your project. To edit the 'Target file name' field can specify the target filename.



Address space of code crossing bank: (ignore this paragraph for the code without crossing bank) For code crossing bank, there are two address space of file image, which is from the linker's output with the extension of .s37, to arrange/locate these code. One is sequence address, another is non-sequence address.

If .s37 image is sequence address, 'Sequence address' item must be selected in the 'Address space of code crossing bank' combo box to inform the resource linker to load .s37 file in sequence mode.

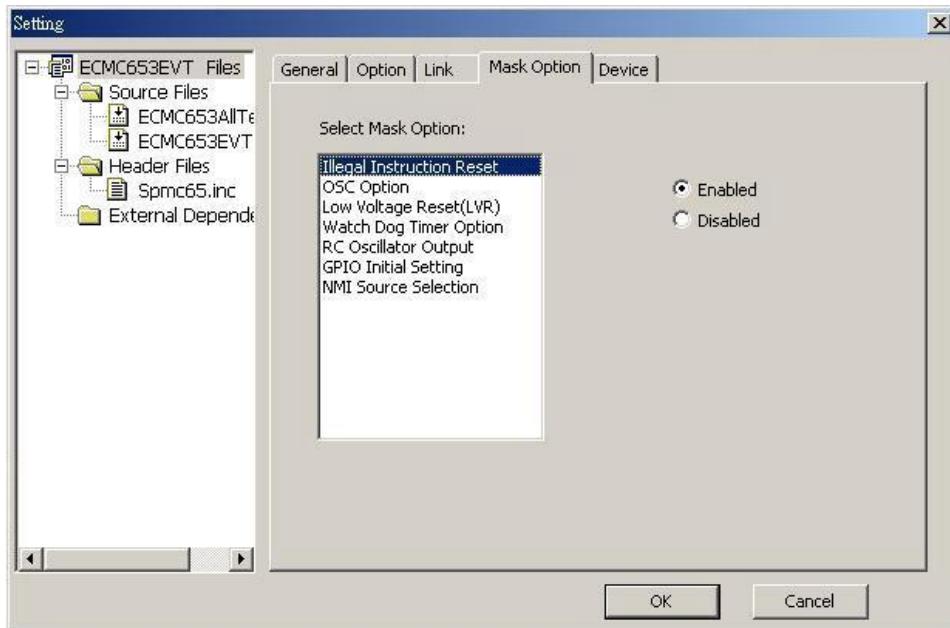
If .s37 image is non-sequence address, 'Non-Sequence address' item must be selected in the combo box to inform the resource linker to load .s37 in non-sequence mode.

For non-sequence address of file image, please reference the online help of resource linker by executing binbind.exe / N [BodyName] in installed directory.

4. Mask Option page

FortisIDE can fill the specific value to fixed locations of a target file after project is built. The current device configuration registers setting of SPMC65X family microcontrollers are OSC Option, Low Voltage Reset (LVR), Watch Dog Timer option, RC Oscillator Output, GPIO Initial Setting, and NMI Source Setting. User can

set the suitable option data in accordance with the actual application through FortisIDE.



5. Device page

Select one ICE interface which connected with ICE. Move mouse to an item and you may find the prompt of supported ICE on the right side. If 'Auto detect' item is selected, IDE will auto detect ICE interface & ICE currently used.



ICE Configuration

Including ICE version, clock source, PC trace, bus tri-state, memory attribute, program code's type, etc..

For SPMC65x family, only some selection items are available as the following.

I. Used ICE Interface

There are two kinds type of probes for connecting computer and emulation board, which are USB probe and printer probe. The default selection is “Auto detect” for using any probe, Or user can assign the probe type according to used probe.

II. PC Trace Enable

(a) What's PC Trace Buffer

When something goes wrong in an embedded system, user often wants to know what happened just before the failure. A trace buffer is a section of memory that provides a way to do that. A trace buffer is typically of binary length with each location holding 1 entry of information. Each entry is an action code. Each action code originates at a key location in your program, where you know from the program sequence that the corresponding specific action has occurred.

(b) The Feature of Sunplus PC Trace

The PC Trace is a high-speed RAM used to capture the bus and IO action of microcontroller. Sunplus provides a powerful PC Trace for debugging your program.

– Dynamically record the reading/writing address and data of CPU, and transmit data through IDE for debugging program and tracking problem conveniently.

– Built-in 1K SRAM PC Trace buffer in ECMC653A and integrated with SPMC65x family EMU board.

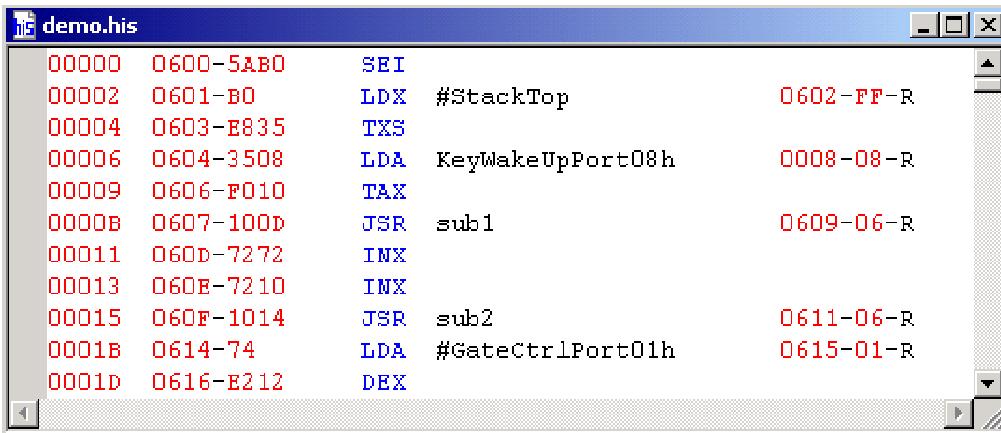
User need not connect any external hardware module.

–PC Trace has 1K recordable depth entry.

(c) How to use this function

Enable PC Trace function if it is checked. There are two kinds of setting mode for operating the PC Trace function. User can set two hardware/software breakpoints between some locations where user wants to know if there are some problems in the code. After free running to the second breakpoint, user can click

 from the toolbar in debug mode to enable PC Trace function. And then see the following picture. The window shows the executed status and data for debugging.



```

00000 0600-5AB0    SEI
00002 0601-B0      LDX #StackTop          0602-FF-R
00004 0603-E835    TXS
00006 0604-3508    LDA KeyWakeUpPort08h   0008-08-R
00009 0606-F010    TAX
0000B 0607-1000    JSR sub1              0609-06-R
00011 060D-7272    INX
00013 060E-7210    INX
00015 060F-1014    JSR sub2              0611-06-R
0001B 0614-74      LDA #GateCtrlPort01h   0615-01-R
0001D 0616-E212    DEX
    
```

1. Overwrite mode: This is supported by ICE 3.0 and ICE 2.0. Its capacity is 1k cycles recorded. This mode is used to record data continuously. When the buffer is full, the system will overwrite from the beginning.
2. Extended mode: It is only supported in ICE 3.0. This mode is also used to record data continuously. But the difference between overwrite and extended mode is that, while enabling extended mode, the debugging system will stop if the trace buffer is full. The system will forward the buffer data to PC automatically and then record next 1K entries continuously. In this mode, there is an illimitable PC Trace buffer for debugging. It may affect the normal interrupt frequency of the timer due to the hardware mechanism, so key detection or voice play may be lost or broken in some applications. Sunplus recommends user not to use this function in the real-time application.

III. The Storage Type of Code

I Use RAM:

Software & hardware breakpoint are available in ICE 3.0, and only hardware breakpoint is available in ICE 2.0. Download always is done.

I Use ROM:

Only hardware breakpoint is available separately in ICE 3.0 & ICE 2.0, and download bypassed to support ROM debugging in downloading.

22.2.7 Operation of Project File

Add File to Project

- I. Click [Project] → [Add File to Project] → [File] to display 'Add File' dialog.
- II. Select filenames in 'File Name' list box and click [Add] to add file.

After you close the 'Add Files' dialog, FortisIDE™ can add selected file to the corresponding group determined by filename extension.

Alternative Step:

Move mouse cursor directly pointing on the file group to be added and press right key and select [Add File to Folder]. Adding file to project window means only adds the file to project file list, but not add/copy the file

from the original location to current project directory.

Delete File

Move cursor on the file to be deleted and press [DEL] to delete file. Or, you can click right key to pop up a sub-menu and select [Remove] to delete file.

Move File

Move cursor to one file to be moved and drag it to the target file group.

22.2.8 Build Project

1. Compile & Build Project

Fortis IDE is able to compile and recompile the entire project/single program in one project. Fortis IDE can only compile the files that are changed in the last compilation while building project. Rebuild function compiles all files in the current project.

- I. Click [Build] → [Compile] to compile the active file owning the focus on the editor window. (Equal to right click on *.asm, etc. in the 'Workspace' window and select [Compile].)
- II. Click [Build] → [Build] to build the project, this function only is used to compile the files that are changed after the last compilation.
- III. Click [Build] → [Rebuild] to recompile all files in the current project.
- IV. Click [Build] → During compilation, [Stop Build] is used to stop compilation.

Build information is shown in Build tab of Output window. Compilation is successful if no error existed.

Note!

User must reserve an empty line in the end of project code, otherwise the compiler will show an error message as the following picture. To solve the 'Newline' error, user just presses the enter key after '.END' directive of project code.

```
-----Configuration: TestIstruct - Debug-----  
C:\PROGRAM~1\Sunplus\FORTIS~1.5\xasm8.exe -Tb -s -l -d -m -z -o ".\Debug\TestInst.obj" "E:\SPMC653 Series\Sunplus 6502 Assembler - Version 1.6.8 , Copyright(c) by Sunplus.  
Apply for 65b02(with bit operation) instruction set  
Error: Spmc65.inc: no newline at end of file. Please add <Enter> to the end of file.  
1 error(s), 0 warning(s).
```

2. Run Program

After you compile the project and no error is reported, the program can be run under IDE if ICE board is ready.

To run program

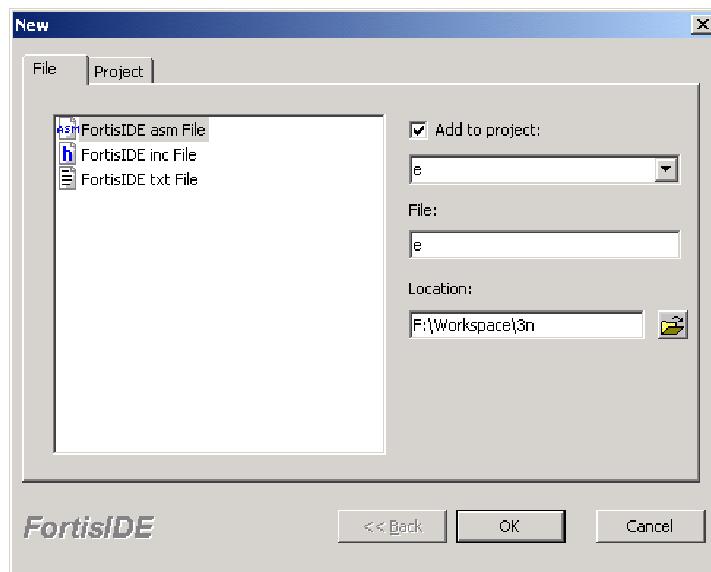
- I. Click [Build] → [Start Debug] → [Download] to download the executable program
- II. Click [Build] → [Start Debug] → [Go] to run program.

You also can download pure executable file, [File] → [Load Program] can be used efficiently.

22.2.9 Text Editor

1. Create Document

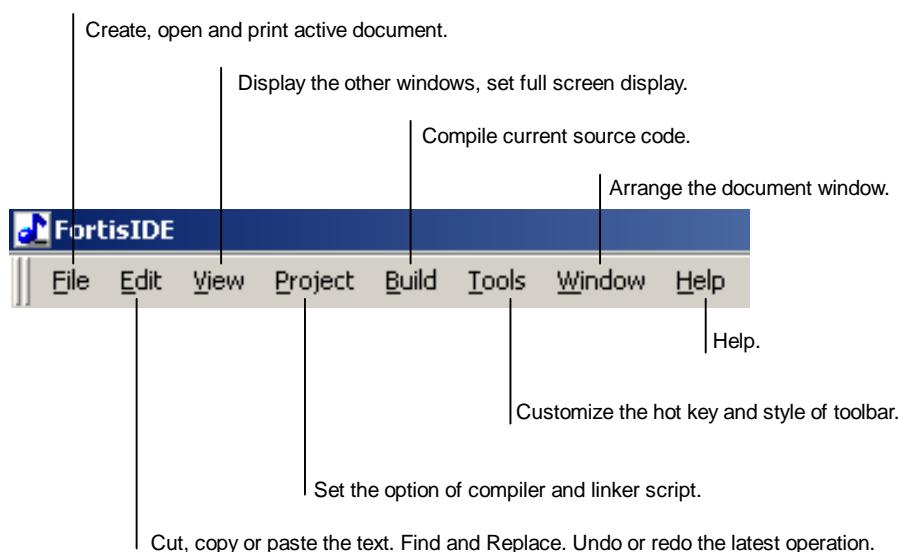
The asm (assembly file), inc (include file) and txt (text file) are supported in FortisIDE™. Click [File] → [New] to open 'New' dialog.



Select file type in the left-side list box. And, check the checkbox 'Add to project' to add the file to a project. Meanwhile, enter a new filename and directory separately in File box and Location fields. Finally, click [OK] to complete the operation.

Menu is universal for all FortisIDE™ Editor Windows. FortisIDE™ starts up appropriate menu for current editor.

The following indicates the valid menus in the active text editor window.



2. Open Document

FortisIDE™ offers the MDI editor window that is able to open multiple documents at the same time. If document

window is maximized, the title will be displayed on the title bar of the top of the screen. Press Ctrl + F6/click the document name in the 'Window' menu to switch between the documents.

Click [File] → [Open] to show 'Open' dialog.

Another way to open:

Click [File] → [Recent Files] to select one document and open it directly. Recent Files menu lists 8 most recently accessed files.

3. Document Layout

User can view documents in full screen mode by clicking [View] → [Full Screen]. In full screen mode, title bar and status bar are invisible. Press [Esc]/click the button on floating toolbar to return to the normal view. Window menu provides a list of all opening documents. Click on the filename in this menu to activate the document.

Click [Window] → [Cascade]/[Tile] to arrange the layout of documents.

Click [Window] → [New Window] to create another window for the active document (each window has its own scroll bar and cursor).

Next, click [Window] → [Tile] to arrange windows. All windows for the active document can be changed synchronously.

4. Save Document

The document name on the title bar or on the 'Window' menu marked with '*' means that the content has been un-consistent between the document and its corresponding file. Save the document, and '*' is disappeared. If close the document with '*', editor can prompt you to save it.

The following is the ways to save the document:

Click [File] → [Save] to save file with the same filename.

Click [File] → [Save As] to save file with different filename.

Click [File] → [Save All] to save all open documents.

5. Bookmark

Bookmark is provided for user to quickly locate some special lines in the text editor. Bookmark can be easily set and deleted. To set a bookmark, follow the steps: Move cursor to target line.

Press Alt + F2/click [Edit] → [Bookmark].

The line marked by bookmark will have one green mark on the left margin. Press F2/click [Edit] → [Next Bookmark] to move cursor to the next bookmark line. Three ways to delete bookmark:

Move cursor to target line, and press Alt + F2 to close the bookmark.

Press Shift + F2 to close all bookmarks.

Closing the document also closes the relevant bookmarks as well.

6. Find Text

Text editor provides three ways to find text:

1. Look up text in an opening document.
2. Replace text with other text in an opening document
3. Search for text in a disk file.



SPMC65X Family PROGRAMMING GUIDE

I. Find in an Opening Document

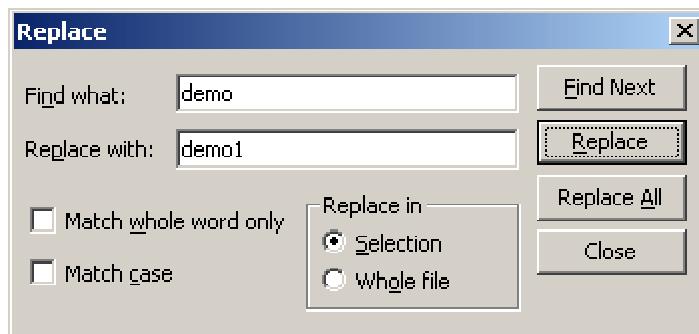
1. Click [Edit] → [Find] to open the 'Find' dialog.



2. Input the string to be found in 'Find what' area.
3. Select 'Match Case'/'Match Whole Word Only' to set the matched condition further.
4. Select 'Up'/'Down' to search for the string from the current location of the cursor to the beginning of the document or to the end of the document.
5. Click [Mark All] to mark all matched strings with bookmarks, and click [Unmark All] to unmark all matched strings with bookmarks.
6. Click [Find Next] to begin to find matched string step by step.

II. Find/Replace in an Opening Document

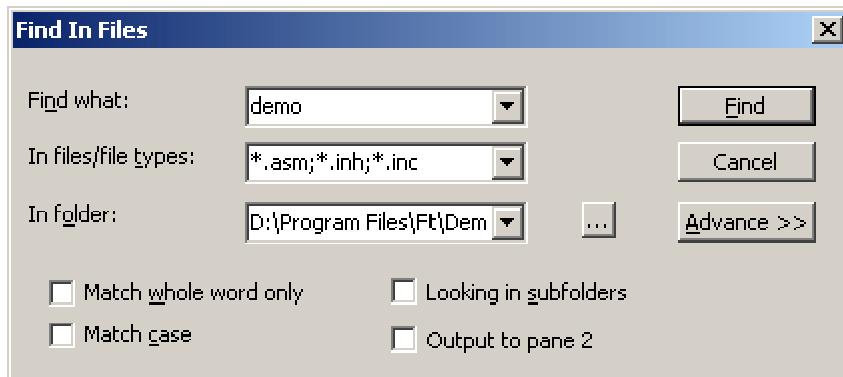
1. Click [Edit] → [Replace] to open 'Replace' dialog.



2. Input string 1 in 'Find what' area.
3. Input string 2 in 'Replace with' area.
4. Select 'Match Case'/'Match Whole Word Only' to set the matched condition further.
5. Click radio box to select range of replace/find if document manipulated is marked in advanced.
6. Click [Find Next] to begin to find matched string.
7. Click [Replace]/[Replace All] to begin to replace step by step /replace all matched strings in current document.

III. Search for Text in Disk File

1. Click [Edit] → [Find In File] to pop up a dialog. Here user can specify a string, file type, and find path.



2. Input string in 'Find what' area.
3. Default folder is the current project folder. User also can input another directory to 'In folder' by clicking button '...' to specify the directory.
4. Select file type in 'In files/file types'.
5. Select 'Match Case'/'Match Whole Word Only' to set the matched condition further.
6. Select 'Looking in subfolders' to search in subfolder (Default option).
7. Select 'Output to pane 2' to output the result; user can view the result by clicking Tabulating Button, 'File In file2', at the bottom of the 'Output' window. If 'Output to pane 2' is un-checked, the result will output to 'Find in File 1' of 'Output' window.
8. Click [Advance] to set additional directories for finding.
9. Press [Find] to begin to find.

22.2.10 How To Use Debugger

The integrated debugger can help you to locate program logic error after the syntax error are cleared. All contents in variables, memory, and registers can be shown on your screen in debug mode.

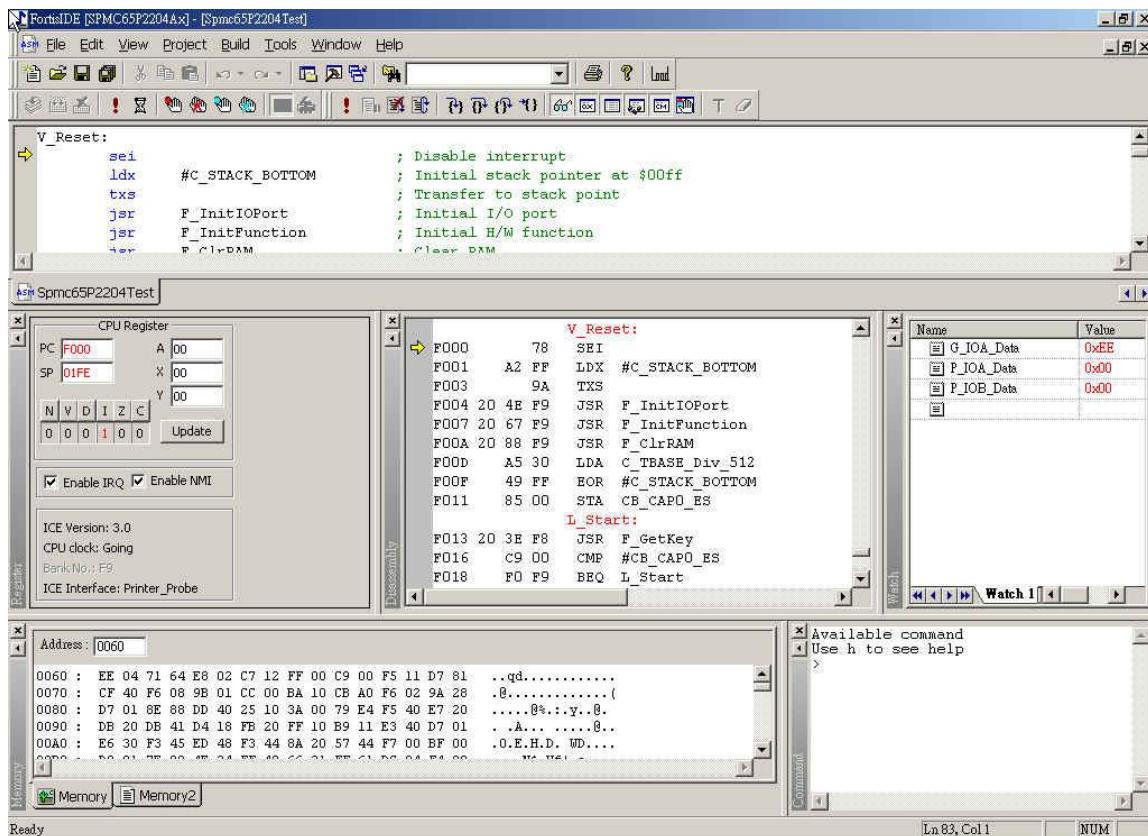
1. Run Control

First, click [Build] → [Start Debug] → [Download] to download program and start debugging program in debug mode. Once the program is debugging, you can use menu commands on Debug menu to control program flow. The following table lists all commands and descriptions.

Go (F5)	Run program from current position until a break point is caught.
Restart	CPU and program counter is reset.
Stop Debug	Stop debugging the program, and return to normal editing state.
Break	Break running.
Step Into	Run one instruction (step by step running the program).
Step Over	Run one instruction or one function/procedure.
Step Out	Skip out function/procedure and return to calling function.
Run to Cursor	Run from current line to the cursor line.

2. Debug Windows

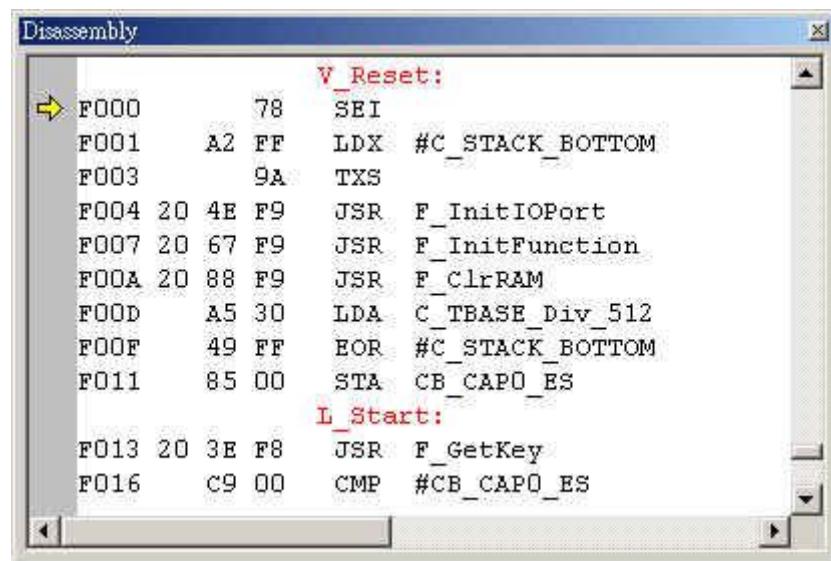
In debug mode, debug windows can be toggled on the Debug toolbar or on [View] → [Debug Windows]. These debug windows are used to display/modify the status of the CPU registers, memory, IO etc.



Note: To view different information at the same time, you can adjust each window size by dragging the rectangle's edge, or its position by dragging on the title name.

I. Disassembly Window

Disassembly: [View] → [Debug Windows] → [Disassembly]



```

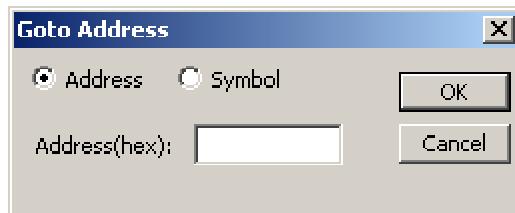
Disassembly
V_Reset:
F000      78      SEI
F001      A2 FF    LDX #C_STACK_BOTTOM
F003      9A      TXS
F004 20 4E F9  JSR F_InitIOPort
F007 20 67 F9  JSR F_InitFunction
F00A 20 88 F9  JSR F_ClrRAM
F00D      A5 30    LDA C_TBASE_Div_512
F00F      49 FF    EOR #C_STACK_BOTTOM
F011      85 00    STA CB_CAP0_ES
L_Start:
F013 20 3E F8  JSR F_GetKey
F016      C9 00    CMP #CB_CAP0_ES
    
```

The 'Disassembly' window can be used for debugging code. The contents of disassembly will be changed with the PC address. Right click the window, you can find a hot key menu.



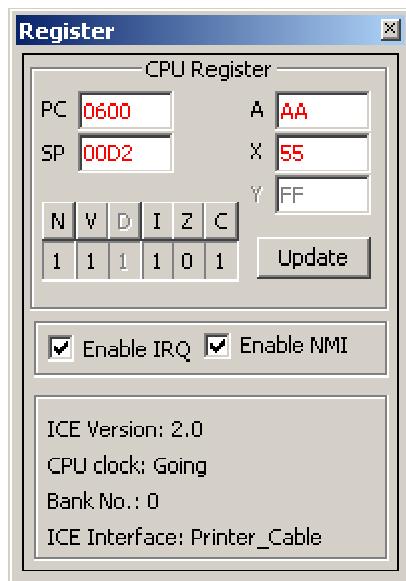
You can go to a specified address or symbol by selecting [Goto] function. The 'Goto Address' dialog is popped-up, then, enter the address or symbol and select [OK], the disassembly line of that address will be displayed on the top of the window.

To turn the view back to the address the program counter pointed to, you can select 'Return to PC'. To show the machine code, click 'Show Code Bytes'.



II. Register Window

Register: [View] → [Debug Windows] → [Register]



The 'Register' window is used to display the information of general registers, CPU status register, ICE version, ICE interface, CPU clock status, bank number (the display of bank no. in running is only supported in ICE3.0). Besides, it can allow the user to modify or edit some members mentioned above, and this modification can be delivered to ICE / EV board immediately.

PC: Program Counter

SP: Stack Pointer

A, X, Y: General register

N, V, D, I, Z, C: Negative, Overflow, Decimal, Interrupt, Zero, Carry

Enable IRQ/NMI: Enable or disable IRQ/NMI

ICE Version: Current used ICE

CPU clock: CPU clock's status (sleeping or going)

Bank No.: Number in bank port

ICE Interface: Currently used interface connected to ICE

[Update]: Update the modified configuration to ICE / EV board

III. Memory Window

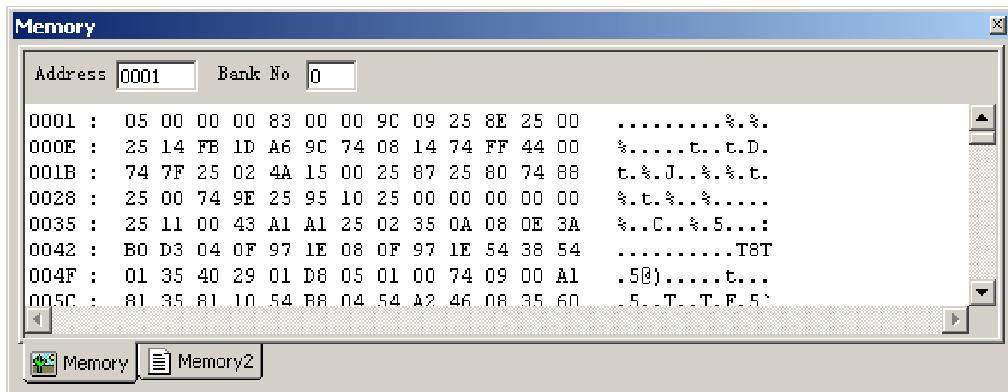
Memory: [View] → [Debug Windows] → [Memory]

'Memory' window displays the memory content in **Memory** and **Memory2**

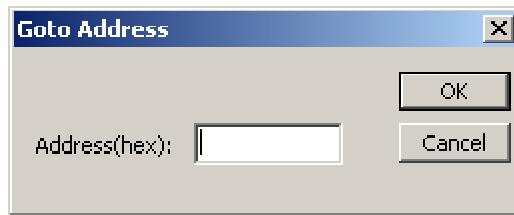
Memory

Memory window displays numbers in hexadecimal format. Assign the beginning address in the 'Address' field and 'Bank No' field, and you can view the memory data from that position. You may also view another block of data by clicking on the scroll bar. Furthermore, you can modify the content in the specified address by simply pointing to the location you want, and type one value of

byte unit into it.



Right click the window and you can find the [Goto] function and [Dump] function. To go to a certain address, please select [Goto]. After typing the address in the dialog, the 'Memory' window will display the memory data from that address.



Memory dump function is used to dump a block of memory content to one file. Input the address range separately in the 'Start Addr' and 'End Addr' area, enter the file name and click [OK], then, the data will be dumped to a file.



Memory2

Memory																			
	Label	Addr	No	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	R_IntTemps	0081	0A	FF	FF	FF	40	00	00	02	FF	FF	FF						

Memory2

Memory2 provides one spreadsheet field that displays the contents of one consecutive memory location beginning from some address specified by the input label in the 'Label' field or hex address in the 'Address' field. The 'No' field with maximum value 10(hexadecimal) specifies the number of the bytes to be shown.

IV. Watch Window

Watch: [View] → [Debug Windows] → [Watch]

Watch		
Name	Value	Address
R_IntFlags	0xC0	0x0080
R_IntTemps	0xD0	0x0081
0x82	0x40	0x0082
0x600	0xB0	0x0600

Watch 1 Watch 2 Watch 3 Watch 4

'Watch' window is used to watch the value of the symbol or hex address represented by prefix \$ when debugging program. You can also modify their values directly in this window by typing the value you desired into 'Value' column. Each page contains one spreadsheet field that can display label information. You can input the symbol name or address into the 'Name' column by directly typing or dragging the text from editor window. Then, debugger will auto fill the 'Address' column by the evaluating the address of text inputted, and fill the 'Value' column by reading the memory of that address.

If the value of a symbol or address appears in red, it indicates that the value has been just changed. By right clicking on this window, a context menu featuring **Paste**, **Show decimal/hexadecimal**, **Hide**, **Remove** and **Remove all** functions is enable.



Paste: You can cut or copy a symbol from editor window and paste it into the 'Watch' window.

Show Decimal: Display value in decimal/hexadecimal.

Hide: Hide 'Watch' window.

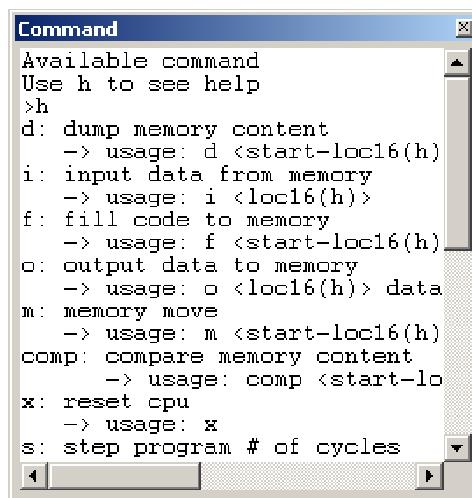
Remove: Remove the rows with selected focus from 'Watch' window.

Remove All: Remove all rows from 'Watch' window.

To resize a column to fit its content, drag the vertical divider at the column edge.

V. Command Window

Command: [View] → [Debug Windows] → [Command]



The online provide detail description and usage for each command by typing H in prompt line '>'.

Command:

d: dump memory content

i: input data from memory

f: fill data to memory

o: output data to memory

m: memory move

comp: compare memory content

x: reset cpu

s: step program no. of cycles

g: execution program

ft: program forward trace (only available in ice2.0)

clk: set cpu clock source

ms: set memory block attribute

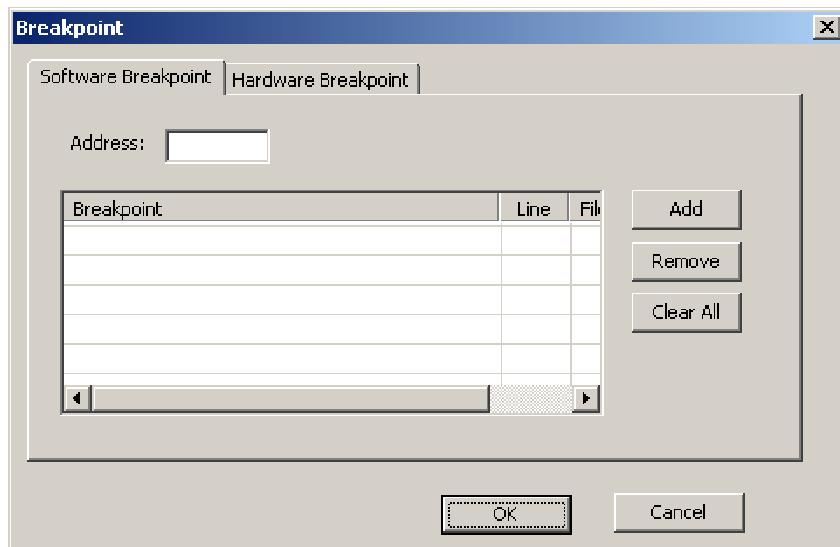
cls: clear listed commands

VI. Breakpoint Window

'Breakpoint' window include software breakpoint window and hardware breakpoint window. Here, you can set, remove, enable/disable and view breakpoints. The breakpoints you set will be saved as a part of your project configuration. After the hardware / software breakpoint is set, a blue / red point is marked in front of the line.

Software Breakpoint Window

Click [Edit] → [Software Breakpoint].



Description:

1. Software breakpoint only supports instruction break, that is, breakpoint only set on the instruction line.
2. Number of software breakpoint is infinite.
3. Software breakpoint is available only in ICE3.0 and uses RAM as the storage of the program code (reference about the setting of '[The Storage Type of Code](#)').

Usage:

Addr: The address to be caught.

[Add]: Add one software breakpoint.

[Remove]: Remove a breakpoint.

[Clear All]: Remove all breakpoints.

Alternative ways to set software breakpoint

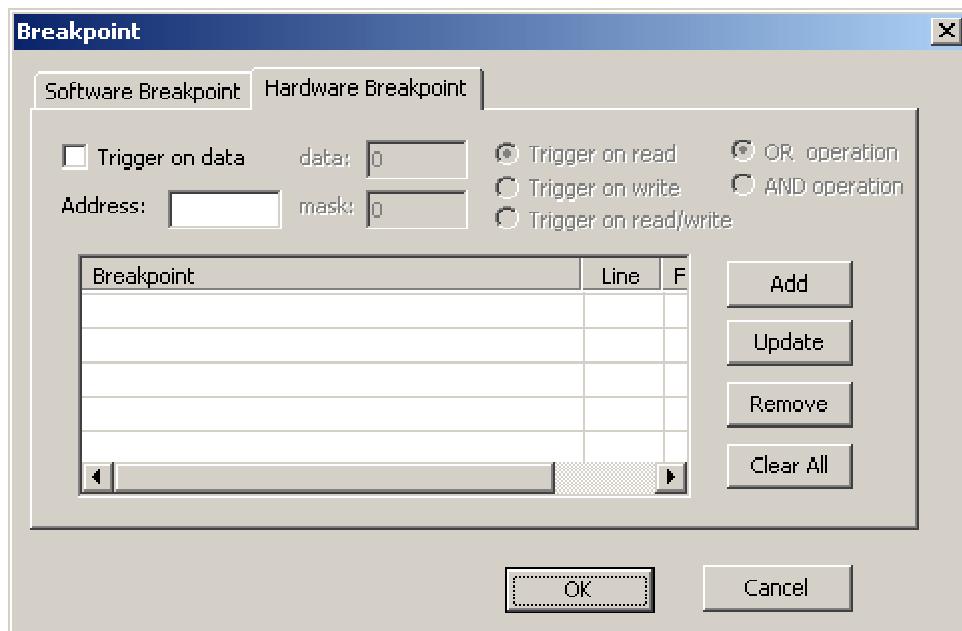
1. Press [F9] on an instruction line in the source code window or in the 'Disassembly' window.

2. Click icon  on an instruction line in the source code window or in the 'Disassembly'

window.

Hardware Breakpoint Window

Click [Edit] → [Hardware Breakpoint].



Description:

1. Hardware breakpoint supports both instruction break and condition break.
2. Two hardware breakpoints are provided.
3. Hardware breakpoint is available in both ICE2.0 and ICE3.0.

Usage:

Trigger on data: Check for condition break, else, for instruction break.

Address: The address to be caught.

data: Value to be caught.

mask: Mask the value set on 'data' field.

Trigger on read/Trigger on write/Trigger on read-write: Select one execution's behavior to be caught. OR operation/AND operation: Select one logical operation applied to two condition breakpoints.

[Add]: Add one hardware breakpoint.

[Update]: Update the setting for the line with the focus.

[Remove]: Remove a breakpoint.

[Clear All]: Remove all breakpoints.

Alternative ways to set hardware breakpoint:

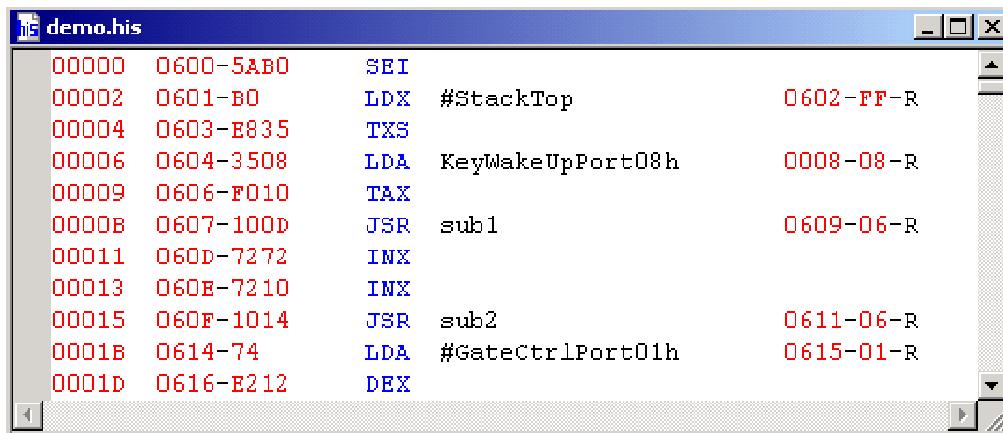
1. Press [F6] on an instruction line in the source code window or in the 'Disassembly' window.

2. Click icon  on an instruction line in the source code window or in the 'Disassembly'

window.

VII. Trace Buffer Window

The executed instructions, status and memory contents will be saved in trace buffer when program is running. Be sure to check 'PC Trace Enable' before you use trace buffer (about the setting of PC Trace Enable, reference [Device page](#)).



The screenshot shows a software interface titled 'demo.his'. The main window displays a list of memory addresses, their corresponding values, and assembly instructions. The data is organized into three columns:

Address	Value	Instruction
00000	0600-5AB0	SEI
00002	0601-B0	LDX #StackTop
00004	0603-E835	TXS
00006	0604-3508	LDA KeyWakeUpPort08h
00009	0606-F010	TAX
0000B	0607-100D	JSR sub1
00011	060D-7272	INX
00013	060E-7210	INX
00015	060F-1014	JSR sub2
0001B	0614-74	LDA #GateCtrlPort01h
0001D	0616-E212	DEX

Click  from the debug toolbar to read the trace buffer, and its content is displayed in the 'Trace Buffer' window in the format as following:

Click  from the debug toolbar to clear the content of trace buffer, and meantime, trace buffer window is closed.

Note: 'Clear Trace Buffer' function only can be supported in ICE3.0.

22.2.11 Toolbar

-  Show/hide the Workspace window
-  Show/hide the Output window
-  Manage all the open documents
-  Find text in multiple files
-  Load program to device
-  Compile the file
-  Build the project
-  Stop building
-  Start or continue the program
-  Download program
-  Insert/remove software breakpoint
-  Remove all software breakpoint
-  Insert/remove hardware breakpoint
-  Remove all hardware breakpoint
-  Break program running
-  Stop debugging
-  Restart running program
-  Step into next statement
-  Step over next statement
-  Step out current statement
-  Run to current cursor
-  Show/hide the Watch window
-  Show/hide the Register window
-  Show/hide the Memory window
-  Show/hide the Disassembly window
-  Show/hide the Command window
-  Show trace buffer
-  Clear trace buffer

22.3 In-Circuit Emulator

ECMC653A is a powerful emulation chip which has complete function for emulating SPMC65X Family general purpose microcontrollers and integrates Sunplus ICE. ‘SPMC65X Family EMU Board’ is the emulation development board to emulate entire SPMC65X family microcontrollers, and it has the necessary connectors to expand user self hardware system, meanwhile, there is also an ICP (In-Circuit Programmer) on the EMU board can implement on line programming function. The complete emulation system let user finish all design follow from coding, debugging, emulating, to programming device. Sunplus establishes an overall development environment which combines SPMC65X Family Emulator and FortisIDE™ to let user develop product design follow.



Fig. 22-1 SPMC65X family Emulator

22.3.1 Hardware Architecture of Emulation Board

The hardware block diagram of SPMC65X Family EMU Board is as follows, which consists of ECMC653A, power supply, ICE probe, oscillating circuit, ICP circuit and extended connectors.

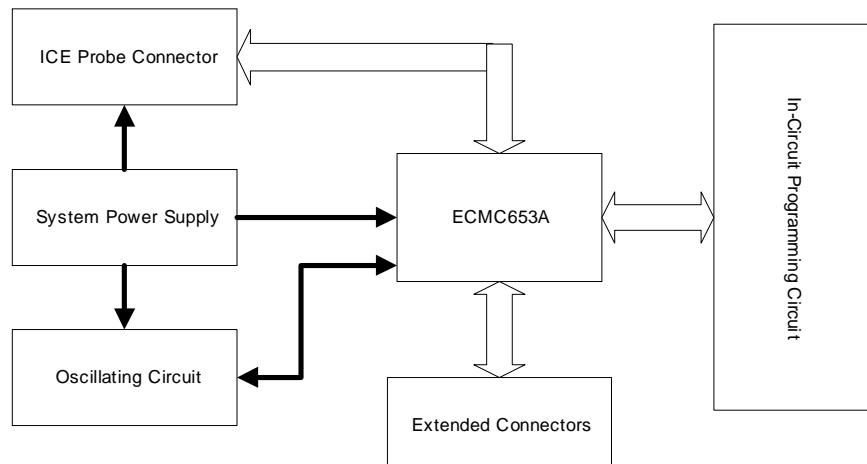


Fig. 22-2 SPMC65X family EMU board Hardware Block

(1) ECMC653A

There are PLCC68 and LQFP80 packages on EMU board. Sunplus will provide several EMU boards and user can choose appropriate object according to cost or convenience consideration.

(2) Power supply

The power circuit supplies 5V and 3V3 to ECMC653A, ICE probe, and extended connectors. The default working voltage is 5Vdc converted from the adaptor input voltage; besides, EMU board can also output power to user self target board or accept external power input via on-board connector.

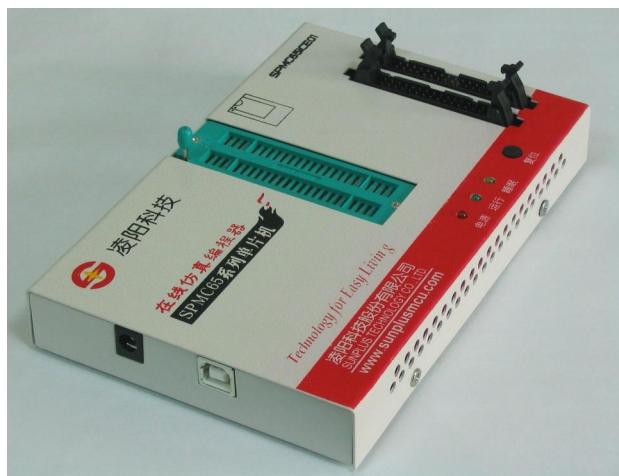


Fig. 22-3 DC Outlet and USB Socket

(3) ICE Probe Connector

There are three kinds of ICE probes which are USB probe, printer port probe, and simple probe provided for user. SPMC65X family EMU board supports simple probe using general USB interface and it can be connected with EMU board. Simple probe is a Sunplus' new developed ICE probe to provide user lower cost and more convenient selection.

(4) Oscillating Circuit

ECMC653A supports three kinds of clock sources which are crystal or ceramic resonator, RC oscillator, and external clock source. SPMC65X family EMU board supports crystal or RC oscillator as the clock source. The default clock source is a 16MHz crystal. User can select the suitable clock source for product development.

(5) ICP Circuit

Sunplus first invents a real-time serial programming interface integrated with ECMC653A. To cooperate with external application circuit, Sunplus build an ICP system on the EMU board. User can program binary code generated from FortisIDE™ to a blank SPMC65X family microcontrollers through the EMU board. In addition, Sunplus built an application program for operating ICP called 'Q-Writer'. Q-Writer has a friendly and handy operation interface similar to Sunplus OTP_MTP_Writer. User can start the program independently or via FortisIDE™ to call the program. There is a connection interface between FortisIDE™ and Q-Writer, but user just can operate an application program in the meantime.

(6) Extended Connectors

SPMC65X family EMU board reserves some extension connectors for connecting user self hardware board and

SPMC65X family daughter board. User can use the emulated daughter boards which are pin compatible with SPMC65X family microcontrollers to connect to product hardware board for system development. Meanwhile, the EMU board also has additional connectors to be the all IO port extensions.

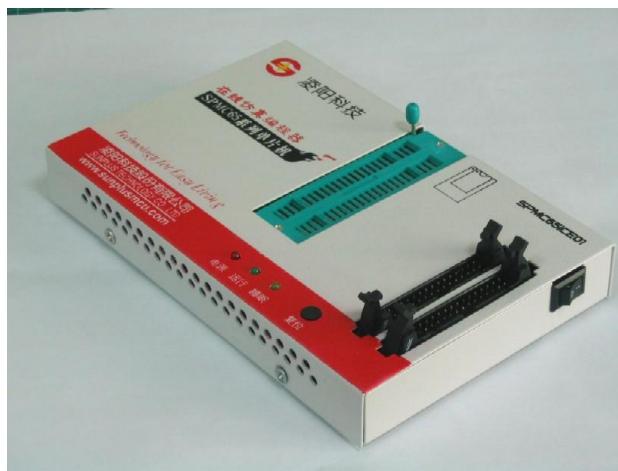


Fig. 22-4 Extended Connector and Power Switch

22.3.2 Operation Description

In order to let user understand SPMC65X family EMU board and use it properly, this section describes how to set the hardware configuration.

n Power Supply

There are two kinds power supply modes for SPMC65X family EMU board. One is to provide the working power from DC-9V power adaptor; another is to use external power from user's target board.



Fig. 22-5 DC-9V Adaptor

A switch shown as the following picture is used to switch the supply power. While switching to "O" side, the supply power of SPMC65X family EMU board is independent of user's target board. The working power of EMU board is 5Vdc converted from DC-9V adaptor input. In this operation, EMU board and user's target board have a common ground and independent power supply.

While switching to "I" side, EMU board and target board have the same VCC and GND. User can select the working power supplied from DC-9V adaptor or user's target board. If the working power is supplied by target

board, SPMC65X EMU board will have a over-voltage protection circuit to prevent damaging ECMC653A while the voltage exceeds 6Vdc.

If the supplied power from target board is under 3.3Vdc, then the simple probe will not work and user cannot run FortisIDE.

n Oscillating Circuit

SPMC65X family EMU board provides two kinds of clock sources for project development. One is quartz crystal or ceramic resonator, and another is RC oscillator. SPMC65X family EMU board provides crystal as the default operating clock source.

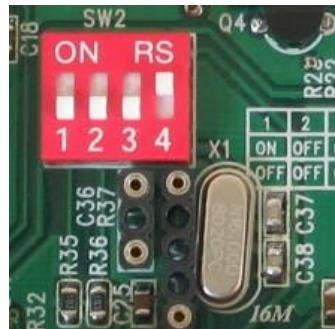


Fig. 22-6 Crystal

If user wants to use RC oscillator, user must open the housing first and then follow the steps to complete the setting.

1. Select the applicable resistor and capacitor, and then place the resistor at R37 and the capacitor at C36.
2. Change the configuration of DIP switch (SW2) as the indicative table.



Fig.22-7 RC oscillator

n ICE Probe

SPMC65X family EMU board provides simple probe for debugging in the development environment. User can plug in the probe to execute FortisIDE™ application program as the following picture. SPMC65X family EMU board provides a simple interface to connect with computer via the USB transmission line.

n Extended Connectors

SPMC65X family EMU board provides an extended interface to connect with SPMC65X emulated daughter board as the following picture. User can use SPMC65X emulated daughter board to emulate desired microcontrollers and connect to user's hardware board for product development. The IO pin placement is shown on the housing top. If user wants to provide the working power to SPMC65X family EMU board, the target board shall provide the

appropriate voltage and be connected to the input terminals according to the illustration.

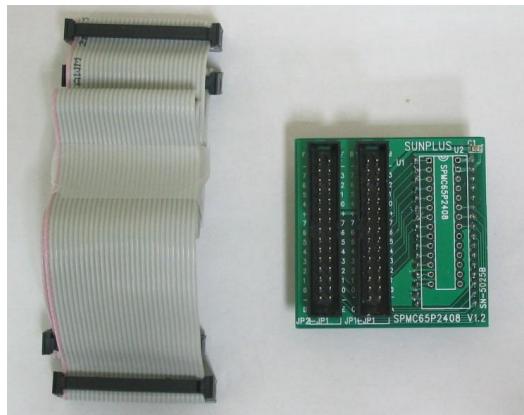


Fig.22-8 SPMC65X emulated daughter board

22.3.3 Real-time serial programming

Sunplus provides an easy and useful well real-time programming tool called 'Q-Writer' for SPMC65X family development. Q-Writer is implemented using ICE interface which is integrated with EMC653A and programming algorithm. After user has finished the project design in the FortisIDE™, they can continue to program project code to the selected SPMC65X microcontroller through Q-Writer and verify the result in user's hardware board immediately. It is convenient and reliable to help user product development. The detailed description is written in next chapter.

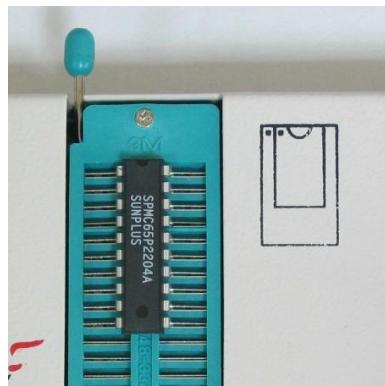


Fig.22-9 In-Circuit Programming fixture

22.4 Related Application Notes and Libraries

This section lists some application notes and libraries about various usage of SPMC65X family. The application notes may help user develop quickly and offer the concepts about the usage of this topic. Sunplus also supports some powerful libraries about general operation, mathematics, communication, etc. The purpose of library is to help user speed up development. The current application notes and libraries related to Developed Tool are listed below.

Application Note:

Title

Application Note Series Number

No associated application notes at this time.

Library:

Title	Application Note Series Number
SPMC65X series software basics manual	AN_O0100.DOC
SPMC65X series software advances 1 manual (data processing operation)	AN_O0101.DOC
SPMC65X series software advances 2 manual (mathematical operation)	AN_O0102.DOC

22.5 Revision History

Revision	Date	Remark
V0.1	03/31/2005	First edition

23 In-Circuit Programmer

23.1 Description

This chapter illustrates the types of programmer and how to use them. Sunplus always makes efforts in providing user more convenient and reliable programming tool. At the moment, Sunplus provides two kinds of programmers and also requests the third-party manufacturer to design gang-writer for mass production. One is the new programmer called Q-Writer combined with SPMC65X family EMU board, and another is the original Sunplus ALL-Writer. This chapter will describe their definition and usage of serial number and product information; moreover, it also introduces how to set the security via Sunplus writers.

23.2 Q-Writer

23.2.1 General Description

The Q-Writer is a Sunplus new developed programming tool which comes with EMC653A and is integrated with FortisIDE. The tool let user be able to complete all design flow in the SPMC65X family development environment. Q-Writer is built with a friendly operation interface and concise hardware architecture. The new generation programming tool provides many advantages like more convenient usage, high reliability and low cost etc.

23.2.2 Integrated Hardware Design

Q-Writer is a tool based on SPMC65X family EMU board. Sunplus uses ICE interface and external power supply to implement the serial programming operation. EMC653A controls the serial programming signals and enables external power supply. External power supplies VDD and Vpp for entering EPM mode.*

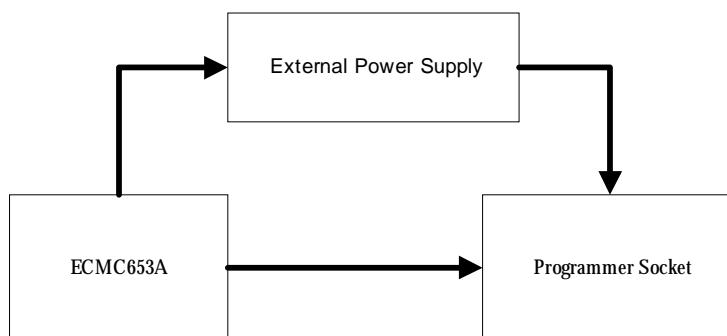


Fig.23-1 ICP Block Diagram

* In normal state, OTP is in MCU mode; while programming the device, OTP will enter EPM mode if VDD is 6Vdc and Vpp is 13Vdc.

23.2.3 Installing and Starting

Since the new generation development tool, FortisIDE will have two version numbers. One is for application program, and the other is for chip management. User can see the relative information as the following picture.



Sunplus provides a high-integrated software package for user to install FortisIDE, FortisIDE Body, and Q-Writer at the same time. Of course, Sunplus also provides the independent installing package to maintain the three software sets. Q-Writer software is packaged with the latest FortisIDE development tool and the body version of SPMC65X family. Therefore, FortisIDE supports Q-Writer function since V1.6.5. Generally, Q-Writer can be run on Windows95®, Windows98®, Windows 2000, Windows NT®¹ and Windows XP®.

n **Installing**

The Q-Writer should be installed by following the below procedures.

1. Run Q-Writer V1.0.1.exe and the welcome screen is shown as follows:

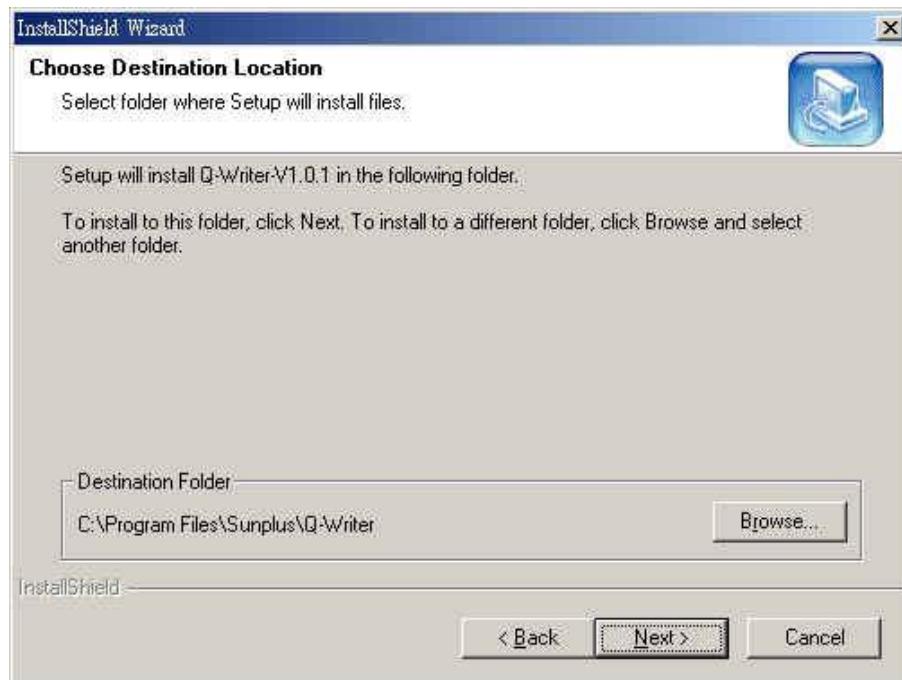


Note: V1.0.1 is the software package version. The package version may be modified following the

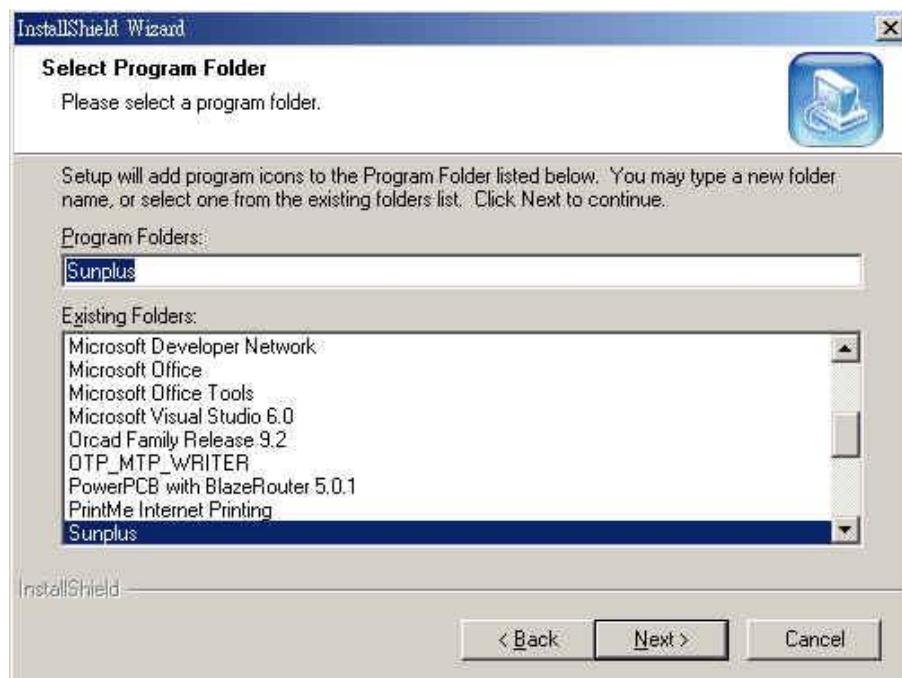
¹: Windows and Windows NT are registered trademarks of Microsoft Corporation. The names of actual companies and products mentioned herein may be the trademarks of their respective owners.

software release.

2. Follow the instruction on the screen to install Q-Writer on your computer.
3. Select installing path and program folder. (Suggest using the default strongly.)



4. Select "Next" ,start to setup.



5. Select [Start Menu] → [Program] → [Sunplus] → [Q-Writer] → [Q-Writer-V1.0.1.exe] to run Q-Writer application program.

n Starting

Connect the SPMC65X family emulator to the PC properly via the suitable probe. Then start the Q-Writer application program. There are two methods to start Q-Writer.



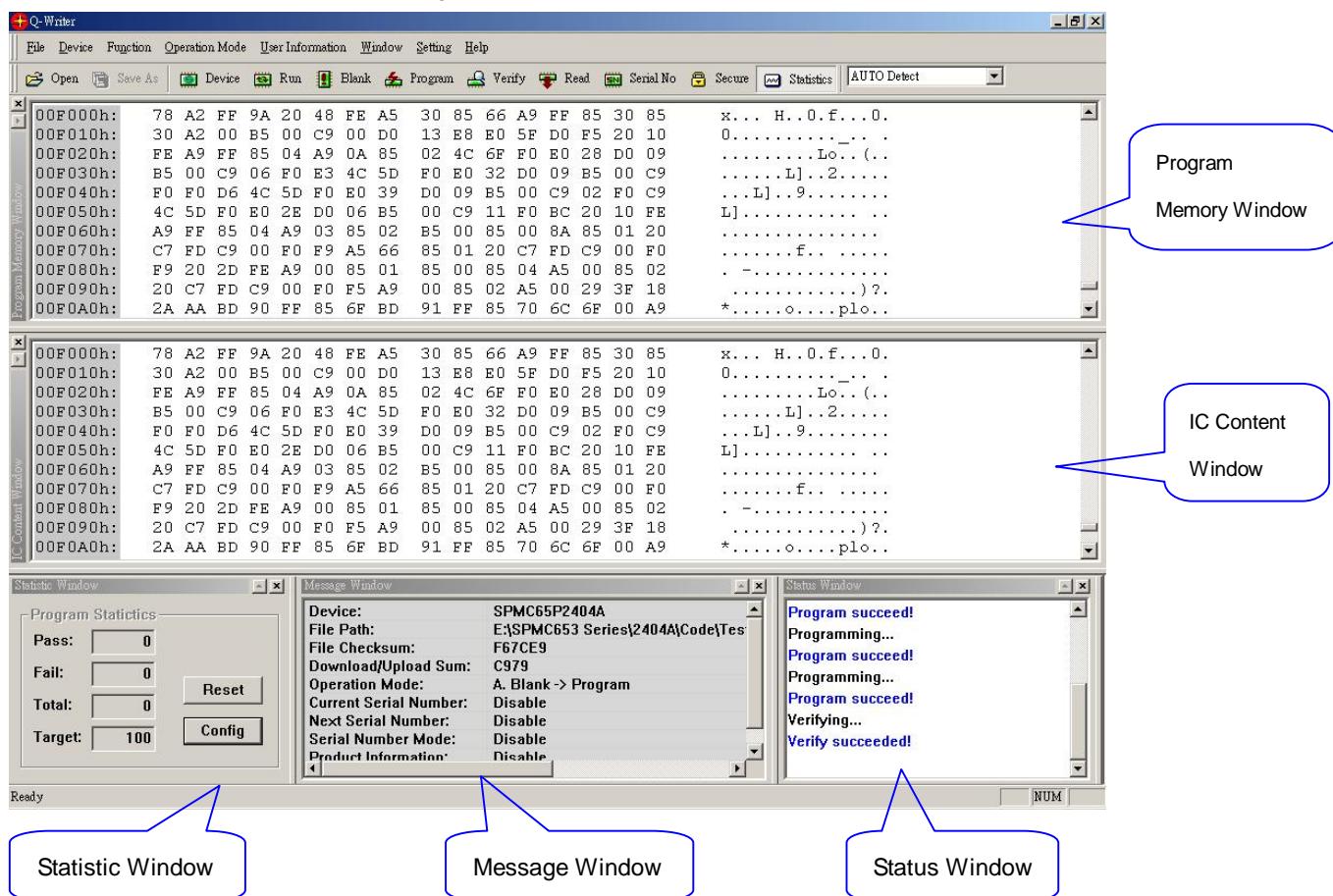
1. After opening FortisIDE, user can start Q-Writer by pressing  on the toolbar or select [Tools] → [Q-Writer].
2. Execute Q-Writer-V1.0.1.exe directly.

Note!

When starting Q-Writer, FortisIDE is forbidden to enter debug mode. Even if FortisIDE has gone into debug mode, it still goes back to normal state as long as starting Q-Writer.

23.2.4 How to use Q-Writer

The Q-Writer window is as the following picture which consists of menu, toolbar, program memory window, IC content window, statistic window, message window, and status window etc.



n Menu

The menu of Q-Writer includes File, Device, Function, Operation Mode, User Information, Window, Setting, and Help. The following description illustrates the function and usage for each of them.

–File

Ø Open (Ctrl+O): Open and download the *.bin or *.tsk file to “Program Memory Window”.

Ø Recent Files: Record at most 4 last opened files.

Ø Exit (Ctrl+E): Exit Q-Writer. While closing the software, FortisIDE can operate in debug mode.

–Device

Ø Select: This command will open the device selection menu as follows. User must select a device after executing Q-Writer at the first time. The system also records the last selection device in the message window; hence the last selection device is the default setting if Q-Writer is restarted.

**–Function**

Q-Writer supports six kinds of operation functions. If user doesn't download the binary file ahead, Q-Writer will not enable Auto Run, Program, and Verify functions.

Ø Auto Run: This command is used for continuous operation only. User must select an operation mode before execute this command. While in process, the system must complete the previous action then continuously run the following action. If error happened, the system stops operating.

Ø Blank Check: Check whether IC is blank or not. At the same time, Q-Writer reads out the IC data to IC content window. If this function failed at some address, the system goes to the wrong address and highlights the data with red in the IC content window.

Ø Program: This command is used to program OTP device according to the content of program memory window. At the same time, Q-Writer reads out the IC data to IC content window. If this function failed at some address, the system goes to the wrong address and highlights the data with red in the IC content window. Once executing this function, Q-Writer also executes verify function to ensure the programming reliability in the meanwhile.

Ø Verify: Verify function is used to check the data of source code and target OTP. While verifying IC, Q-Writer reads out the IC data to IC content window. If this function failed at some address, the system goes to the wrong

address and highlights the data with red in the IC content window.

Ø Read: Read data from IC and upload to IC content window.

Ø Secure: This command is used to enable/disable IC encryption. After setting up this function, the device will be secured and not be allowed to read. User must program and verify device before starting security operation. SPMC65X family microcontrollers just allow the IC content of some addresses to be read, which like device configuration registers, user information, and the last 16 bytes of program memory (\$FFF0~\$FFFF). The others are shown as \$00.

Before starting security job, Q-Writer must execute blank check function to prevent mistake for raising system reliability. If OTP is blank, system will stop securing.

–Operation Mode

This menu is used to select the continuous operation mode. There are three kinds of continuous operations as follows. In this mode, the system must finish the first task then can be able to do the next.

Ø A. Blank à Program

Ø B. Blank à Program à Verify

Ø C. Blank à Program à Verify à Secure

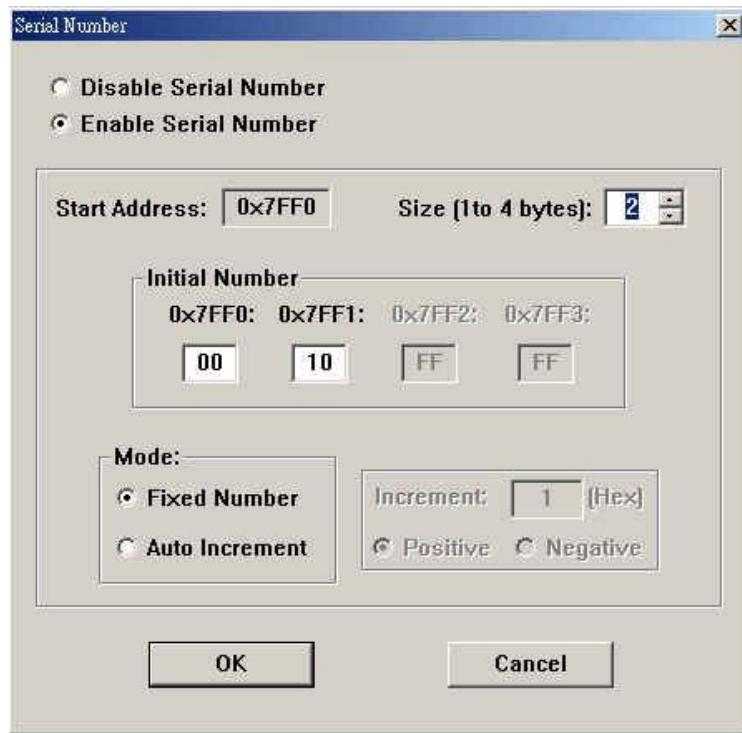
This mode makes Q-Writer operate all programming flows and secure OTP after verifying correctly. After finishing the task, the data of program memory can not be read.

–User Information

This function allows user to set the product information in programmed code. Q-Writer provides two functions to set the data of user information area.

Ø Serial Number

Q-Writer uses 4 bytes to be the serial number locations from \$7FF0 to \$7FF3.



- ✓ **Start Address:** The start address of serial number is \$7FF0. \$7FF0 is the LSB and \$7FF3 is the MSB.
- ✓ **Size (1 to 4 bytes):** User can decide how many bytes of serial number to be used through setting this item.
- ✓ **Initial Number:** The item is used to input the preliminary serial number. When user finishes the setting, the input data will update to program memory window. The default value of unused bytes is \$FF. For example, if the previous serial number setting used three bytes and user changes those to two bytes, then the system will restore the unused byte to \$FF.

Previous â 007FF0h: 00 10 20 FF FF FF FF FF
Next â 007FF0h: 00 10 FF FF FF FF FF FF

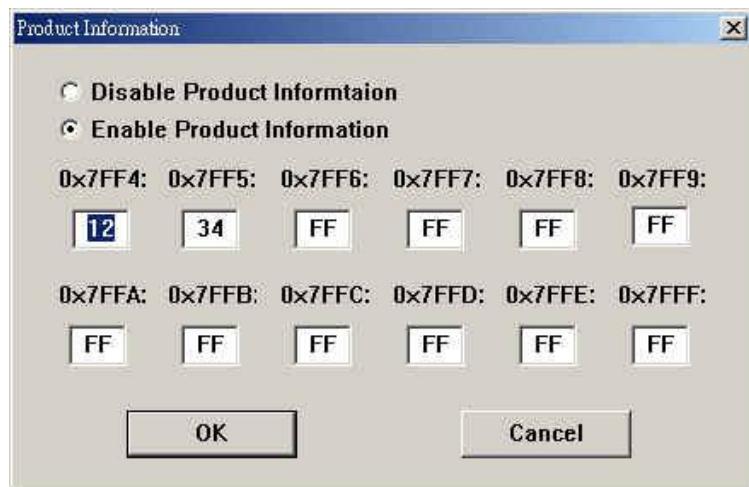
- ✓ **Mode:** The item lets user be able to operate the serial number setting with fixed mode or auto mode. In fixed mode, the increment item becomes gray area and the serial number is fixed as initial number setting; in auto mode, user can decide the increment value and the serial number has regular variation in accordance with this value. There are two ways to change the increment value, which are positive or negative counting.



Ø Product Info:

\$7FF4~\$7FFF total 12 bytes are used to be the product information location. User can use free those

locations to record configured product information such as production date and factory. The default value of those bytes is \$FF.



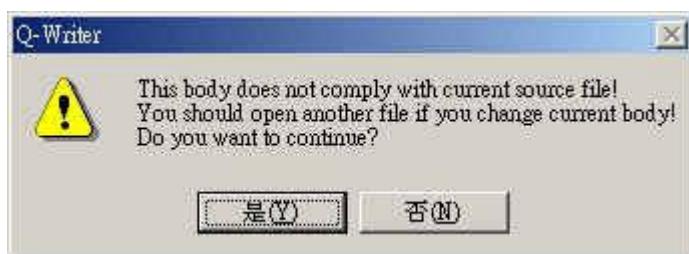
—Window

There are five windows assisted user to operate Q-Writer. When pressing down the left key of mouse, the frame of selected window becomes blue. To open or close any window, user can enable/disable the corresponding item by window menu.

Ø Program Memory Window: This window records the downloaded binary file and it is also the source for programming.

Note!

If user starts Q-Writer from FortisIDE, the system will automatically download the binary code of developing project to program memory window. This design makes user program OTP and verify project code very conveniently. Be careful that if user changes the selected body without redownloading the source code, the system will show a warning message as follows to remind user. While the selected body is not equal between FortisIDE and Q-Writer, and furthermore the data content of program memory window has not been updated yet, Q-Writer will appear this message.



User can also right click the mouse to operate some functions as follows.



C Goto Address: Jump to any address located from \$0000 to \$FFFF.

Ø IC Content Window: This window records the content of OTP. While something error happened during operation, the window shows the error address and read back data.

Ø Statistic Window: This window is used to manage the programmed quantity.

I. Editbox

z Pass: Record the quantity of programmed successfully IC.

z Fail: Record the quantity of programmed failure IC.

z Total: The sum of pass and fail IC.

z Target: Set the expectative programmed quantity. When total quantity reaches the setting of target number, the system will have a message window to ask if clear the setting.

II. Push button

z Reset: Clear all statistical items except for target number. While pressing down the button, the system shows the message box as the following to double confirm the action.



z Config: Set the quantity of programmed target.



Ø Message Window: To enable or disable exhibited items of this window by selecting the checkboxes of option.

X Device: Indicate the selected device name.

Note!

After installing Q-Writer application program on your computer, this item is blank and user has to select applicable device while starting Q-Writer at the first time. In general, this item shows the last selected device name. If user starts Q-Writer from FortisIDE, the default setting of device name is the same as the selected body from FortisIDE.

X File Path: Indicate the file path of downloaded code.

X File Checksum: Calculate the checksum of whole downloaded file. The calculated range is 64K bytes from \$0000 to \$FFFF. File checksum is the same as the checksum calculated by FortisIDE.

X Download/Upload Sum: This is a calculating checksum for the IC content window. The calculated range consists of program memory , device configuration registers and user information bytes.

Note!

Programmer often uses 'DB', 'DS', and 'DW' to define the variable space in the project code. If using 'DB' or 'DW' to define the variable, the FortisIDE compiler will assign \$01 or \$00 in the RAM area. If using 'DS', the compiler will assign \$FF in the RAM area. To retain the equivalent file checksum, Sunplus strongly suggests that user should use '.DS' directive to name the variables.

X Operation Mode: Indicate the continuous operation mode.

X Current Serial Number: Record the current serial number value which format is hexadecimal. If user does not enable the serial number function, the item will show 'Disable'.

X Next Serial Number: Record the next serial number value which format is hexadecimal. If user does not enable the serial number function, the item will show 'Disable'.

X Serial Number Mode: Indicate the increment mode. If user does not enable the serial number function, the item will show 'Disable'.

X Product Information: Indicate that product information is enabled or disabled.

Ø Status Window: Q-Writer shows all of the operating state in this window. User can see the whole operation procedure clearly.

-Setting

Ø Option: Enable this function to open the following dialog for message management. If user unchecks the security setting selection, the system will not display a confirmation window before securing IC.



Ø Program Statistic: The function is the same as the icon of toolbar to enable/disable statistic function.

Ø 16 Bytes per Row: The function controls the arrangement amount of per row. If enabling the function, per row of memory window shows 16 bytes data.

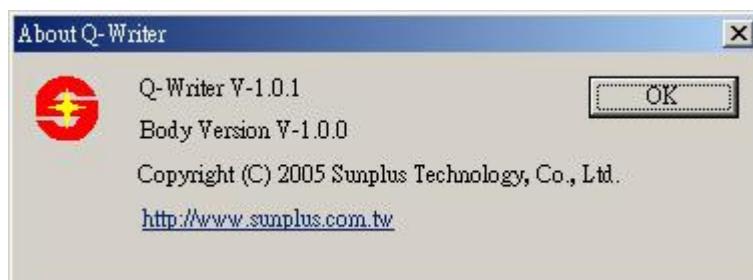
Ø Show ASCII Code: Show the corresponding ASCII code against the binary data of memory window.

00F010h:	01 A9 FF 85 30 85 30 20 84 FD C9 00 F0 F9 A9 000.0
00F020h:	85 01 A5 00 85 02 20 84 FD C9 00 F0 F5 A9 00 85
00F030h:	02 A5 00 29 3F 18 2A AA BD 90 FF 85 76 BD 91 FF	...)?.*.....v...
00F040h:	85 77 6C 76 00 A9 00 AA 8A A8 98 48 68 85 01 20	.wlv.....Hh..
00F050h:	84 FD C9 00 F0 F9 A9 55 AA 8A A8 98 48 68 85 01U.....Hh..
00F060h:	20 84 FD C9 00 F0 F9 AA AA 8A A8 98 48 68 85U.....Hh..
00F070h:	01 20 84 FD C9 00 F0 F9 A9 FF AA 8A A8 98 48 68Hh
00F080h:	85 01 20 84 FD C9 00 F0 F9 38 A9 55 E9 55 D0 448.U.U.D
00F090h:	A9 01 85 01 20 84 FD C9 00 F0 F9 38 A9 55 E9 568.U.V
00F0A0h:	10 32 A9 02 85 01 20 84 FD C9 00 F0 F9 18 A9 FF	.2....
00F0B0h:	69 02 90 20 A9 04 85 01 20 84 FD C9 00 F0 F9 18	i...

-Help

Ø Help Topics (F1): Q-Writer supports on-line help function to search the related topics.

Ø About Q-Writer: Indicate the software version of Q-Writer and provide the connection path to Sunplus website. The software version is divided into application program and body firmware. User can find and download the newest software in Sunplus website.



n Toolbar



Open the source file with binary format. User must download a binary code to enable all of Q-writer functions .



Save the content of program memory as another path.



Select device. User has to select a correct device to operate Q-Writer.



Execute auto run (continuous) function.



Execute blank check function.



Execute program function.



Execute verify function.



Execute read function.



Enable serial number function.



Execute security function.



Enable/Disable statistic function.



Select probe. There are three kinds of items to select the suitable probe.

23.2.5 System Message

All data will be cleaned to 0, do you want to continue?

Ú While user presses down the reset button of statistic window, the message box will double confirm whether clearing all statistical items except for target number.

Can not detect any probe!

Ú The system can not detect any plugged probe on EMU board. User must check whether the probe is connected well to EMU board or EMU board has powered on. During the operating process, if the USB probe is unplugged, then the system will show the following message.



Sunplus provides a thoroughly considered design to protect IC for preventing suddenly destruction during the programming process. Even if the USB probe is unplugged during programming process, the IC can be

still programmed with the previous code after re-plugging the USB probe. Before operating again, user must reselect the applicable probe by clicking the list box of probe selection to enable the Q-Writer function.

☞ EMU board is not ready. Please check if EMU board is set or connected properly.

Ú The message shows that the selection probe does not match with the plugged probe or EMU board has some problems. If the probe selection item is 'Auto Detect', the system will detect plugged probe automatically.

☞ File cannot be opened to save!

Ú It means user can not save as the current file. Maybe there is something wrong with the computer or the attribute of access file.

☞ IC has been secured already!

Ú Before starting security task, the system will check whether IC has been secured or not. If IC has been secured, the system will show the message and stop securing.

☞ Incorrect file format!

Ú User just only can download .bin or .tsk file to Q-Writer. The message tells user that the format of downloaded file is not acceptable.

☞ No body is selected!

Ú When user starts Q-Writer at the first time and does not select a device in advance, the system will show the warning message and then open the device selection menu automatically.

☞ Q-Writer is already opened!

Ú Only one copy of Q-Writer can be opened.

☞ The source file of program memory has been modified. Do you want to reload it?

Ú As long as the downloaded file of program memory window modified and saved by other program, the system will show the message to ask user.

☞ USB Probe detected but is used by other application!

Ú User should check if there is another application program also uses USB probe.

☞ You are going to add Security on IC, which is a non-reversible action. Are you sure to do this?

Ú The message shows that the system will execute securing action later. User must confirm that IC is ready to be secured. At the same time, the message window also has a checkbox to decide if displaying the dialog each time.

☞ You have reached the target amount! Do you want to reset Statistics Window?

Ú When the total number is equal to target number of statistic window, the system will show the message to remind user, and user can select 'Yes' to clean the statistical number. If selecting 'No', the system will ignore the message and not clean the statistic data.

☞ Your IC is blank! Securing is prohibited!

Ú The programmed IC is blank and the system does not allow a blank IC to be secured. User shall verify the IC has been programmed with code.

☞ Your Serial Number could have been used again!

U The serial number increases automatically in auto mode. While serial number is over the initial number and user continues the operation, maybe some ICs have the duplicate serial number.

23.3 OTP/MTP Writer

23.3.1 General Description

The OTP/MTP Writer is a programming tool for SUNPLUS MTP (Multi-Time Programmable IC) and OTP (One-Time Programmable IC). The writer system includes two primary parts: hardware boards and software tool (the programmer). We will discuss them in the later sections.

23.3.2 Hardware

Four components are needed to program an OTP/MTP: an OTP/MTP writer, a COM port cable, a DC-18V adapter and an adapt-board (varied for different IC).

Follow the steps below for hardware setup:

- Select an adapt-board and insert it into OTP/MTP Writer.
- Connect the OTP/MTP Writer and PC via a COM port cable.
- Supply power to the writer with a DC-18V adapter.



23.3.3 Adapt-board

For various IC series, a unique adapt-board is required. Due to Sunplus supplies a good design, SPMC65X family microcontrollers have the identical pin location for programming. For this reason, all of SPMC65X family microcontrollers are able to use the same adaptor board and this lets user maintain adaptor board easily.

23.3.4 Software Tool

The software programmer for OTP/MTP Writer is able to program majority of SUNPLUS OTP/MTP ICs. Generally, the programmer can be run on Windows95[®], Windows98[®], Windows 2000, Windows NT[®]¹

¹: Windows and Windows NT are registered trademarks of Microsoft Corporation. The names of actual companies and products mentioned herein may be the trademarks of their respective owners.

and Windows XP.

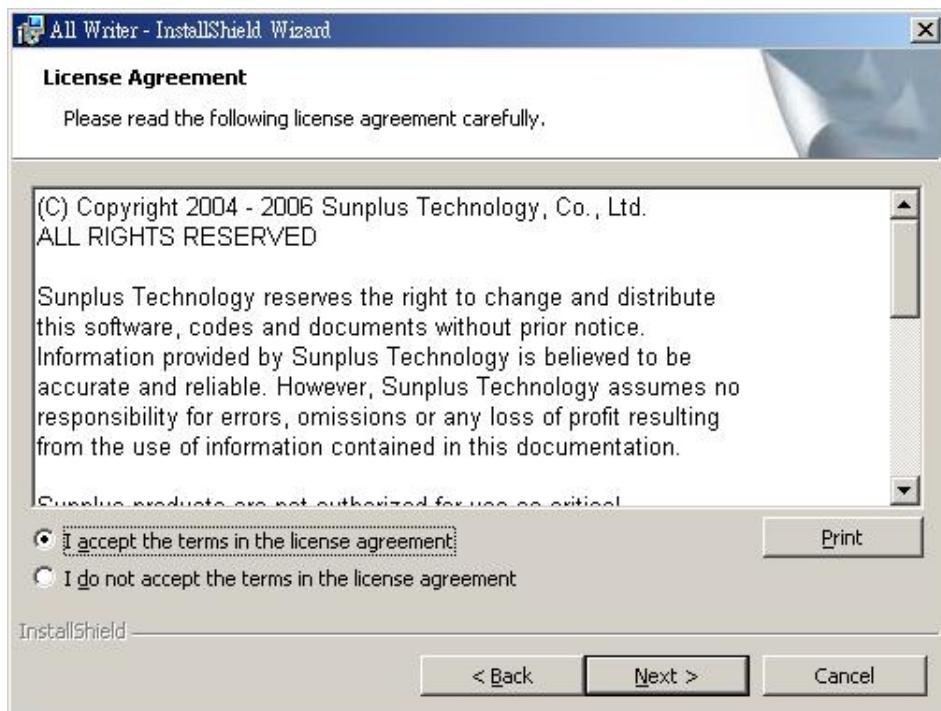
n **Installing and Setting up**

The Sunplus OTP/MTP Writer should be installed by following the below procedures,

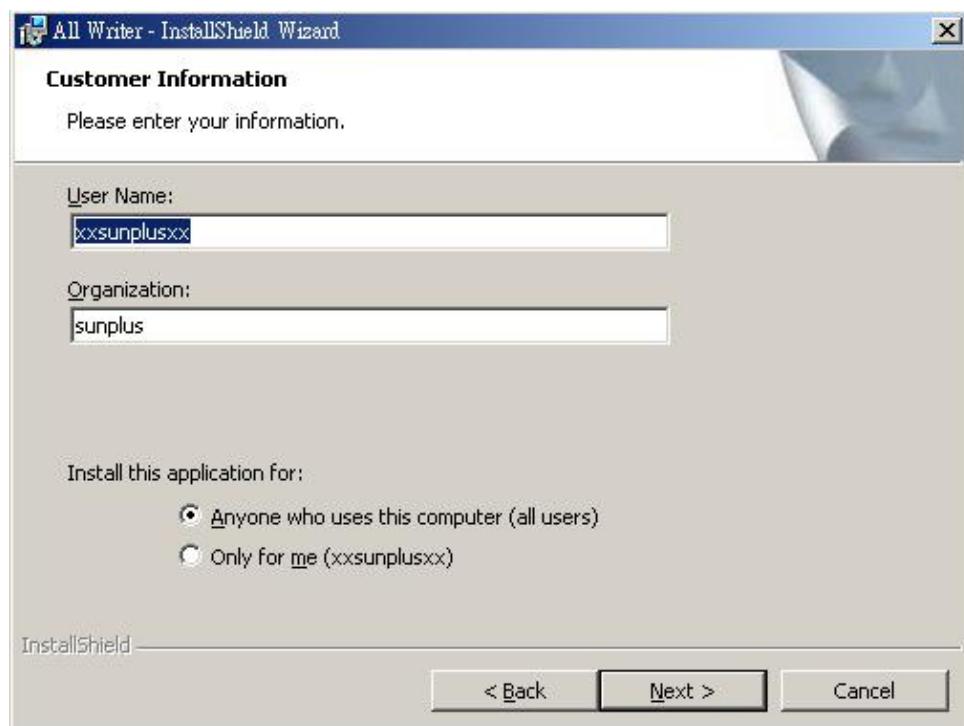
- Ø Click **Start**, point to **Settings**, click **Control Panel**, and then double-click **Add/Remove Programs**.
Open the **Install/Uninstall** tab, click **OTP_MTP_WRITER** in the list of installed programs, and then click **Add/Remove**.
- Ø Execute the “S+ OTP_MTP_Writer” installation package and select language



- Ø Follow the on-screen instructions to install the Sunplus OTP/MTP Writer software.



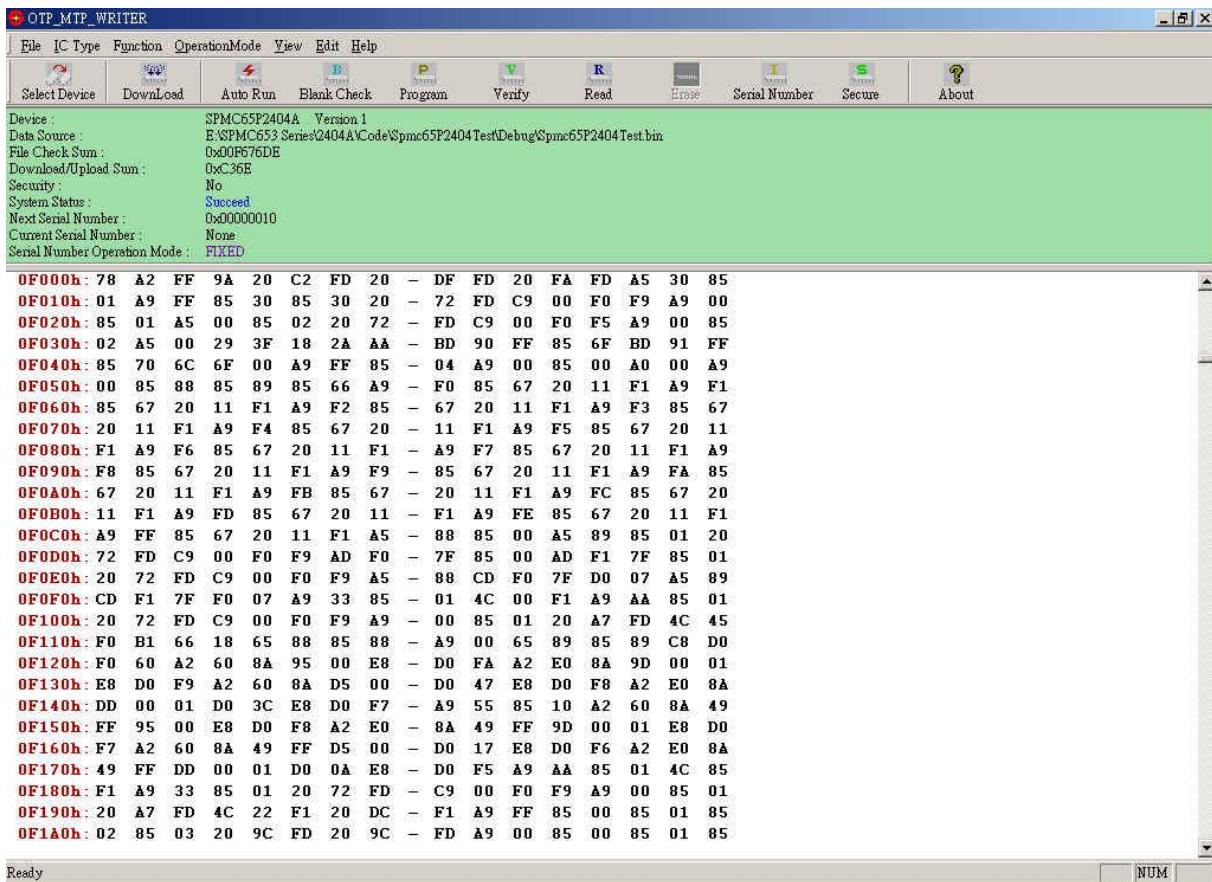
- Ø Fill in your name and company information



- Ø Check the installation directory of “OTP_MTP_WRITER.ini” by IC Type->Setting of the function list.
The file should be in the following path, C:\Program Files\Sunplus\OTP_MTP_Writer\library.



23.3.5 Function Description



File

- | **Download (Ctrl + D):** Opens a specified file and downloads program into board.
- | **Save (Ctrl + S):** Saves data to PC.
- | **Save As:** This command saves data to PC with a different filename.
- | **Exit:** Apply this command to exit from OTP/MTP Writer.

IC Type

I Select Device:

This command selects a device (use appropriate adapt-board) and downloads the corresponding resource file (*.das) to the board. This resource file is used to execute writer function, and it varies for different adapted boards. If a device is properly selected, a correct IC type will show on message window. On the other hand, "Not Selected" will show on message window if incorrect selection is made and therefore, OTP/MTP Writer function cannot be executed.

I Setting:

Set the resource files directory. The default directory is "C:\Program Files\Sunplus\OTP_MTP_Writer\library ". After user installing Sunplus OTP/MTP Writer AP and starting first time, maybe the system needs user to indicate the installation path of library. Please indicate the correct path in accordance with

the foregoing directory.

Function

I Auto Run (F5):

Program data of memory window to OTP/MTP device in accordance with the setting of "OperationMode".

This is a continuous operation.

I **Blank Check:** Check whether IC is blank or not.

I **Program:** Program data of memory window to OTP/MTP.

I **Verify:** Verify the data on the IC and data on the board RAM.

I **Read:** Read data from IC, and uploads data to PC.

I **Erase:** Erase data for MTP.

I **Serial Number:** Allow user to designate the serial number for mass production.

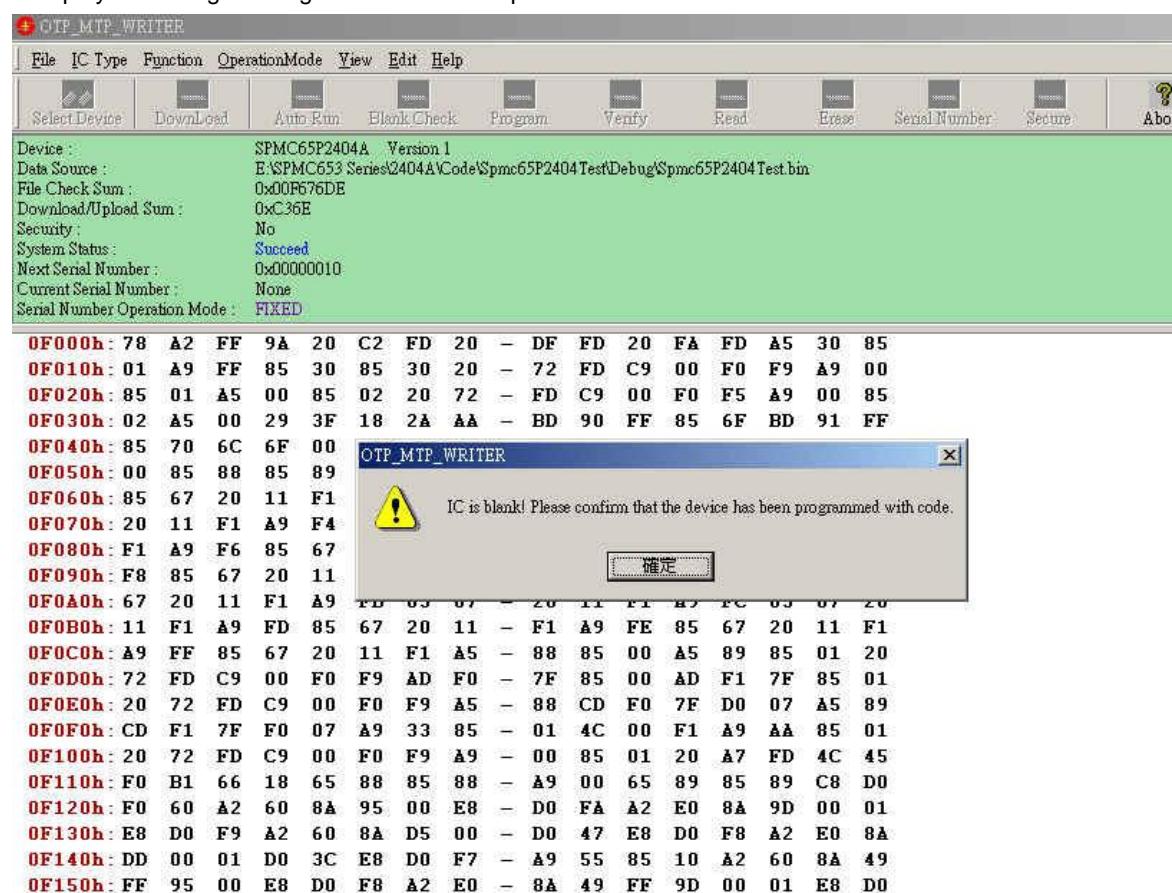
I S. Security:

This command is used to enable/disable IC encryption. After setting up this function, the device will be secured and not allow to read. User must program and verify device before starting security operation.

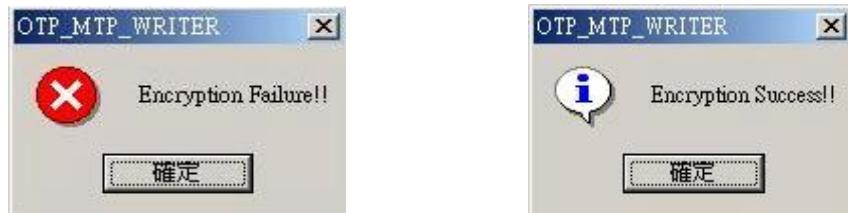
Expatiate on the operation as follows:

(1) The IC must have been programmed before operating this function; otherwise, Sunplus OTP/MTP

Writer will display a warning message and not allow to protect this IC.



(2) If the device has been programmed, Sunplus OTP/MTP Writer will execute programming security byte and verify whether the result is successful or not. There are two message windows to indicate that the operation result is successful or failure.



Write Mode

| **Blank Check->Program:** Executes blank check function, and then performs write function.

| **Blank Check->Program->Verify:**

This command executes blank check function, and then executes write function. Finally, run the verify function.

| **Blank Check->Program->Verify->Secure:**

continuously operate the blank check, program, verify, and protect function. It's a convenient operation to complete all of the writer function for user. Furthermore, user can see the each operating procedure clearly.

View

| **Toolbar:** Enable / Disable Tool bar

| **Status Bar:** Enable / Disable the status bar

| **Sunplus ID:** Show the checksum of upload/download data using Sunplus algorithm.

| **File Checksum:** Indicate the checksum of specified file.

| **Goto Address:** This command will show PC data starting from specified address.

Edit

| **Edit:** Edit data of memory window at PC side.

Help

| **About OTP/MTP Writer:** Indicates the version of the writer..

23.3.6 Serial Number Function

The serial number function allows user to specify serial number at specific address for mass production management. Some products need this function to assign different serial number of each OTP, example for remote controller. Sunplus OTP/MTP Writer provides a simple and easy use interface to setup the serial number and help to enhance the production efficiency. There are four specific addresses to be used for the serial number of SPMC65X family microcontrollers, which are \$7FF0 to \$7FF3. \$7FF0 is the

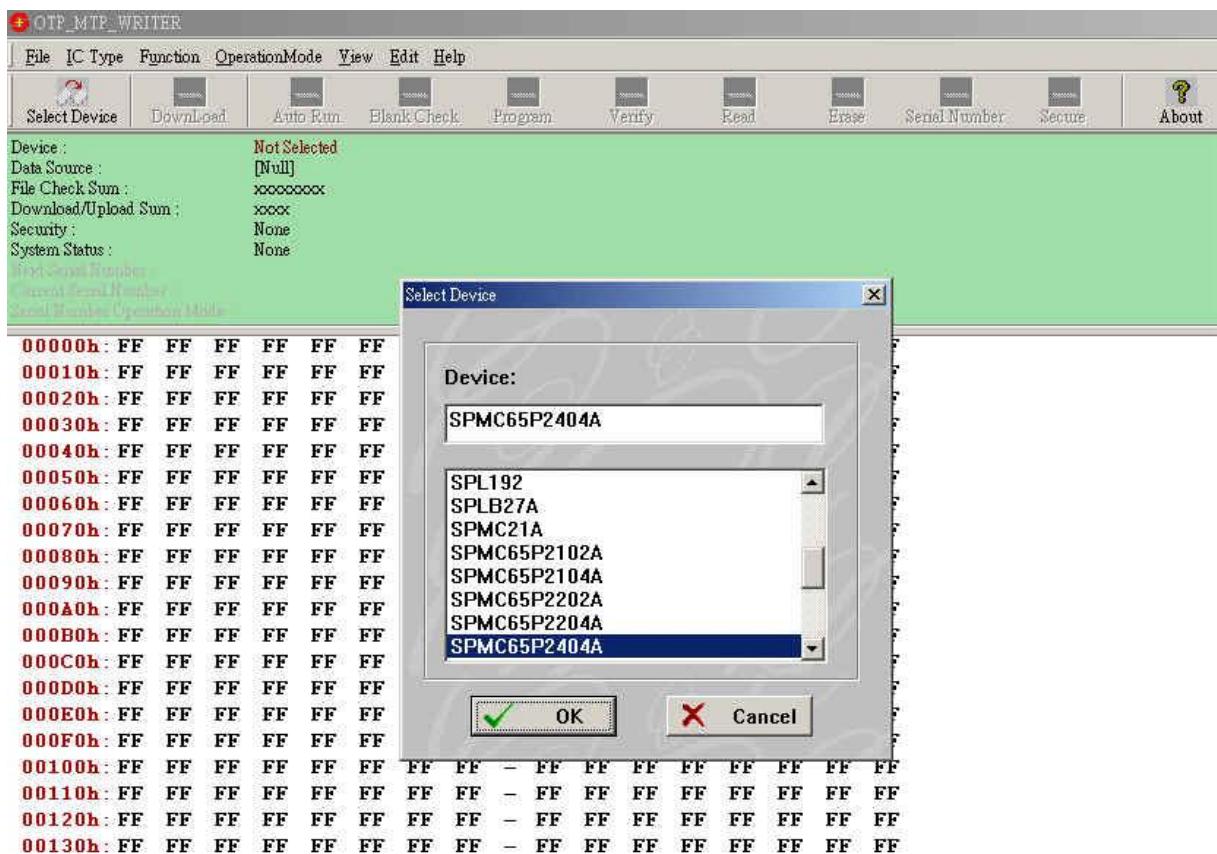
LSB and \$7FF3 is the MSB. Sunplus restricts the four bytes to record serial number. If user would not like to set serial number, the default content of the addresses is \$FF.

Normal State

OTP_MTP_WRITER		File	I ^C	Type	Function	OperationMode	<u>View</u>	Edit	Help	
Select Device	Download	Auto Run	Blank Check	Program	Verify	Read	Erase	Serial Number	Secure	About
Device : SPMC65P2404A Version 1										
Data Source : [Null]										
File Check Sum : xxxxxxxx										
Download/Upload Sum : 0x0000										
Security : No										
System Status : Succeed										
<i>Memory Address</i>										
<i>Content</i>										
00000h:	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
00010h:	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
00020h:	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
00030h:	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
00040h:	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
00050h:	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
00060h:	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
00070h:	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
00080h:	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
00090h:	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
000A0h:	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
000B0h:	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
000C0h:	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
000D0h:	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
000E0h:	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
000F0h:	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
00100h:	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
00110h:	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF

Operation Procedure

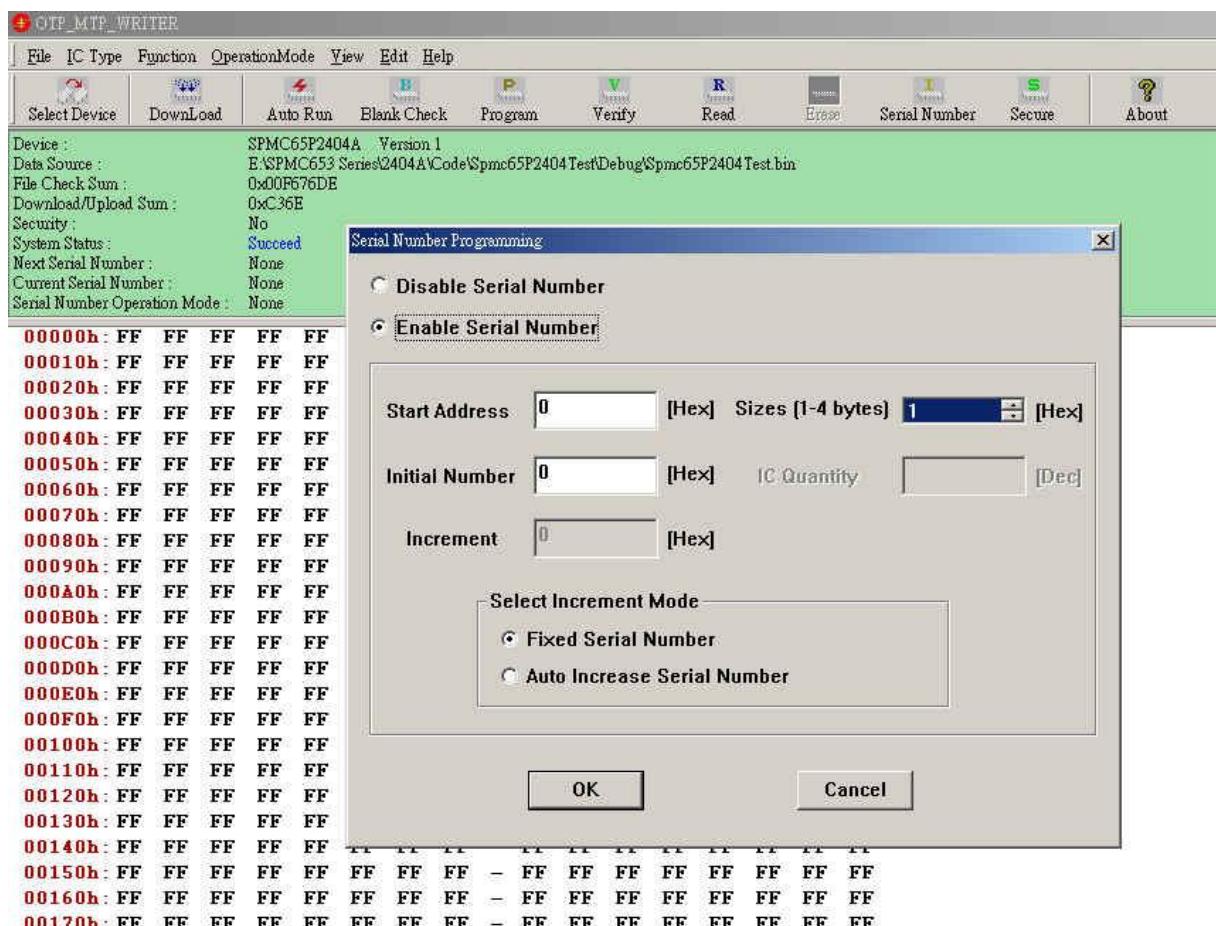
- Select supported OTP



■ Download source code

■ Enable serial number function

To enable this function select **Functions >> Serial Number** from the function menus or press **Serial Number** icon. All of the setting items are written by hexadecimal value and it allows upper case or lower case.



I Start Address

The default start address is \$7FF0.

I Sizes (1-4 bytes)

Choose the byte size of serial number. The maximum size is 4 bytes.

I Initial Number

Input the initial serial number.

I IC Quantity

Unused in this version.

I Increment

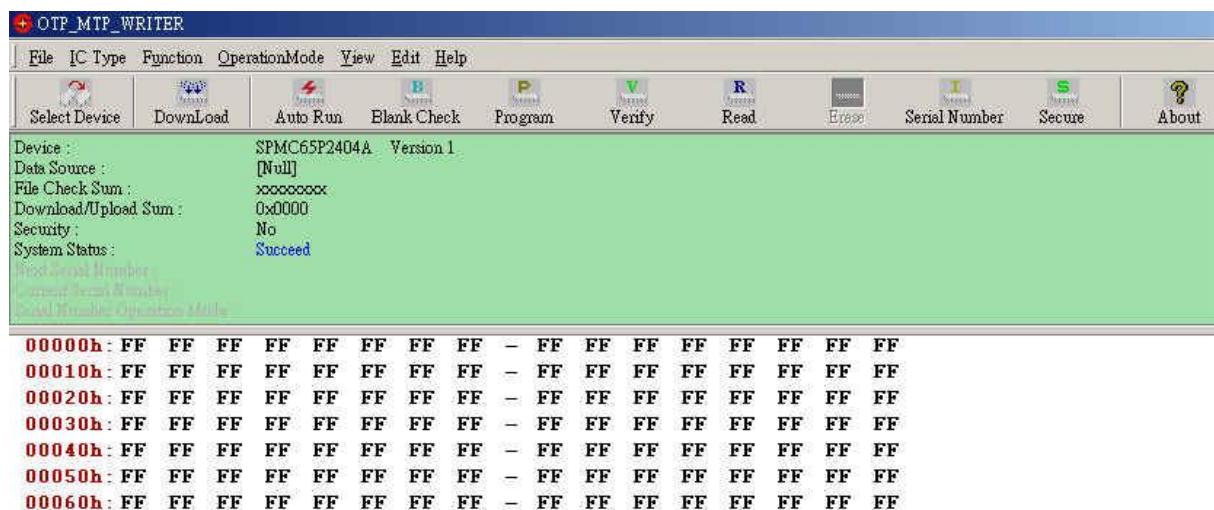
Setting up the increment value. It's useful while selecting auto mode.

I Mode Selection

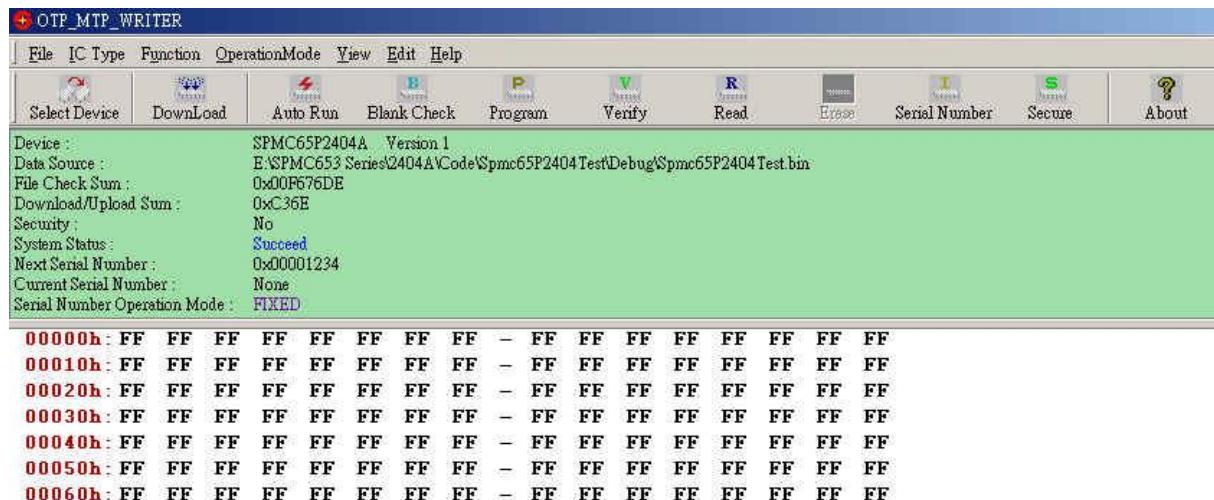
There are two operation modes for setting the serial number, one is fixed mode, and the other is auto mode. When selecting fixed mode, the increment item becomes gray area and the serial number is fixed; when selecting auto mode, user can decide the increment value and the serial number has regular variation in accordance with the increment.

n Message Window

The message window that is the green area as following picture shows all the operation information of Sunplus OTP/MTP Writer.



There are three list items about serial number operation in the message window. In the initial state, user needn't download the source code and the items are disabled (gray color). While operating the function first time, the entire list items about serial number show "None" as the previous picture.



While setting up the function, user can see the related information in the window.

I Next Serial Number

Record the prepared serial number for programming, but it has not been written into any OTP.

I Current Serial Number

Record last programmed serial number. The item shows "None" at the first time.

I Serial Number Operation Mode

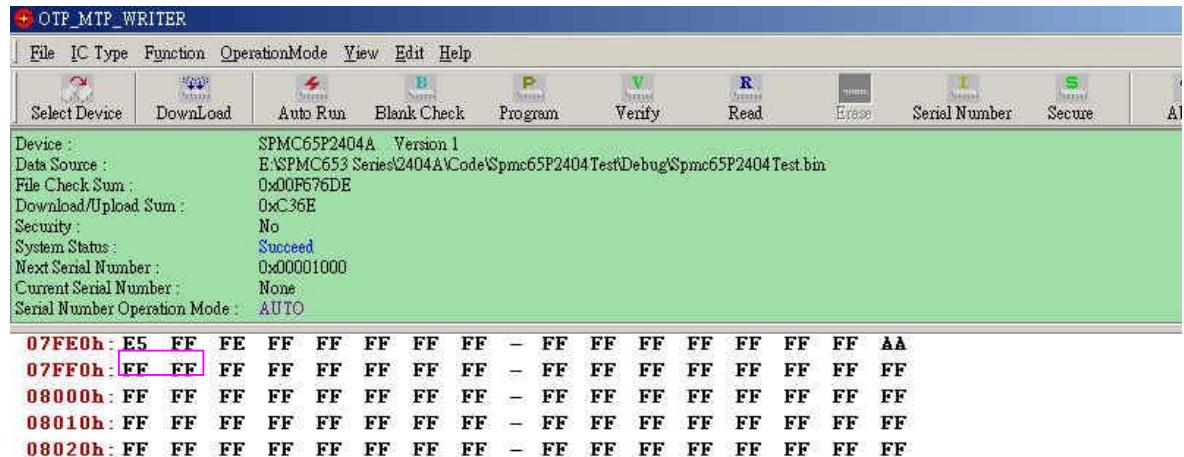
The item indicates the selected operation mode is "Fixed" or "Auto".

n Example

I Input Conditions

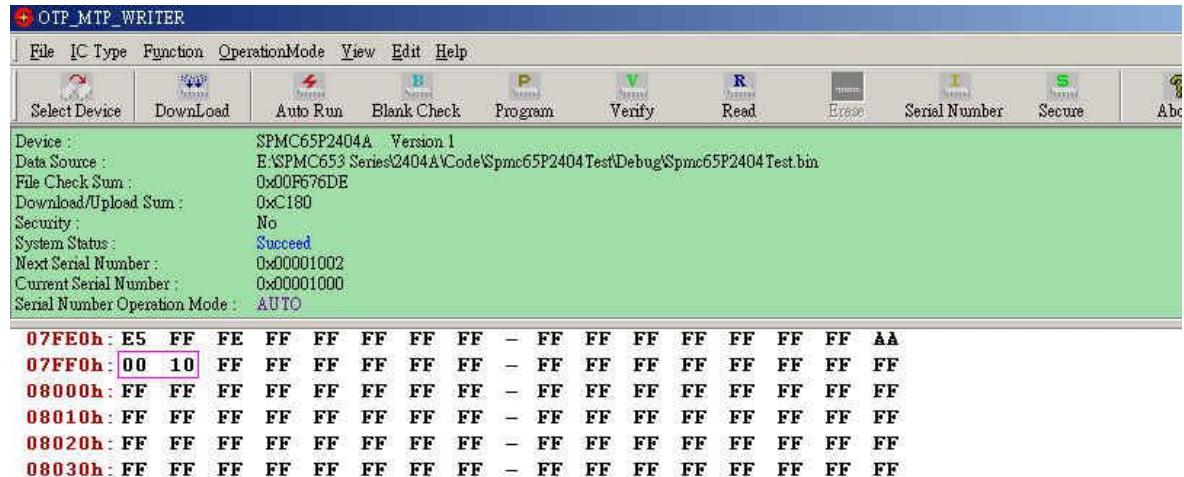
- (1) Start Address:\$7FF0
- (2) sizes: 2
- (3) Initial Number: \$1000
- (4) Increment: \$02
- (5) Mode: Auto

Then, the first serial number is \$1000 located at \$7FF0 and \$7FF1 (\$7FF0 is LSB). After setting up the operation information, the state is as follows. The serial number is still not available on the memory window now.



I Press "Program" icon

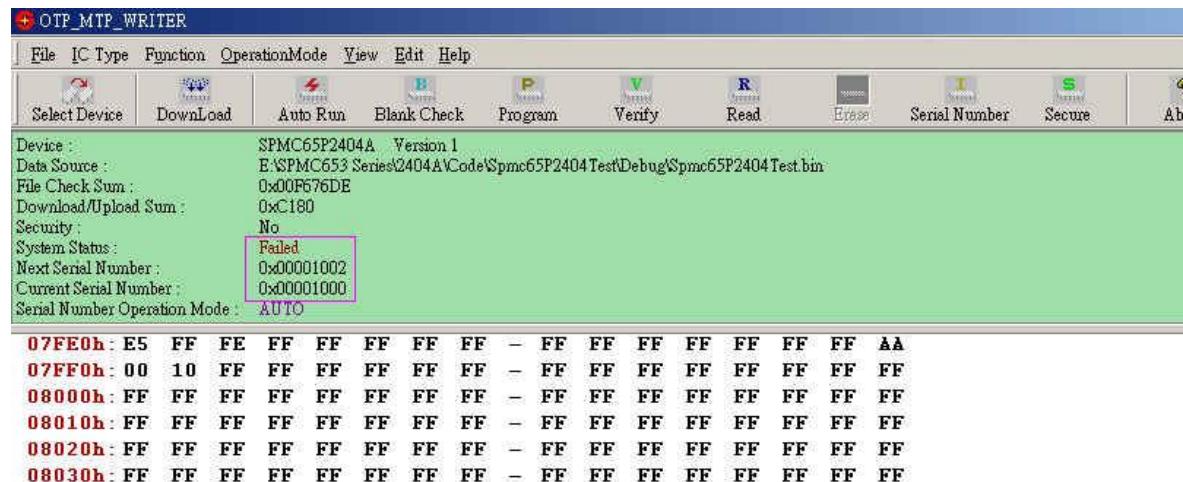
After finishing the programming flow, the “Current Serial Number” becomes \$1000, and the “Next Serial Number” becomes \$1002, meanwhile, the checksum is also recalculated. If the programming is successful, the serial number and checksum are accumulated continuously.



I Error happening

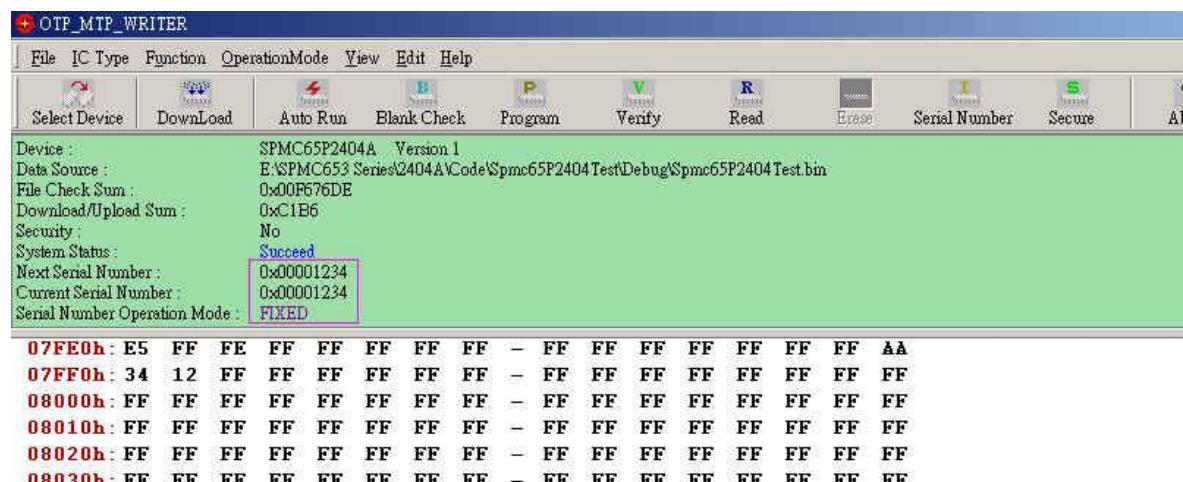
If there is an error while programming, Sunplus OTP_MTP_Writer will restore the system to previous operation state. For example, to continue last case, the system prepares to program

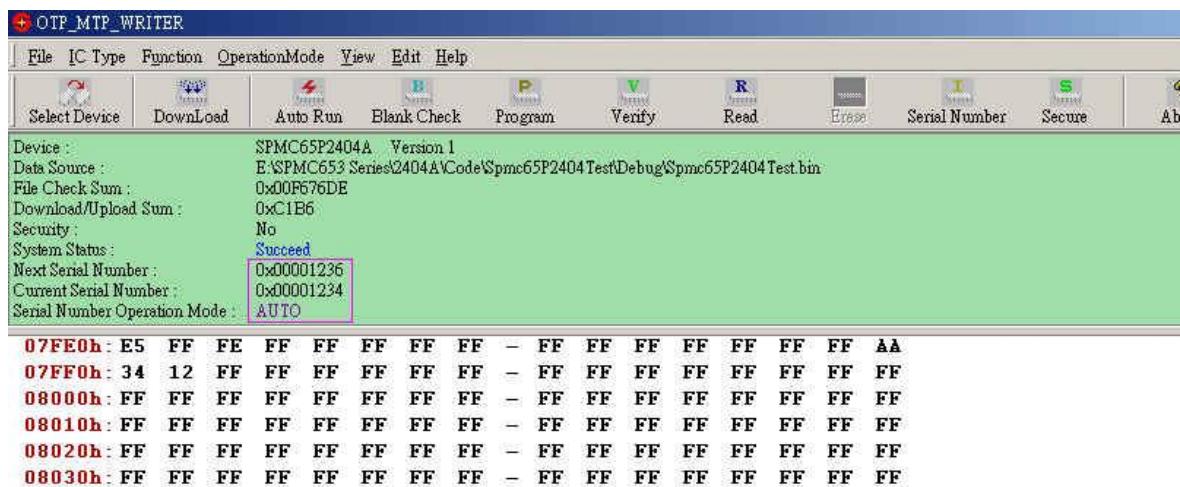
next OPT which next serial number is \$1002. But something is wrong and can not finish the programming action, then the message window shows "Failed" and the serial number setting keeps last value.



I Read data from some OTP

Sometimes, maybe user wants to read code from some OTP and then continues to program. If the operation mode is in fixed mode, then the next serial number is the same with current serial number; otherwise, the next serial number is the sum of read out value (recorded in current serial number) and increment in auto mode.





23.3.7 Important Notice

- | Fill \$FF in unused memory to reduce programming time.
- | To match the checksum calculated by third-party manufacture, Sunplus OTP_MTP_Writer changes the algorithm of checksum calculation of Download/Upload Sum. For SPMC65x family, the Download/Upload Sum is consisted of the contents of program memory, device configuration registers, and user information. The Download/Upload Sum of OTP_MTP_Writer is the same as the checksum calculated by Hilosystems programmer., and is also equal to the Download/Upload Sum of Q-Writer.
- | The file checksum of OTP_MTP_Writer is the same as FortisIDE.
- | No matter what actions to do, all progresses will be shown on the PC.
- | The performing time varies in different write modes. Once error is occurred, the next step will not be executed.
- | The erase and security functions are not applicable for all ICs.
- | The program can save the type of IC you used last time. Therefore, it will ask you whether to select the default IC type (the type you used last time) if you execute writer again. If default is selected, resource file will be downloaded automatically.
- | The program will save write mode you configured last time, and it will be loaded as default setting (configured last time) when executing the program next time.
- | The adapt-board can be removed from writer only when writer is in standby. Once adapt-board is replaced, a corresponding IC type must be selected again.
- | If the message window shows “Device: Not Selected”, it means IC type is incorrect. Select IC type again.
- | Once IC type is selected, the code file (*.bin, *.tsk) in RAM (located on board) will be cleared.
- | You must reselect the IC type again when you press the reset key (located on board).
- | If you only execute write function with write key on board, the progress will not be shown on PC, and write mode is not effective (for write mode only). However, you can disconnect COM-Port after downloading code in this situation.



SUNPLUS

SPMC65X Family PROGRAMMING GUIDE

- | After open or edit a file, data will be downloaded to writer-board automatically.

Note!

Sunplus ALL-Writer just programs the security address while enabling security and the device would not allow to read its content since. Be sure that the OTP has been programmed with source code and verified successfully before starting security operation.

23.4 Third-party programmer

To provide more selection of programmer, Sunplus has requested some third-party manufacturers to support the programming of SPMC65x. The new ALL-100 is the latest programmer of Hilosystems which can support a variety of device types, moreover, it is used as gang-writer. ALL-100 can support 8 pieces OTP programming at the same time. User can connect 8 ALL-100 at most through PC to control the 64 pieces OTP programming. The features of ALL-100 are listed as follows:



23.4.1 Introduction

ALL-100 supports device programming for a variety of device types covering EPROM, EEPROM, Serial PROM, Flash, PLD/CPLD/FPGA, MPU/MCU, NAND Flash, RS-MMC/MMC/SD Card etc. Most of programmable devices in DIP, SDIP, SOP, SSOP, TSOP, PLCC, QFP, or BGA package types can be programmed on a DIP48 Programming Module directly or through an appropriate ADAPTER/CONVERTER.

ALL-100 features high-tech pin drivers to provide excellent programming quality at high speed, high reliability and high expansion flexibility. All pin drivers built in ALL-100 are fully programmable to meet modern programming / testing needs, including newly released low voltage devices in the market.

A high speed USB port is provided to connect to the majority of notebook or desktop PC running under Windows 98 SE/ME/2000 SP4/XP SP1/Server 2003. Up to 8 sets of ALL-100 can be run simultaneously via tiered star USB to achieve the highest throughputs.

23.4.2 Feature & Benefits

1. Excellent Performance - Utilizing flexible pin drivers to get accurate waveforms, high programming speed plus over current protection, self-diagnostics, etc. ensure ALL-100's excellent performance.
2. Wide IC Coverage - Over 9,000 device types, covering EPROM, EEPROM, Serial PROM, Flash, PLD/CPLD/FPGA, MPU/MCU, NAND Flash, RS-MMC/MMC/SD Card etc. in package type of DIP, SDIP, SOP, SSOP, TSOP, PLCC, QFP, BGA, etc., can be programmed.
3. Inherit ADAPTER/CONVERTER from ALL-series - Over 500 ADAPTERs, 150 CONVERTERs can be used with

Single-Socket Programming Module to ensure your capital investment retained.

4. Ideal for Engineering as well as Production Environment - Wide device coverage, and flexible future expansion capability help users to achieve time-to-market challenge.
5. Multiple Sites Working Simultaneously - Via tiered star USB topology, up to 8 sets of ALL-100 can be connected to one PC to achieve the highest throughputs.
6. Easy to Use – to run the program under Windows. After master read or file download from PC, user only needs to select Blank check, Program, or Auto function from Menu, then hit YES key on programmer to start programming function. Operation is very easy.
7. Release New Device Support via Internet Weekly - New device support S/W at no charge.

23.4.3 Specifications

Device Support	
	Pin Count : from 8 pins up to over 300 pins Device Type: covering EPROM, EEPROM, Serial PROM, Flash, PLD/CPLD/FPGA, MPU/MCU, NAND Flash, RS-MMC/MMC/SD Card etc.
Device Contact	
	Default - DIP48, Textool. Others - Most of SOP, TSOP, PLCC, QFP, MLF, SDIP etc. through optional CONVERTERS or ADAPTERs. Some of SOP, TSOP, BGA, MLF, etc. through Single-Socket Module, and a few of BGA need extra TOP.
Max. Sockets in Parallel	
<input type="checkbox"/>	Up to 8 sockets on Optional Universal Gang Programming Module
Controller	
<input type="checkbox"/>	16 bits high-speed controller with big size FPGA & CPLD.
Interface Port	
<input type="checkbox"/>	1x USB port.
Data Transfer Rate	
<input type="checkbox"/>	USB 1.1 : 12 Mb/S USB 2.0 : 480 Mb/S
Max.Sites in Parallel	
<input type="checkbox"/>	Up to 8 sets of ALL-100 can work together through USB Hubs.
Functions	
<input type="checkbox"/>	Load file, Read Master, Program, Verify, Auto, ID Check, Checksum, Blank Check, Erase, Secure, Protect/Unprotect, Edit, Function Configuration, Self-Diagnostics.
Host Computer Requirements	
<input type="checkbox"/>	An Intel Pentium or compatible processor with 64MB of RAM. An Intel Pentium or compatible processor with 64MB of RAM. 20 MB free hard disk space under Windows 98 SE/ME/2000 SP4/XP SP1/Server 2003 operating system CD-ROM Driver.
Power	
<input type="checkbox"/>	AC voltage - 100 - 240 VAC. Frequency - 50 - 60 Hz Power consumption - 50W
Dimension	
<input type="checkbox"/>	L x W x H - 260mm x 150mm x 100mm.
Weight	

<input type="checkbox"/>	4 KG.
Operating Temperature	
<input type="checkbox"/>	0-40°C(32-105 °F)
Safety Certificate	
<input type="checkbox"/>	CE Approved.

23.5 Related Application Notes

This section lists some application notes and libraries about various usage of SPMC65X family. The application notes may help user develop quickly and offer the concepts about the usage of this topic. Sunplus also supports some powerful libraries about general operation, mathematics, communication, etc. The purpose of library is to help user speed up development. The current application notes and libraries related to In-Circuit Emulator are listed below.

Application Note:

Title	Application Note Series Number
No associated application notes at this time.	

Library:

Title	Application Note Series Number
SPMC65X series software basics manual	AN_O0100.DOC
SPMC65X series software advances 1 manual (data processing operation)	AN_O0101.DOC
SPMC65X series software advances 2 manual (mathematical operation)	AN_O0102.DOC

23.6 Revision History

Revision	Date	Remark
V0.1	03/31/2005	First edition



SPMC65X Family PROGRAMMING GUIDE

24 Assembly Tool

24.1 Description

The SUNPLUS 6502 assembly tools kit includes an assembler (xasm8.exe) to assemble program to encapsulate a binary data into a object file and a linker (xlink8.exe) to integrate multiple object/lib files into one single executable file. Plus, the library maker (xlib8.exe) makes programs or codes to be reused in various applications. Generally, most of SUNPLUS 8-bit micro-controller projects can be developed on the CPU6502 tools kits. We will introduce the usage for each tool in the later sections. Note that this manual only focuses on how to use these tools. For more information on how to write a CPU6502 program, instructions or addressing modes, please refer to chapter 25. All of these executed files, including xasm8.exe, xlink8.exe,xlib8.exe, are combined into Fortis IDE tool. In other words, user only used Fortis IDE tool could do several operations such as complier , linker and library maker.



Caution: The sunplus assembly tools can only be used with a Sunplus In-Circuit-Emulator (ICE).
In addition, these tools are applicable to Sunplus 8-bit IC only.

24.1.1 Assembler (xasm8.exe)

Xasm8 is the assembler for all SUNPLUS 8-bit microprocessors. The assembler intends to process SUNPLUS **65B02 Code** assembly source code into an object code.

Directive Command and Macro

Xasm8 involves plenty of directive/macro commands for assembly language programming.

Relocatable Object Code

Xasm8 can process assembly language source code into a re-locatable object code.

Conditional Assembly

You can make a section conditional assembly on the result of an assembly-time operation, and can nest conditionals to any level. The assembler checks for unbalanced conditions, and it indicates the current active condition level in the object code field to avoid nesting errors.

Listing File

The screen shows all errors during the listing to indicate the means and locations of errors.

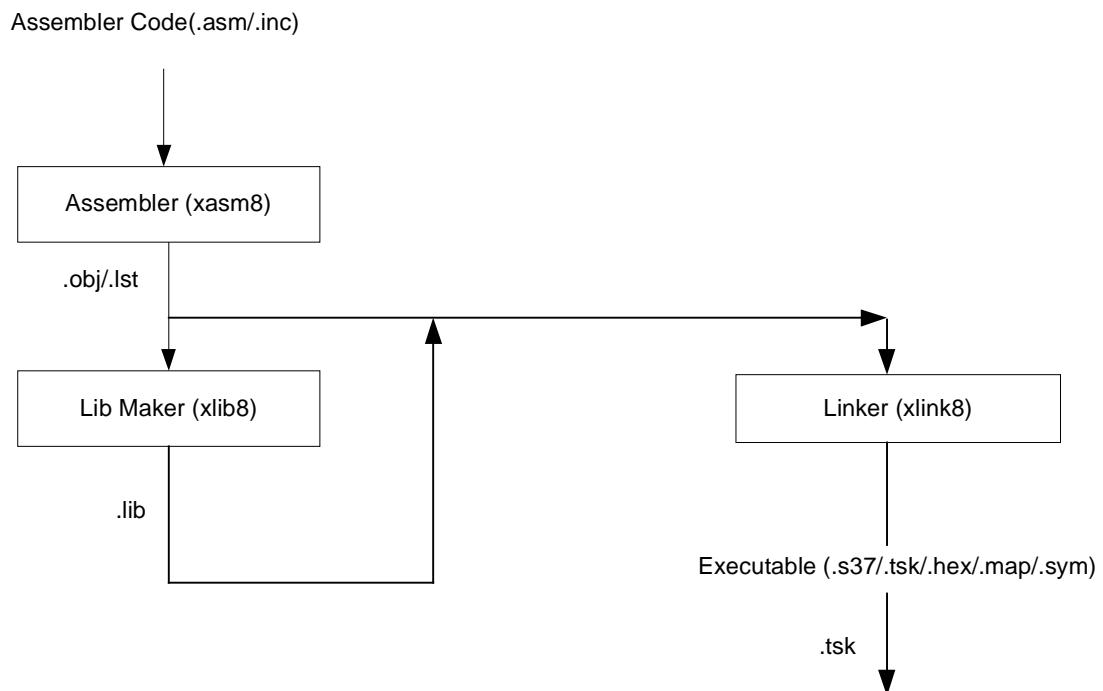
24.1.2 Linker (xlink8.exe)

The linker relocates the code to the execution addresses you specify, and creates the code file in the format you assign. You must always run the linker even if your program is assembled at the correct run address and there is no external reference.

24.1.3 Library Maker (xlib8.exe)

The library maker creates and manipulates libraries of object modules. The library can be reused since you can link a library module to any program created by the assembler

24.2 Coding Flow



24.3 Assembler

24.3.1 Filename extension

The following table shows the default file extensions if not specified.

Assembler	
.asm	Input to the assembler
.obj	Object file from assembler
.lst	Listing file

24.3.2 Operators

All calculations except exponentiation use integer arithmetic. Exponentiation uses an exponent. All comparisons return **1** if true and return **0** if false. The calculation operators are grouped in the following table.

Operator	Description
&&	Logical AND
!!	Logical NOT
	Logical OR
& or .AND.	Bit AND
~ or ! or .NOT.	Bit NOT
^ or .OR.	Bit OR
.XOR.	Bit XOR
+	specifies a positive operand.
-	specifies a negative operand.
**	Unsigned exponentiation
*	Unsigned multiplication
/	Unsigned division
+	Addition
-	Subtraction
>> or .SHR.	Shift the preceding expression right with 0 filled. The subsequent expression gives the number of shifts.
<< or .SHL.	Shift the preceding expression left with 0 filled. The subsequent expression gives the number of shifts.
.MOD.	Remainder

The following byte operators must start and end with a period, a space, or a tab.

Operator	Description
< or .LOW.	(Unary) keeps bits 0 – 7 of an address (Be necessary for re-locatable byte address values).

> or .HIGH.	(Unary) keeps bits 8 – 15 of an address (Be necessary for re-locatable byte address values).
.LOW16.	Keeps bits 0 – 15 of a 32-bit address (Be necessary for re-locatable byte address values).
.HIGH16.	Keeps bits 16 – 31 of a 32-bit address (Be necessary for re-locatable byte address values).
^ or .HIGH8.	(Unary) keeps bits 16 – 23 of a 24-bit address (Be necessary for re-locatable byte address values).

The following byte operators are used for comparisons.

Operator	Description
!= or .NE.	Not equal
> or .GT.	Greater than
< or .LT.	Less than
>= or .GE.	Greater equal
<= or .LE.	Less equal
.UGT.	Unsigned greater than
.ULT.	Unsigned less than

The priorities of the calculation operators are shown in the ascending order as follows. The operators in the same line have a same priority. You can change priorities by using parentheses.

'&&' or '||' or '!!'

'^' or .OR.

.XOR.

'&' or .AND.

.EQ., .NE.

.GE., .GT., .LE., .LT., .UGT., .ULT.

'>', .HIGH., '<', .LOW., .LOW16., .HIGH16., '^', .HIGH8.

'>>', .SHR., '<<', .SHL.

'+', '-' ;Addition, Subtraction

'*', '/' , .MOD.

'+', '-' ;(Unary) specifies a positive or negative operand.

'!' or .NOT., '~'

n High and Low Byte

To get the high byte of a 16-bit value, use .HIGH.. This allows bit8 through bit15 to be used as a re-locatable byte value. To get the low byte of a 16-bit value, use .LOW.. This also makes bit0 through bit7 to be used as a re-locatable byte value.

n Structure and Procedure**Ø Directive Introduction**

STRUCT

Group: Definition
Function: Defines the start of a struct
Syntax: label: .STRUCT
Notes: Begins a struct definition.
Example: test1: .STRUCT

ENDST

Group: Definition
Function: Defines the end of a struct
Syntax: .ENDST
Notes: Terminates a stuct definition.

PROC

Group: Definition
Function: Defines the start of a procedure
Syntax: label: .PROC
Notes: Begins a procedure definition.
Example: test1: .PROC

ENDP

Group: Definition
Function: Defines the end of a procedure
Syntax: .ENDP
Notes: Terminates a procedure definition.

Ø Struct Definition

Syntax:
Struct_name: .STRUCT
 Storage description
 .ENDST

For example:

test1: .STRUCT
 class: .DB 10
 name: .DW 'apple pie'
 count: .DD 01000h
 .ENDST

; In this example, a struct type 'test1' is defined,
; It contains three child fields 'class', 'name', and 'count'
; Each has initial vale 10, 'apple pie', 01000h

Ø Struct Variable Definition

Syntax:

Struct_variable: .struct_name [expression_list]
; The expression_list is used to store value to
; Child fields of struct_variable.

For example:

```
Stru_var1: test1 [7, 'cake',200h]
Stru_var2: test1 [6,,500h] ; Not store a value to the second child
field.
; So it keep the initial value.
```

Ø Struct Variable Reference

Syntax:

Struct_variable@chiled_field

For example:

```
LDA Stru_var1@class ; 'stru_var1' is struct variable defined
above and ; 'class' is a child field defined
above.
```

Ø Procedure Definition

Syntax:

```
Proc_name: .PROC
           Instruction_list
           RTS
           .ENDP
```

Ø Procedure Reference

Syntax:

```
CALL proc_name
```

For example:

```
CALL sub1 ; sub1 is a procedure defined above
```

n Sections

Ø Section Directive

SECTION

Function: Create a user-defined section

Syntax: *label:* .SECTION options

Notes: Create a user-defined section. The options are:

AUTO_STACK COMMON PAGE0 REF_ONLY

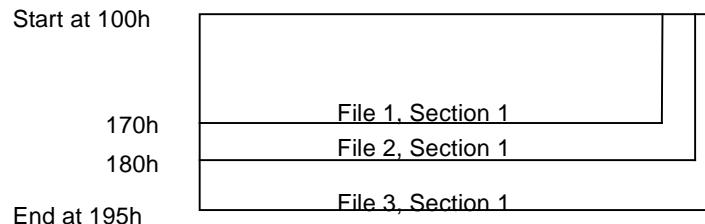
Example: MySec1: .SECTION COMMON
LDA #11

AUTO-STACK

Stacks each section in successive files of the link automatically on the top of the section of the same name in the preceding file.

COMMON

Links the section at the same address as the previous section of the same name. Since this option does not actually specify a relocation address, the linker prompts for an offset. It calculates the end address from load map information, adding the size of largest section to the start address of the first one.



In the above diagram, "file1, section 1" determines the start address, 100h. The section size is 70h, giving an end address of 170h. Overlaying "file3, section 1" with a size of 95h, extends the end addresses to 195h. Overlaying "file2, section 1" with a size of 80h, leaves the same end address since this section fits within the existing range. The first file that uses a COMMON section must not declare it as COMMON. COMMON sections must be reference only to avoid having multiple records with the same address (except RAM address).

PAGE0

The predefined PAGE0 section lets you define values in the range 00h to FFh. External symbols declared in a PAGE0 section (using PAGE0 as a modifier when declaring them) will have PAGE0 values.

```
.PAGE0
var1      .DB      2
.EXTERNAL    PAGE0 var2,PAGE0 var3
```

You can also use the PAGE0 modifier to create user-defined PAGE0 sections and switch to them using their names as directives.

```
Mysec1: .SECTION PAGE0
```

var4 .DB 3

REF_ONLY

Link the section for reference only.

```
MySec2: .SECTION ref_only
          INX
          LDA      #11
```

Ø Section Summary Table

The linker generates a section summary table as part of its load map file. This contains in condensed form the contents of the load map. The table lists the names of all the sections in the link, in the order of being created. For a section, the table contains only the name and the start and end addresses of each section that you actually use. The start address is the address at which the linker relocates the section. If section 1 in two files are located as follows:

For file 1: section1: from 100h to 180h

For file 2: section1: from 230h to 400h

The section summary table would display:

Section Name	Start	End
Section1	000100	000400

n Macro Examples

Ø Number Comparisons

```
Mac1:     .MACRO      arg1
          .IFNMA      1
          .EXIT       "Mac1 needs an argument"
          .MACEEXIT
          .ENDIF
          .IF         1=arg1
month:    .DB         1
          .MACEEXIT
          .ENDIF
          .IF2=arg1
month:    .DB         2
          .MACEEXIT
          .ENDIF
          .IF         3=arg1
month:    .DB         3
```

```
.MACEXIT  
.ENDIF  
.IF4=arg1  
month: .DB 4  
.MACEXIT  
.ENDIF  
.IF5=arg1  
month: .DB 5  
.MACEXIT  
.ENDIF  
.IF 6=arg1  
month: .DB 6  
.MACEXIT  
.ENDIF  
.ENDM
```

Ø String Comparisons

```
Mac2: .MACRO arg1  
.IFNMA 1  
.EXIT "Mac2 needs an argument"  
.MACEXIT  
.ENDIF  
.IFSAME "Monday",arg1  
day .BYTE 1  
.MACEXIT  
.ENDIF  
.IFSAME "Tuesday",arg1  
day .BYTE 2  
.MACEXIT  
.ENDIF  
.IFSAME "Wednesday",arg1  
day .BYTE 3  
.MACEXIT  
.ENDIF  
.IFSAME "Thursday",arg1  
.ENDIF  
day .BYTE 4  
.MACEXIT
```

```
.ENDIF
.IFSAME      "Friday",arg1
day          .BYTE       5
.MACEXIT
.ENDIF
.IFSAME      "Saturday",arg1
day          .BYTE       6
.MACEXIT
.ENDIF
.IFSAME      "Sunday",arg1
day          .BYTE       0
.MACEXIT
.ENDIF
.EXIT        "Argument error "
.ENDM
```

Ø Passing a label name

```
Mac2:      .MACRO      arg1
            .ASCII       arg1
            .ENDM
```

; Call the macro as follows:

```
mac2        label1
```

; The expanded macro reads as follows:

```
.ASCII      label1
```

Ø Passing argument into operand field

```
good:      .MACRO      arg1,arg2,arg3
name:      .DB         arg1
company:   .ASCII      arg2
quantity:  .DD         arg3
            .ENDM
```

; Call the macro as follows:

```
good        "apple pie", UCLA,17425
```

; The expanded macro reads as follows:

```
name:      .DB         "apple pie"
company:   .ASCII      UCLA
quantity:  .DD         17425
```

Ø Passing an argument into the label field

This lets you alter the program structure.

```
good:      .MACRO      arg1,arg2,arg3
arg1:      .DS          25
arg2:      .DS          20
arg3:      .DD          0
        .ENDM
```

; Call the macro as follows:

```
good      name, company, quantity
```

; The expanded macro reads as follows:

```
name:      .DS          25
company:   .DS          20
quantity:  .DD          0
```

Ø Argument with commas

Normally, the comma is the macro argument separator. You can use { } to substitute a delimiter character pair for the separator. This let you use commas as commas, passing into a macro an argument string containing a comma. Each argument must be enclosed by a delimiter pair. The only valid pairs are { }.

```
mac1:      .MACRO arg1,arg2,arg3
          .DB arg1
          .DB arg2 arg3
        .ENDM
```

;the macro call:

```
mac1      23h,{14h},{,62h}
```

;produces the result:

```
.DB 23h
.DB 14h,62h
```

Ø Recursive Macro

In this recursive macro, **arg1** (count) controls the number of recursions. During each recursion the macro reserves one byte and fills them with the values specified by **arg2**. Each successful execution decreases the count.

```
mac3:      .MACRO      arg1,arg2
count:     .VAR         arg1
          .IF          count = 0
          .MACEEXIT
```

```

.ENDIF
count:    .VAR           count - 1
          .DB            arg2
          mac3           count,arg2
.ENDM

; Calling the macro:
mac3      5,1ah

; Produces the result:
count:    .VAR           5
          .IF            count = 0
          .MACEXIT
          .ENDIF
count:    .VAR           count - 1 ;
          .DB            1ah
          mac3           count,0ah
          ...
count:    .VAR           count - 1
          .IF            count = 0
          .MACEXIT
          .ENDM
          ...
.ENDM

```

It is perfectly legal for a recursive macro to call another recursive macro and so on to any level you like. You need not to keep IF/ENDIF balance before exiting from the macro and the **xasm8** will do this automatically (the directive MACEXIT returns all conditionals to their original state).

24.3.3 Assembly Language

n Number Bases

The default number is based on decimal (10) system in the assembler. The following table illustrates the numerical bases available in assembler. Apply them in the suffixes of numbers.

Binary (2)	B or (% at prefixes)
Octal (8)	O or Q
Decimal (10)	D or non-indication
Hex (16)	H or (\$, 0X or 0x at prefixes)
ASCII String	Double or single quotes (e.g. "car" or 'car')

Value Definition

A numerical *value* directive argument is treated as the current overall number base (the default is

10). To assign a number with a different base, you must use a base suffix/ prefix.

n String Definition

A *string* argument must be enclosed in double/single apostrophes (" " or ' ') unless the directive syntax description is else specified.

n Spaces

In XASM8, you may leave spaces between operators, or an operator and its operand.

```
.IFTRUE A.AND..NOT. B  
or .IFTRUE A .AND. .NOT. B
```

n Labels

All labels are case sensitive. The length of a non-local label can have any number of characters & numbers, but the maximum length can be up to 32 only. All labels must start with an alphabetic character or underscore (_). Non-alphanumeric character cannot be used for labels except the underscore (_) or *question mark* (?). Macro name can start with an alphabetic character, underscore (_) or percentage (%). All label suffix could be alphanumeric character, underscore (_) or pound sign (#).

n Local Labels

Local labels are used as non-local labels, but the definition of a local label is valid only within its own “local area”, one bounded by labels, which keep their definition throughout the entire program. Because of this “local area only” referencing, you can reuse the local labels when a program passes from one local area to the next.

LABEL1:	or	LABEL1:
?x: INX		x?: INX
?y: JMP ?x		y?: JMP x?
JMP ?y ;		JMP y?

LABEL2:	or	LABEL2:
?x: INX		x?: INX
?y: JMP ?x		y?: JMP x?
JMP ?y		JMP y?

LABEL3:	or	LABEL3:
?x: INX		x?: INX
?y: JMP ?x		y?: JMP x?
JMP ?y		JMP y?

Note

I The assembler normally identifies a local label by a question mark (?) prefix or

suffix.

- | A local label must start with an alphabetic character or question mark (?).
- | Each local label has a different definition when referenced in a different local area.
- | The "?x" is not equivalent to "x?".
- | A local label can be up to 32 characters. Never use operators (e.g. +) in a local label. It is recommended to follow the non-local label rules for all label names.

n Case Sensitive

Assembler directives are not case sensitive. You can type them in lower, upper case or combination of both. However, all labels, including macro name, struct name, struct variable name, section name and procedure name are case-sensitive.

n Program Counter Sign

You can use the special characters, dollar sign (\$) or asterisk (*), in an expression to specify the program counter.

Example:

```
Smart_Branch: .macro Target
    .if (((+$2)-Target) <= 127)
        BEQ      Target
    .else
        BNE      $+5
        JMP      Target
    .endif
.endm
.Code
Cond1:
...
Cond2:
...
LDA Index
CMP #60
Smart_Branch Cond1
...
```

n External Symbol

Although you may declare multiple externals (see **EXTERNAL** directive), the assembler does not support math or any logical expression involving more than a single external.

This declaration is valid:

.EXTERNAL VAR1,VAR2,**PAGE0** VAR3,VAR4

This instruction is illegal:

LDA #(VAR1 .AND. VAR2 .OR. VAR4)

n Pre-defined Sections

The XASM8 has the pre-defined sections, **CODE** (the default), **DATA**, and **PAGE0**. If you use pre-defined sections, they are always linked first before any user-defined sections in the sequence of:

CODE, DATA, PAGE0

This sequence cannot be changed. For more flexibility in linking, use user-defined instead of pre-defined sections.

n User-Defined Sections

You can generate self-defined section names with the SECTION directive. Each section name can be up to 32 characters. You can have up to 4096 user-defined sections. The directive OPTIONS field controls the way of section to be linked. Options may be in any sequence, or in any letter case; commas are used to separate multiple options.

n Comments

The optional COMMENT field must begin with a semicolon (;) or the directive ".COMMENT".

24.3.4 Assembler Directives

Directives control the assembler behaviors, and it is distinguished from the assembly language instructions. An assembly directive, starts with a decimal point, can start anywhere in a program. A bracketed field or argument is optional. If an argument involves double brackets, the argument is optional, but its syntax requires the inner set of brackets. For instance, *[[variable1]]* is an optional argument, but it must be entered as *[variable1]* when applied.

n Setting Directives

ABSOLUTE

Function: Switches to non-re-locatable Mode

Syntax: *.ABSOLUTE*

Notes: Always assembles instructions in relative mode (assembler will generate relocatable object code). This instruction makes switching to non-relocatable mode.

Example:

```
.ABSOLUTE  
    test1:    .DS 1  
    test2:    .DS 1  
    test3:    .DB test1&test2
```

LINKLIST

Function: indicates the linker to relocate the assembler listing file.

Syntax: .LINKLIST

Notes: This directive must be used in conjunction with the SYMBOLS in order to relocate the listing file.

Example: .LINKLIST
.SYMBOLS
.CODE
LDA #17

RELATIVE

Function: Switches to re-locatable mode

Syntax: .RELATIVE

Notes: Switches from Absolute to Relative mode.

SYMBOLS

Function: Outputs symbol information to the object file

Syntax: .SYMBOLS

Notes: This directive must be used in conjunction with the LINKLIST to relocate the listing file.

Example: .LINKLIST
.SYMBOLS
.CODE
INX

n Definition Directives

ASK

Function: Prompts for a keyword response during assembly execution.

Syntax: [label:] .ASK *prompt*

Notes: Outputs a prompt to the terminal and waits for a 1-character response, from which it subtracts 30h, setting *label* equal to the result. The purpose of this is usually to introduce a 0/1 flag into a program. A carriage return is not necessary. On pass 2, outputs *prompt* and the response.

Example:

```
ver_num: .ASK AP code version for 2.0 (=1) or 1.0 (=0) ?:
```

CODE

Function: Switches to predefined CODE section

Syntax: .CODE

Notes: All storage variables and instructions following will be stored in this section. If no section is specified in the file, then CODE will be the default. The assembler should always be in relative mode when assembling executable instructions.

Example:

```
.CODE  
Reset:      LDA      #7AH
```

COMMENT

Function: Enables a multi-line comment

Syntax: .COMMENT char
String char

Notes: Lets you write blocks of comments without starting a semi-colon in every line. The assembler treats everything between the chars as a comment block. This is a useful temporary way to prevent code executing. The comment string can be as long as you prefer, but it must not start until the line after the directive and its initial character argument.

Example:

```
.COMMENT @  
...  
...      @
```

DATA

Function: Switches to predefined DATA section

Syntax: .DATA

Notes: All storage variables following will be stored in this section. The section should only contain storage variables and no instruction allowed.

Example:

```
.DATA  
Var1:      .DB      12H
```

DEFL / VAR / SET

Function: Equates a label to a value

Syntax: *label*: .DEFL *value*
label: .VAR *value*
label: .SET *value*

Notes: You can change the assignment at any point in the program. Do not use DEFL to redefine a label defined as a variable.

Example: Count: .VAR 10
.IF Count
 INX
 Count .VAR Count - 1
.ENDIF

END

Function: Defines the end of a program

Syntax: .END

Notes: Defines the end of a program or an included file.

Example: .END

ENDM / MACEND

Group: Definition

Function: Defines the end of a macro

Syntax: .ENDM
.MACEND

Notes: Terminates a macro definition.

Example: Mac1 .MACRO arg1
 LDA arg1
.ENDM

ENDP

Function: Defines the end of a procedure

Syntax: .ENDP

Notes: Terminates a procedure in which definition starts with the PROC directive.

Example:

```
test1:    .PROC
          LDA #IO_PORT1
          AND Mask_Value
          STA Setting
          RTS
          .ENDP
```

ENDST

Function: Defines the end of a struct

Syntax: .ENDST

Notes: Ends a struct definition

Example:

```
.PAGE0
Body1:   .STRUCT
          Index:   .DB      10
          name:    .DB      'Drive1'
          value:   .DD      0ffcH
Body1:   .ENDST
MySt1:   Body1[20,'Car',7dh]
.CODE
          LDA MySt1@Index
```

EQU / EQUAL

Function: Equates a label to a value

Syntax: *label*: .EQUAL *value*

label: .EQU *value*

Notes: Equates *label* to *value*. The *value* may be another symbol or an expression.

The value can be an expression formed by variable or immediate value. Thus, the variable in the expression must be declared in advance. Fail to declare such variable may result "illegal forward reference".

Example: label1: .EQUAL 3Ah
 label2: .EQU 20

EXTERN / EXTERNAL / XREF

Function: Declares a label that has already been defined in other files.

Syntax: .EXTERN *label[,label][,...]*
 .EXTERNAL *label[,label][,...]*
 .XREF *label[,label][,...]*

Notes: States that each *label* is defined in another file. A comma separates multiple labels. The assembler does not support any math or logic operation involving two or more externals.

Example: .EXTERNAL sn_lee, sn_ert
 .EXTERN var1, PAGE0 var2 ; var2 is in PAGE0 section

GLOBAL / PUBLIC / XDEF

Function: Declares a label that may be used in other files

Syntax: .GLOBAL *label[,label][,...]*
 .PUBLIC *label[,label][,...]*
 .XDEF *label[,label][,...]*

Notes: Defines each *label* as a global label for other files reference. A comma separates multiple *labels*. The linker will resolve all global and external references.

Example: .GLOBAL var1 ;Declares label var1 accessible to other files
 .GLOBAL var2,var3 ;Multiple declarations on the same line are legal
 ;when separated by a comma. The spaces are
 ;ignored

INCLUDE

Function: Includes this file in the assembly

Syntax: .INCLUDE *filename*
 .INCLUDE “*filename*” ;for long filename format

Notes: Instructs assembler to include the named file in the assembly. *Filename* may include a path. You must completely specify *filename* extensions. Each included file requires a separate included directive. You must use double apostrophes for long filename format.

Example: .INCLUDE src\lib\sub1.asm
.INCLUDE "C:\Program Files\Sunplus\Fortis IDE\Hardware.inc"

MACEXIT

Function: Exits from a macro
Syntax: .MACEXIT
Notes: Exit is immediate and unconditional. MACEXIT will restore all states before the macro invoked.
Example: mac1: .MACRO arg1,arg2
 .IFSAME arg1,arg2
 .MACEXIT
 .ENDIF
 .INY
 .ENDM

MACRO

Function: Defines the start of a macro
Syntax: *label*: .MACRO *args*
Notes: Begins a macro definition.
Example: mac1: .MACRO arg1,arg2,arg3

MESSAGE / MESSG

Function: Displays a screen message on pass 2
Syntax: .MESSAGE *use_message*
 .MESSG *use_message*
Notes: Outputs a user-defined message to the screen on pass2 only.
Example: Output: .MESSAGE Compiling AP code...

PAGE0

Function: Groups data for addresses 0h through OFFh
Syntax: .PAGE0
Notes: PAGE0 data is that which is defined in address range 0h through OFFh. PAGE0 may only be relocated within that address range. This will allow the assembler using 8 bit offsets and optimize error checking.

Example: .PAGE0

Var1: .DB 21h

PROC

Function: Defines the start of a procedure

Syntax: *label*: .PROC

Notes: Begins a procedure definition.

Example: Refer ENDP

SECTION

Function: Creates a user-defined section

Syntax: *label*: .SECTION options

Notes: Creates a user-defined section. The options are:

AUTO_STACK PAGE0 REF_ONLY COMMON

Example: MySec1: .SECTION COMMON

LDA #11

STRUCT

Function: Defines the start of a struct

Syntax: *label*: .STRUCT

Notes: Begins a struct definition.

Example: Refer ENDST

n Storage Directives

ASCII

Function: Stores an ASCII string

Syntax: [*label*:] .ASCII *string*

Notes: Stores each character of the string into memory.

Example:

test1: .ASCII John

test2: .ASCII "Mary"

BLKB

Function: Reserves bytes and stores values in each reserved byte.

Syntax: [label:] .BLKB bytes[,value]

Notes: Reserves a decimal number of bytes and stores the value in each.

Example:

Label1: .BLKB 17, 3AH ; Reserves 17 bytes and stores 3AH in each

Label2: .BLKB 7,0 ; Reserves 7 bytes with 0 in each

Label3: .BLKB 8 ; Reserves 8 bytes with 0 in each

BLKL

Function: Reserves long words and stores the values in each reserved long word.

Syntax: [label:] .BLKL words[,value]

Notes: Reserves a decimal number of 32-bit long words and stores the value in each.

Example:

Label1: .BLKL 17,43218765H ; Reserves 17 long words and
; stores 43218765H in each

Label2: .BLKL 7,0 ; Reserves 7 long words with zero in each

Label3: .BLKL 8 ; Reserves 8 long words with zero in each

BLKW

Function: Reserves words and stores the values in reserved words.

Syntax: [label:] .BLKW words[,value]

Notes: Reserves a decimal number of 16-bit words and stores the value in each.

Example:

Label1: .BLKW 17,4321H ; Reserves 17 words and stores 4321H in each

Label2: .BLKW 7,0 ; Reserves 7 words with zero in each

Label3: .BLKW 8 ; Reserves 8 words with zero in each

BYTE / DB / DEFB / STRING

Function: Stores a value in an 8-bit location

Syntax: [label:] .BYTE [value][,value][,...]

[label:] .DB [value][,value][,...]

[label:] .DEFB [value][,value][,...]

[label:] .STRING [value][,value][,...]

Notes: Stores *values* in consecutive memory locations. Use comma to separate

values (may be any mix of operand types). Bracket ASCII character strings with apostrophes (apply two apostrophes for an embedded apostrophe).

Example:

```

label1: .BYTE    0      ;Reserves 1 zeroed byte
label2: .DB       6      ;Reserves 1 byte ,storing 6
label3: .DEFB    5,2,8   ;Reserves 3 bytes, storing 5,2,8 in order
label4: .STRING   'Kitty' ;Stores the ASCII equivalent of the string "Kitty" in
                        ;consecutive locations
label5: .BYTE     'Kitty',0DH ;Same as before, with a carriage return at the end.
                        ;Space before operand is ignored, but a comma
                        ;is required
label6: .DB      'XASM"8' ;Embedded apostrophe
label7: .DB      "Xasm'8"

```

DD / LONG

Function: Stores a value in a 32-bit location

Syntax: [label:] .DD [[count]][value][,...]
 [label:] .LONG [[count]][value][,...]

Notes: Stores *value(s)* in 32-bit consecutive locations. Separates multiple *values* by commas. A *value* may be any numeric base, but is stored in hex.

Example:

```

label1: .DD        0A10H   ;Stored as 0A10H
label2: .LONG      'High'   ;Stored as 48H, 69H, 67H, 68H,and
                           ;stored in four 32-bit location
label3: .LONG      [5]31A2H ;Stores five 32-bit values with 31A2H

```

DEFS / DS

Function: Reserves a decimal number of bytes

Syntax: *label*: .DS *bytes*
label: .DEFS *bytes*

Notes: Stores value (\$FF) in the reserved bytes unless '-z' flag is enabled.

Example: label1: .DS 16 ;Reserves 16 bytes with \$FF
 label2: .DEFS 25 ;Reserves 25 bytes with \$FF

DEFW / DW / WORD

Function: Stores a value in a 16-bit location

Syntax: [label:] .DEFW [[count]][value][,...]

[label:] .DW [[count]][value][,...]

[label:] .WORD [[count]][value][,...]

Notes: Stores *values* in consecutive memory locations. [count] is the number of words reserved. Values are decimal, but are stored in hex. A comma separates multiple *values*, which may be any mix of operand types. Bracket ASCII character strings with apostrophes (using two apostrophes for an embedded apostrophe).

Example: label1: .WORD 0 ;Reserves 1 zeroed word
label2: .WORD 10 ;Reserves 1 word=10 decimal
label3: .DEFW 1,2,3 ;Reserves 3 words,=1,2,3 in that order
label4: .DW 'Hello', 0DH;Stores the ASCII equivalent of the
;string "Hello" in consecutive word
;addresses, and with a carriage return
;at the end spaces before operands
;is ignored but the comma is required
label5: .DW [4]31E2H ;Reserves 4 words, storing 31E2H to
;each word

DOUBLE

Function: Expresses a value as a double

Syntax: label: .DOUBLE value[,value][,...]

Notes: Convert *value(s)* to double-precision floating-point number expressed in IEEE format. A comma separates multiple *values*. Value must be defined by floating point.

Example:

label1: .DOUBLE 345.65 ; Stored as 66H,66H,66H,66H,66H,9AH,75H,40H
label2: .DOUBLE 70.0 ; Stored as 00H,00H,00H,00H,00H,80H,51H,40H

DUP

Function: Works with other storage instruction to store a value

Syntax1: [label:] .DB number DUP (value)

Notes 1: Reserves number of 8-bit byte and stores the value in each.

Syntax2: [label:] .DW number DUP (value)

Notes 2: Reserves number of 16-bit words and stores the value in each.

Syntax3: [label:] .FLOAT number DUP (value)

Notes 3: Reserves number of 32-bit floats and stores the value in each.

Syntax4: [label:] .LONG number DUP (value)

Notes 4: Reserved number of 32-bit long words and stores the value in each.

Syntax5: [label:] .DOUBLE number DUP (value)

Notes 5: Reserved number of 64-bit floats and stores the value in each.

Example:

.PAGE0

label1: .DB 20 DUP (0) // Reserves 20 zeroed bytes

label2: .DW 20 DUP (\$0FF) // Reserves 20 words,
// Storing 0FFh in each

label3: .DW 11Q DUP (20) // Reserves 9 words,
// Storing 20 in each

label4: .DW 11 DUP (\$20) // Reserves 11 words,
// Storing 20h in each

; The .FLOAT/.DOUBLE value is expressed in IEEE format

label5: .FLOAT 20 DUP (10.982) // Reserves 20 float and stores
// 10.982 in each.

label6: .DOUBLE 5 DUP (5223.29) // Reserves 5 double and stores
// 5223.29 in each

label7: .DD 20 DUP (0) // Reserves 20 zeroed long words.

label8: .DD 20 DUP (\$12345678) // Reserves 20 long words and
// stores \$12345678 in each

label9: .DW 5 DUP (?) // Reserves 5 zeroed words

label10: .DB 3 DUP(17,20,12) // Reserves 9 bytes

FLOAT

Function: Expresses a value as a float

Syntax: label: .FLOAT value[,value][,...]

Notes: Converts *value(s)* to single-precision floating-point number expressed in IEEE format. A comma separates multiple *values*. FLOAT truncates the float's fraction if over 24

bits.

Example: label1: .FLOAT 134.125 ;Stored as 00H,20H,06H,43H
label2: .FLOAT 102.0 ;Stored as 00H,00H,CCH,42H

ORG / ORGIN

Function: Sets the assembly address for a section

Syntax: .ORG *value*
.ORIGIN *value*

Notes: *value* is the assembly address for a section. It can be numeric or symbolic. If you add a numerical value to offset at link time, the relocated address will be offset and org.

Example: .CODE
.ORG 600H

n Conditional Directives

ELSE

Function: Assembles if previous condition is false.

Syntax: .ELSE

Notes: Defines the next statement to be assembled if a condition result is false.

Examples: .IF
...
.ELSE
...
.ENDIF

ENDC / ENDIF

Function: Defines the end of a conditional assembly block.

Syntax: .ENDIF
.ENDC

Notes: Defines the end of a conditional assembly block. Unmatched IF – ENDIF pairs generate an error message.

Example: .IF
 INX
.ENDIF

EXIT

Function: End assembly

Syntax: .EXIT *message*

Notes: Executed inside a conditional, EXIT terminates assembly, outputting the user-defined *message* of up to 500 characters. If the surrounding condition is true, EXIT executes. If it is false, assembly continues uninterrupted. Since EXIT stops assembly on Pass 1, there will be no listing file.

Example: Count: .DB 25H
.IFNDEF Count
.EXIT "Variable Not Defined"
.ENDIF

IF / IFN / IFNFALSE/ IFNZ / IFTRUE

Function: Assembles if this condition is true

Syntax: .IF *value*
.IFN *value*
.IFNFALSE *value*
.IFNZ *value*
.IFTRUE *value*

Notes: If *value* does not equal to zero, assembles subsequent statements. You can nest conditionals to any level. The *value* can be an arithmetic expression, another symbol, or a string.

Example: .IF label1
...
.ENDIF

IFABS / IFNREL

Function: Assembles if this label is absolute.

Syntax: .IFABS *label*
.IFNREL *label*

Notes: Searches the symbol table for the *label*. If the *label* is absolute, assembles subsequent statements. If the label is relocatable, ignores everything before the next ELSE

or ENDIF. If the *label* cannot be found, generates an error message.

```
Example: .IFABS      label1  
          ...  
          .ENDIF
```

IFCLEAR

Function: Keeps conditional directive pairs balanced in a recursive macro

Syntax: .IFCLEAR

Notes: In a recursive macro, IFCLEAR keeps conditional directive (ex. IF-ENDIF) pairs balanced, it will keeps program flow under full IF checking control. Since some kind of IF always do a macro exit, IFCLEAR could be used when a macro contains MACEXIT for early exit.

Example:

```
%DeclareTimers .MACRO Count  
    .IFZ      Count  
        .IFCLEAR  
        .MACEXIT  
        .ENDIF  
    .IFDEF D_Timer|<Order>  
        %RAM_ALLOC R_Timer|<Order>,1  
    .ENDIF  
    Order: .VAR      Order+1  
    %DeclareTimers Count-1  
.ENDM
```

IFDEF

Function: Assembles if this label has been defined

Syntax: .IFDEF *label*

Notes: Search the symbol table. Assembles subsequent statements if *label* is defined. If not, ignore before the next ELSE or ENDIF.

```
Example: var1:   .DEFL     14H  
          .IFDEF     var1  
          INX  
          .ENDIF
```

IFDIFF / IFNSAME

Function: Assembles if two strings are not identical

Syntax: .IFDIFF *string1,string2*
 .IFNSAME *string1,string2*

Notes: Compares the two strings. If not matched, assembles subsequent statements. If they match, ignores everything before the next ELSE or ENDIF. Strings come in two kinds: with spaces and without. Enclose a “spaces” string in apostrophes, and enter an embedded apostrophe as two apostrophes. “Non-space” string does not need apostrophes. You can only compare same type of strings, and must separate them with a comma. IFFDIFF is useful for comparing macro parameter arguments.

Example: .IFNSAME 'my str1','my str1'
 .IFFDIFF 'Xasm"8','Xasm"8'
 .IFFDIFF "Kitty","Kitty"
 .IFFDIFF arg1,"January"

IFE / IFFALSE / IFNTRUE / IFZ

Function: Assembles if this condition is not true

Syntax: .IFE *value*
 .IFFALSE *value*
 .IFNTRUE *value*
 .IFZ *value*

Notes: If the condition is true, ignores everything before the next ELSE or ENDIF.

Value can be an arithmetic expression, another symbol, or a string.

Example: .IFZ *label1*
 ...
 .ENDIF

IFEXT

Function: Assembles if this label is an external

Syntax: .IFEXT *label*

Notes: Searches the symbol table for *label*. Assembles subsequent statements if the label is external. If not, ignore before the next ELSE or ENDIF. If the *label* cannot be found, it generates an error message.

Example: .IFEXT label1

IFMA

Function: Assembles if this macro argument exists

Syntax: .IFMA *argument#*

Notes: Used inside a macro. It scans the macro call line for the argument number is specified. If it finds the argument, it assembles subsequent statements. If not, it ignores everything up to the next ELSE or ENDIF. IFMA will not find an argument if *argument#* is 0, but it will assemble subsequent statements) if the call line contains none.

Example: .IFMA 2
ECO_Flag Var 1
.ENDIF

IFNABS / IFREL

Function: Assembles if this label is relocatable

Syntax: .IFNABS *label*
.IFREL *label*

Notes: Searches the symbol table for the *label*. If the *label* is relocatable, assembles subsequent statements. If the label is absolute, ignores everything till the next ELSE or ENDIF. If the *label* cannot be found, it generates an error message.

Example: .IFNABS label1

IFNDEF

Function: Assembles if this label is not already defined

Syntax: .IFNDEF *label*

Notes: Searches the symbol table. It assembles subsequent statements if the label is not defined. Otherwise, it ignores before the next ELSE or ENDIF.

Example: .IFNDEF label1

IFNDIFF / IFSAME

Function: Assemble if these two strings are identical

Syntax: .IFNDIFF *string1,string2*

.IFSAME string1,string2

Notes: Compares two strings. If matched, assembles subsequent statements. If not matched, ignores everything before the next ELSE or ENDIF. Strings come in two forms, with spaces and without space. Enclose a “space” string in apostrophes, and enter an embedded apostrophe as two apostrophes. “Non-space” string does not need apostrophes. You can only compare the same type of strings, and must separate them with a comma. IFSAME is useful for comparing macro parameter arguments.

Example: .IFSAME 'my str1','my str1'

```
...
.ENDIF
.IFSAME 'Xasm"8','Xasm"8'
...
.ENDIF
.IFSAME "Teach","Teach"
...
.ENDIF
.IFNDIFF arg1,"January"
...
.ENDIF
```

IFNEXT

Function: Assembles if this label is not an external.

Syntax: .IFNEXT *label*

Notes: Searches the symbol table for *label*. If the *label* is not external, assembles subsequent statements. If the *label* is external, it ignores everything before the next ELSE or ENDIF. If the *label* cannot be found, it generates an error message.

Example: .IFNEXT var1

IFNMA

Function: Assembles if this macro argument does not exist

Syntax: .IFNMA *argument#*

Notes: Used inside a macro; checks the macro call line for the specified argument number. If it does not find the argument, it assembles subsequent statements up to an ELSE or ENDIF. Otherwise, it ignores everything up to the next ELSE or ENDIF. If

argument# is 0, IFNMA will detect any argument, and if there is at least one, it will assemble subsequent statements.

Example: .IFNMA 2

IFNPAGE0

Function: Assembles if this label is not a PAGE0 variable

Syntax: .IFNPAGE0 *label*

Notes: Searches the symbol table for the *label*, and if the *label* is not defined as a PAGE0 variable, assembles subsequent statements. If the *label* is a PAGE0 variable, ignores everything before the next ELSE or ENDIF. If the *label* cannot be found, generates an error message.

Example: .IFNPAGE0 label1

IFPAGE0

Function: Assembles if this label is a PAGE0 variable

Syntax: .IFPAGE0 *label*

Notes: Searches the symbol table for the *label*, and if the *label* is defined as a PAGE0 variable, it assembles subsequent statements. If the *label* is not a PAGE0 variable, it ignores everything before the next ELSE or ENDIF. If the *label* cannot be found, it generates an error message.

Example: .IFPAGE0 label1

IFSSEQ

Function: Assembles if the first string is a sub-string of the second

Syntax: .IFSSEQ *n,string1,string2*

Notes: Compares the first strings with the sub-string starting at the *n*th character of the second string. If the string matches, assembles subsequent statements. If the string do not match, it ignores everything before the next ELSE or ENDIF.

n is a decimal integer from 1 to the number of characters in the second string.

Example: .IFSSEQ 4,'car','The car is mine'

IFSSNEQ

Function: Assembles if the first string is not a sub-string of the second

Syntax: .IFSSNEQ n,string1,string2

Notes: Compares the first strings with the sub-string starting at the nth character of the second string. If the string not matched, it assembles subsequent statements. If the string matched, it ignores everything before the next ELSE or ENDIF.

n is a decimal integer from 1 to the number of characters in the second string.

Example: .IFSSNEQ 4,'car','The car is mine'

24.3.5 Macros

n Definition

A macro is a sequence of source lines to be substituted for a single source line. You must define a macro before you can use it. On pass 1, the assembler stores the definition and, when it reaches the macro name, substitutes the defined source lines. A macro definition may include arguments, substituted into any field except the comment field. Dummy arguments may not contain spaces. The start of a macro must be defined by a MACRO directive. The macro name goes in the label field. The macro must end with an ENDM (MACEND) directive.

n Calling a Macro

In a macro call, arguments may be any of following types: direct, indirect, character string or register. Only an ASCII string bracketed by apostrophes may include spaces (Apostrophes in the string must appear as double apostrophes). As the dummy argument names are identical, arguments can be passed to nested macros. Memory space is the only limitation on macro nesting. Arguments must be separated by commas. Leading spaces and tabs are ignored. A single comma acts as a place-holder for a missing argument.

n Argument Separators

In a macro body, valid argument separators are:

, { }

Example:

Math_Add: .MACRO var1,var2,sum

LDA var1

CLC

ADC var2

STA sum

.ENDM

; Calling the macro:

```
Math_Add S1,S2,S3
Math_Add S1,{Speech_Table,X},S3
Math_Add S1,{Speech_Table,Y},S3
; Produces the following result:
Math_Add S1,S2,S3
    LDA S1
    CLC
    ADC S2
    STA S3
Math_Add S1,{Speech_Table,X},S3
    LDA S1
    CLC
    ADC Speech_Table,X
    STA S3
Math_Add S1,{Speech_Table,Y},S3
    LDA S1
    CLC
    ADC Speech_Table,Y
    STA S3
```

n Implicit Labels

Macro definitions can contain explicit (user-defined) or implicit (auto-defined) labels. The assembler will not alter an **explicit** label. Adding a suffix (#) to a label makes it **implicit**, and informs the assembler to automatically substitute a digital following an underscore (_) and 6-digit expansion number for the #. The label and its expansion may not exceed 32 characters.

```
mac1:      .MACRO      arg1,arg2,arg3
          arg1
var#:      .DB         arg2
wed#:      .DB         arg3
          .ENDM
```

; Calling the macro:

```
mac1      RTS,19,"House"
```

; Produces the following result:

```
RTS
var1_1568: .DB      19
wed1_1568: .DB      "House"
```

n String Concatenation

The broken bar character '|'(hex 7C) is the string concatenation operator. You can concatenate only inside a macro.

n Value Concatenation

You can concatenate a string and a value with the broken bar character followed immediately (with no intervening space) by a variable enclosed in angle brackets.

```
mac2      .MACRO  arg
value:    .VAR     value+1
arg|<value>.EQU    1
.ENDM
```

Initializes value before calling the macro to avoid label having different value on pass 1 and 2.

```
value:    .VAR     0
```

Now call the macro:

```
mac2      label
```

The expanded macro reads as follows:

```
label1:    .EQU    1
```

24.4 LINKER

24.4.1 General Description

The function of linker is to cohere assembly programs, files, modules, and sections to an integrated program file. It resolves external reference, relocates addresses and modifies listings to show run-time addresses and final op-codes. The linker, generates the most commonly used file formats, can be executed in RAM with most output file formats. It allows linking files with any size as long as memory is available.

24.4.2 Filename Extensions

The following table shows the default filename extensions if they are not specified.

.obj	Input file to linker
.lib	Library
.tsk	Executable Object Code
.s37	Motorola s37
.hex	Intex Hex / Extended Intel Hex
.map	Mapping File
.sym	Symbol Table (Microtek/AD High Level)

24.4.3 Load/Run-Time Address

You can specify a section link address at link time, but you can only define a section as reference. Therefore, although the linker includes its link information, the section does not appear in the output file. A section can be linked indirectly with different load and run-time addresses, making ROM code to be moved to RAM at run time.

24.4.4 Search Order

The search order for finding library global symbol is as follows:

1. Input Files
2. Library Files

24.4.5 Address Relocation

In relocating addresses, the linker adds the offset to the addresses generated by the assembler. If using the ORG directive in your assembly file, the relocated address is org. The assembler keeps an attribute table associated with each symbol used in the program.

1. A label preceding an instruction is re-locatable.
2. If a label is defined with an EQU directive, its operand type determines whether it is re-locatable. If the argument contains only one re-locatable token, the label is re-locatable. If it does not contain any, the label is not re-locatable. It must not contain more than one token. If it does, the assembler generates an error message.

SIZE: .EQU 10h ; absolute define
LDA SIZE ; mean load 10h to A register.

Label:

LDA ..
NOP
SIZE: .EQU Lable+10 ; The Label is assigned value in linker stage, re-locatable define
LDA SIZE ; SIZE is unknown in assembler stage,

3. If a label is defined under absolute mode, it will not be re-locatable.

24.4.6 Symbol Table Output Formats

The linker can output the Microtek or ADHighLevel symbol table and code file with s37, tsk and Extended Intel hex formats.

24.4.7 Linker Features

You can invoke the linker in interactive, command Line, and linker script file. The load map, an alphabetic global symbol cross-reference list and all link errors can be saved on disk. In Linker Script File mode, you may link each of your files in one of the following ways.

Auto-stacking sections

Link each section of each file in a link automatically on top of the previous section of the same name.

Indirect linking

Useful if your program resides in ROM, but the data has initialized values that will be changed as the program runs. You need a start-up routine to move it from ROM to RAM. It stacks the data normally in ROM, but links it to a run-time address.

Offsetting sections

Link a section at a specific address. If using the ORG directive in your assembly file, the relocated address is org address.

Stacking sections

Link, as a single unit, all sections of the same name from different files.

In all modes, <Ctrl>C terminates the linker and returns to the operating system.

24.4.8 Interactive mode

To run the linker, type: **xlink8<cr>**.

Input

You will see the prompt **Input Filename:**

Type the input filename and press <cr> (The default extension is **.obj**). The linker opens the file, then prompts another **Input Filename:**

If there is no more input file, enter <cr>.

Output

You will see the prompt **Output Filename:**

Enter the output filename and press **<cr>**.

If you type **<cr>** only, the linker gives the output file with the same name as the first input file. The extension depends on the output format you select

Libraries

The next prompt is **Library Filename:**

Enter the library filename and press **<cr>**. You can omit an extension since the linker automatically searches files with extension **.lib**. Hit **<cr>** if no more library files.

Options

The linker always prompts for options.

Options (D, C, M, N, X, 3, E, H, Z, I <CR>= Default):

Enter the options and press **<cr>**. The options let you set load map destination and code, and symbol file formats (if “Z” is selected, tsk file gap will be filled by \$0).

Offsets

You will see the following prompts:

Enter Hex ROM Offset For “(section name)” in “(input file name)”:

Enter the offset, and press **<cr>**.

Enter Hex RAM Offset For “(section name)”:

To make the prompted section loading address is the same as its running address, press **<cr>**. If the input files and library files have multiple sections, the linker will prompt for each section’s offset.

24.4.9 Interactive Mode Options

The option field creates a load map, specifies its destination, and selects output file and symbol table formats.

The option prompt is:

Options (D, C, M, N, X, 3, E, H, Z, I <CR>= Default): (See the following table for the interactive modes options)

Options are not case-sensitive. The default settings are:

Load Map	none
Symbol & Code Files	processor's defaults

- | One Load Map destination (**D**).
- | One Symbol Table format (**C** or **M** or **N**).
- | One Code File format (**X** or **3** or **E** or **H**).

If you enter more than one option in a category, only the last one is effective.

Interactive Modes Options Table	
Option	Description
-D	Creates a file containing all link errors, a global symbol table and the load map. This file has the same name as the linker output file with the extension of .map .
-C	Creates an AD High Level Symbol Table.
-M	Creates a Microtek Symbol Table with 16 bits address.
-N	Creates a Microtek Symbol Table with 24 bits address.
-X	Generates a pure binary executable output file.
-3	Generates a Motorola s37 output file.
-E	Generates a extend Intel hex.
-H	Generates a Intel hex
-Z	Tsk file gap will be filled by \$0
-I	Generates a 64K-byte pure binary executable output file.
<cr>	Generates the processor's default output format.

24.4.10 Command Line Mode

You can invoke the linker from the command line. The command line format is as follows (bracketed arguments are options):

`Xlink8 -c file1 [-loffset] [file2 [...]] [-ofile] [-Lfile] [...] [-options]`

-c

Precedes the first input filename. It sets the linker to command line mode.

file1

You must specify at least one input filename.

-I

(Lower case L) precedes an optional offset for each section in the input file (i.e. if there are 12 sections, there should be an -I entry for each section that you want to specify an offset). You can prefix an offset with the (-) operator. “**-I-nnnn**” is a valid offset.

file2

Second input file. You may link as many files as you like with the same syntax for filenames and section offsets as for the first input file.

-o

Precedes an output filename. If you omit an output filename, the linker will create a default output file with the same name as the first input file. The extension depends on the output file format.

-L

(Upper case L) introduces the name of each library file to be included in the link. You do not need to specify an extension since the linker looks automatically for files with the extension .lib.

-

The minus sign (-) prefixes the options list (There must be no separating spaces.). If the command line omits options, the linker will prompt for them.

24.4.11 Linker Script File Mode

Linker Script File mode is a powerful way to operate the linker. A set of keywords provides greater flexibility than the interactive modes. You can link directly or indirectly to addresses of your own choice: define an address for a section, specify options in any order and link sections in almost any order (You cannot locate a position of a section after another is not yet located).

n Command

Keywords are not case sensitive.

Command	Modifiers
Version:	[Pseudo-parameter]
[Options:]	tsk, map, adhighlevel, microtek, microtek24, m37, ehex, hex, zero, img
Input: or file:	Filename [, filename...]
Output:	Filename
Library:	Filename [, filename...]
Locate:	after section linkafter section stack reference at address linkat address common

Version:

This must be the first keyword in a file.

Function: Informs linker to use Linker Script file mode.

Syntax: version: [pseudo-parameter]

Example: version: 1.1.5 dated 06/26/01

Comments may precede or follow version on the same or separated lines, but it must be the first keyword in the file. Version is insensitive to trailing characters on the same line. You can add a version number, date, or other information as a pseudo-parameter.

Options

Function: Controls the creation and destination of a load map, the formats of output symbol and code files.

Syntax: options: option [, option...]

Example: options: tsk, map

The following table describes the option parameters in detail.

Link Options Parameters	
Load Map	
Map	Creates .map disk file containing link errors, section summary, and load map.
Symbol Table	
Adhighlevel	AD High Level Symbol Table
Microtek	Microtek Symbol Table with 16 bits address.
Microtek24	Microtek Symbol Table with 24 bits address.
Code File	
Tsk	Executable (pure binary)
M37	Motorola s37
Ehex	Extended Intex Hex
Hex	Intel Hex
Zero	Tsk file gap will be filled by \$0 (must work with option Tsk)
Img	Generates a 64K-byte pure binary executable output file.

Input or File

Function: Names the input file(s) to be linked. The default object extension is .obj, but you can specify else.

Syntax: **Input:** "filename"[,"filename"]...

Examples: Input: "mytest", "yourtest"

File: "mytest.obj", "yourtest.obj"

Output

Function: Names the output file. The default extension is the processor's default code file format extension, but you can specify else.

Syntax: **output:** "filename"

Examples: output: "ourtest.tsk"

Library

Function: Names the library file(s). The default extension is .lib.

Syntax: **library:** "libfile"[,"libfile"]...

Examples: library: "Cmacro.lib", "PrintfI.lib"

Locate

Function: Locates a section in object (module) of library at (linkat) an address or after (linkafter) a section already located.

Syntax: **locate:** section at address, after section name

[linkat address, linkafter section name]

[stack, common, reference]

Examples: locate : section1 at 1234h

locate : section2 after section1

locate : section3 at 8000

locate : section4 at 9000

locate : section5 at 1000h linkafter section2 stack

locate : section6 after section4

Notes: One of **at** or **after** can be used with one of **linkat** or **linkafter**.

* In all modes, the default of address base is **hexadecimal**, and that cannot be changed.

24.4.12 Offsets

In all modes, the linker requires an offset for each section of each file in a link. The load address is the offset (The examples assume no ORG directives). If you omit an individual section's offset, the linker assumes 0000H, and loads the section at 0000H. If you omit ALL offsets for the first file, the linker loads the first section at 0000H, and stacks the rest with each immediately on top of the previous one.

SectionD
SectionC
SectionB
SectionA

↳ 0000H

If you omit ALL offsets for the second or later file, the linker **auto-stacks** each subsequent section on top of the previously loaded section of the same name.

File2/SectionD
File1/SectionD
File2/SectionC
File1/SectionC
File2/SectionB
File1/SectionA

↳ 0000H

24.4.13 Indirect Linking

The data stored in ROM cannot be modified unless the data is moved into RAM. Indirect linking achieves this by stacking the data normally at a load address in ROM, but linking it to a run-time address in RAM.

■ Direct Linking

Your program resides in ROM. The data consists entirely of lookup tables or constants, intend to be read only, and never to be modified.

n Indirect Linking

The same program resides in ROM, but has initialized data whose contents will change as the program runs. You need to move the data from its load address in ROM to a run-time address in RAM. Link indirectly. The table shows the differences in run-time link addresses.

Direct		Indirect	
Load	Run-Time	Load	Run-Time
(ROM)	(ROM)	(ROM)	(ROM)
CODE DATA	CODE DATA	CODE DATA	CODE
(RAM)	(RAM)	(RAM)	DATA (RAM)

Start .Section .CODE

Label1:

LDA #10

In link script file:

Locate Start at \$800 linkat\$70 ; indirect linking

LDA Label1 ; A **B** \$70

; Mean Start section BIN code is stored at \$800, but program access is at \$70.

24.4.14 Linker Script File Mode

1. At least, the first section in a link must be linked directly; the linker has a fixed address from which to calculate the indirect addresses.
2. Indirect sections are stacked in the same order as if they are direct.
3. You can indirectly link a section by using the keywords, **linkat** and **linkafter**.

24.4.15 Symbol Table

Two choices are:

1. AD High Level
2. MicroTek

24.4.16 Executable Code File Format

n Image File

Linker Option: **X**

Default Output File Extension: **.tsk**

An executable code file is a pure binary file of opcodes and operands in which the default starting address is 0000H. Since the linker fills gaps between the end of each section and file, and the start address of the next with OFFh (unless “zero” flag been enabled in TSK format). You could use the ORG directive to set by arbitrary order addresses for successive sections and files.

n Motorola S37

Linker Option: **3**

Default Output File Extension: **.s37**

n Intel Hex

Linker Option: **H**

Default Output File Extension: **.hex**

24.4.17 Map File

n Memory Map Summary

The linker will create a map file (i.e. *filename.map*) if option “D” is chosen in command line and interactive modes or option “map” is chosen in Linker Script file mode. A map file can contain symbols defined in every file of the link. Cross-references may also be included in the map file. The SECTION SUMMARY shows all sections that are contained in the object files along with their load / run time addresses. The

SECTION SUMMARY load / run time addresses will differ if linking a section indirectly.

The linker can relocate listing files. After the linker relocates a listing file, symbol addresses/values are fixed, and thus showing run time addresses/values. Here is a typical map file followed by a description of all its elements.

Wednesday, June 20, 10:16:55, 2001

Sunplus 6502 Linker - Version 1.1.5

Global Symbol Name	Global Value	Global Filename
--------------------	--------------	-----------------

St2	B	Test2.obj
St1	100	Test2.obj
St3	101	Test2.obj
FSub1	72E	Test1.obj
FSub2	737	Test1.obj
FSt1	D	Test1.obj
F_DIV	742	DIV
F_MUL	74E	MUL
F_ADD	753	ADD

* **SECTION SUMMARY** *

* Section Name	Start	End	Description	*
* PAGE0	00000000	00000011		*
* DATA	00000100	00000124		*
* CODE	00000600	00000759		*
* MySec1	00000000	0000000D		*
* MySec2	0000000E	0000001B		*

* **LOAD MAP** *

* Section Name	Starting Address	Ending Address	Size	*
* Test2.obj				*
* PAGE0	00000000	0000000C	0000000D	*
* DATA	00000100	0000011C	0000001D	*
* CODE	00000600	0000072D	0000012E	*
* MySec1	00000000	0000000D	0000000E	*
* Test1.obj				*

```

*   CODE           0000072E    00000741    00000014    *
*   DATA           0000011D    00000124    00000008    *
*   PAGE0         0000000D    0000000F    00000003    *
*   MySec2        0000000E    0000001B    0000000E    *
* DIV
*   CODE           00000742    0000074D    0000000C    *
*   PAGE0         00000010    00000011    00000002    *
* MUL
*   CODE           0000074E    00000752    00000005    *
* ADD
*   CODE           00000753    00000759    00000007    *
*****

```

Linker Output Filename : Test2.tsk
 Disk Mapping Filename : Test2.map
 Symbol Table FileName : Test2.sym
 Format : Microtek

Linker Errors : 0
 Output Format : tsk

24.5 LIBRARY MAKER

24.6 ERROR MESSAGE

24.6.1 Assembler Errors

This list includes the error messages for XASM8 Assemblers.

A0000: syntax error

This occurs when an instruction or expression is not matched to the XASM8 instruction format.

A0001: endm expected before end of file

In the source code file, the numbers of MACRO and ENDM are not balanced. It needs another ENDM to keep balance.

A0002: unmatched '{"

A0003: symbol already defined

A symbol that has already been defined cannot be redefined.

A0004: macro name used in expression

Attempting to use a macro name as a label or variable in an expression.

A0005: section size over 64k

Single section size cannot exceed 64k.

A0006: this chip no support such instruction addressing mode

Not support such instruction format or incorrect chip specified.

A0007: branch too far

You attempt to jump to an address that is out the range. The valid range is -128 ~ +127.

A0008: branch into different section

The branch instruction can't jump to an address that lies in another section.

A0009: file not found

The included file cannot be found.

A0010: macro definition inside another macro definition

You attempt to define a macro inside the definition of another macro.

A0011: incorrect operand

Missing operand in instruction.

A0012: macro name expected

You attempt to define a macro without a macro name.

A0013: undefined Symbol

You attempt to refer a symbol that has not been defined.

A0014: constant expression expected

Some instructions need constant while using a label or variable.

A0015: number too large

The number in instruction is too large and out of range. The legal range is 32-bit integer.

A0016: endif expected before end of file

In the source code file, the numbers of IF and ENDIF are not balances. It needs another ENDIF to keep balance.

A0017: local symbol already defined

The local label has already been defined within its own “local area”, one bounded by labels which keep their definition throughout the entire program.

A0018: ifma should be used inside macro

This occurs when using IFMA directive outside the definition of a macro.

A0019: section name expected

You attempt to create a user-section without section name.

A0020: incorrect section option

Section option is error.

A0021: section name used in expression

Use a section named as a symbol in expression is not allowed.

A0022: add two relative values

This occurs when attempting to add two labels.

A0023: subtract two symbols of different sections

Subtract two symbols defined in different sections.

A0024: subtract constant with relative value

A0025: constant expected

Some instructions need constant while using a label or variable.

A0026: divided by zero

The divisor operand has evaluated to 0.

A0027: symbol already defined other type

A symbol that has already been defined cannot be redefined.

A0028: illegal forward reference

You use a variable that has not been defined.

A0029: global symbol expected

A global symbol is expected to use.

A0030: operand too large

The number in the instruction is too large and out of the instruction addressing mode range.

A0031: address expected

The directives .HIGH., .LOW., .HIGH8., .HIGH16. and .LOW16. will operate at an address.

A0032: incorrect operand

A0033: zero page address expected

A0034: bad use of relative address

This occurs when the expression contains more than one label.

A0035: can't push two-byte operand into one-byte place

A0036: can't push two-byte operand into one-byte place

A0037: can't export such kind of symbol

Can't use the PUBLIC directive to declare a symbol that is defined with VAR or SET directive.

A0038: segment or offset operators should be the outmost operator on an expression

A0039: keyword used as section name

An instruction or a directive is used as section name.

A0040: keyword used as symbol

An instruction or a directive is used as symbol.

A0041: too many sections defined

This occurs when the section you define is over 4096.

A0042: incorrect use of external symbol

A0043: directive DW expected

This occurs when a 16-bit address is used in an 8-bit storage instruction. You can use .LOW. (low 8 bit) or .HIGH. (high 8 bit) prefix to make the low or high byte value relocatable. You also can use the DW directive to

define a 16-bit storage instruction.

A0044: directive DB expected

This occurs when an 8-bit address is used in a 16-bit storage instruction.

A0045: symbol address expected

You attempt to declare a public symbol that is not defined as label, variable and struct variable.

A0046: incorrect conditional assembly

Condition expression is incorrect.

A0047: redefine a macro

You attempt to define a macro that has already been defined.

A0048: zero or positive number expected

You attempt to use a negative number in an instruction, or a positive number is out of the instruction's range.

A0049: can't change section inside procedure

A procedure must be defined inside one section, and cannot change section during the process of the procedure define.

A0050: can't nest procedure

Cannot nest the definition of procedure during the definition of other procedure.

A0051: unexpected end of file before proc end

You have defined a procedure, but missing end definition at the end of the source code file.

A0052: proc and endp unmatched

In the source code file, the number of PROC and ENDP is not balanced. Usually, the number of ENDP is less than PROC.

A0053: not defined a struct name

A0054: error in define 'struct' structure

This occurs when instruction is used during the definition of a struct.

A0055: the 'struct' name are not match

The 'struct' name defined with the STRUCT directive is not the same as defined at the end with the ENDST directive.

A0056: redefine a struct

You attempt to define a struct that has already been defined.

A0057: error in define procedure

A0058: error in define struct variable name

A0059: not define such a struct field

This occurs when you refer to a struct field that has not been defined. Usually, the error occurs if you do not separate the logical operator, relation operator or bit operator with other symbols.

A0060: not defined as a struct variable

This occurs when you refer to a symbol as struct variable.

A0061: not defined struct variable name

A0062: 'end' used inside procedure

Inside a procedure, the END directive cannot be used.

A0063: use address expression with directive float/double

Cannot use the FLOAT/DOUBLE directive to operate an address expression.

A0064: error use directive float/double define string

The FLOAT and DOUBLE directives cannot be used to define a string.

A0065: not defined procedure name

You attempt to define a procedure without a name.

A0066: incorrect parameter

Incorrect operand for instruction.

A0067: struct and endst number not equal

In the source code file, the number of STRUCT and ENDST is not balanced; normally, the number of ENDST is equal to the number of STRUCT.

A0068: can't define a local label field inside struct define

Inside the definition of structure, cannot define a local label field.

A0069: allocate memory failed for macro expansion

A0070: macro parameter length too long

A0071: value expected

Lack of constant value for value concatenation.

A0072: hex and symbol are identical

Symbol name and hex value are ambiguous.

A0073: macro level too deep

This occurs when macro depth-level is over 500.

A0074: operate two symbol of different section

Cause relocation. User cannot operate two symbols of different sections.

A0075: not provide this command

A0076: local symbol can't be declared as public.

A0077: line size over 512 bytes.

A0078: only support 'add' and 'sub' operation for external symbol.

A0079: nested conditional command unbalance detected.

IFCLEAR directive command could be used when a macro contains MACEXIT for early exit.

A0080: no section directive command.

Section must be specified in program code.

24.6.2 Assembler Warning

A2001: incorrect dummy parameter

In the macro definition, the dummy parameter is incorrect.

A2002: incorrect real parameter

In the macro reference, the real parameter is incorrect or is not matched to dummy parameter.

A2003: keyword used as symbol

You attempt to use an instruction or a directive as a symbol.

A2004: unexpected end of line

You attempt to refer to a string which is not correct bracketed with “ ” or ‘ ’.

A2005: unexpected macexit

The MACEXIT directive is used outside of the macro definition.

A2006: unmatched endm

In the source code file, the number of MACRO and ENDM is not balanced. Generally, the number of ENDM is equal to the number of MACRO.

A2007: pass dependence

You attempt to refer to a variable that is defined below.

A2008: operand could be too large

The operand is out of the instruction's range.

A2009: unmatched endp

In the source code file, the number of PROC and ENDP is not balance; normally, the number of ENDP is equal to the number of PROC.

A2010: unexpected else

The ELSE directive is used outside of the condition assembler.

A2011: unexpected endif

The ENDIF directive does not match to IF directive.

A2012: too much parameter

In the macro reference, the number of real parameters is large than number of dummy parameters.

A2013: need an operate number or expression

The storage statement needs an operation number or expression.

A2014: symbol name and hex value are identical

Symbol name and hex value are ambiguous.

A2015: zero page address expected

A2016: subtract constant with relative value

A2017: constant expected, expression mixed with relative value

24.6.3 Linker Errors

L0001: Fatal Error: MFC initialization failed

The Xlink8 cannot run in current platform. An error occurs while MFC library is being initialized.

L0002: No object filename

In command line mode, you must input object filename. The object file is essential.

L0003: Cannot identify the word...

The xlink8 cannot identify the input word.

L0004: Cannot open the file...

The specified file cannot be accessed or opened.

L0005: The library file ... is not Sunplus Library

The specified file is not Sunplus library file.

L0006: No address after the word 'At'

In the section location, there must be an address after the word 'At'.

L0007: No section name after the word 'After'

In the section location, there must be a section name after the word 'After'.

L0008: No section name after the word 'Linkafter'

In the section location, there must be a section name after the word 'Linkafter'.

L0009: The section ... has not been located

In the Linker Script file mode, a section must be located. Otherwise, it will be located at address 00h.

L0010: The external symbol ...has not a public definition

You refer to a local symbol defined in other file as an external symbol. The solution is to declare this symbol with public attribute in the defined file.

L0011: The section ... has not been located before use the section ...after it

In the Linker Script file mode, a section must have been located before being used to locate other sections.

L0012: List file destroyed. Skip update the list file.

The list file has been destroyed and cannot be updated.

L0013: The linkafter section ... has not been defined at any obj file.

In Linker Script file mode, you refer to a section that has not been defined at any file.

L0014: No content in obj file. There is no instruction written in obj file.

24.6.4 Library Maker Errors

L0500: MFC initialization failed

The Xlib8 cannot run in current platform. An error occurs while MFC library is being initialized.

L0510: The second argument must be a lib file

The second argument must be a Sunplus library file, cannot be other library file and the library file's extension must be '.lib'.

L0511: No FIND argument

In the command line, FIND has no argument.

L0512: No ADD argument

In the command line, ADD has no argument.

L0513: No REP argument

In the command line, REP has no argument.

L0514: No DEL argument

In the command line, DEL has no argument.

L0515: Can't identify the command.

In the command line, an unknown command is used.

L0520: Read Lib Error

An error occurs while reading the specified library.

L0522: Can't find the module ...

Cannot find the specified module in the library file.

L0541: Obj Type is not Sunplus !

The specified file is not Sunplus obj file.

24.7 Most Commonly Mistakes

(1) Incomplete instruction

```
.IF ...
...
.ENDIF
```

In above code, "ELSE IF" is an incomplete instruction.

```
.DB      C_Speech,S_19,
```

In above code, it contains an extra comma.

(2) Each comment field must begin with a semi-colon (;)

Incorrect:

```
LDA      #20      Setting IO
LDA      RTDataBank    =1000H-(ClockRate/24/1024/1024)
DB       00,01,02,03,04,05,06,07,08      00,01
```

Correct:

```
LDA      #20          ; Setting IO
LDA      RTDataBank    ; =1000H-(ClockRate/24/1024/1024)
DB       00,01,02,03,04,05,06,07,08 ; 00,01
```

(3) For macro value concatenation, it is illegal to put an expression in the angle brackets.

```
JSR      F_ServiceTask<_TASK_COUNT_-1>
```

It should be rewritten as follows:

```
EXPR_VALUE      VAR      _TASK_COUNT_-1
JSR      F_ServiceTask<EXPR_VALUE>
```

(4) It is illegal to use reserved word as symbol name.

```
LONG      EQU      1
HIGH:    LDA      #11
```

(5) The ORG value expired 255 is illegal for PAGE0 section.

```
.PAGE0
.....
```

.ORG 600H

(6) Illegal symbol definition.

KEY_@:

LDA #GRAD\$

L_#Compare?:

(7) A string must be enclosed in double/single apostrophes. If there involves single apostrophes in the string, it should be enclosed in double apostrophes. The same rule applies to double apostrophes.

v_temp10: .db "XASM"S ; illegal

Rewrite as:

.db 'XASM"S'

Or .db "XASM'S"

(8) Not support UNIX/MAC file format. It should be pre-processed.

(9) The instruction of exclusive-OR memory with Accumulator is "EOR", not "XOR" (XOR is a bit operator)

(10) Xasm8 does not support any logic or math operation involving two or more symbols except subtraction or in absolute mode (non-relocatable mode).

LDA Label1+Label2 ; illegal

LDA .LOW. (Label1 & Label2) ; illegal

LDA Label1+10 ; legal

LDA Label1-10 ; legal

LDA 100 - Label1 ; legal

LDA Label2-Label1 ; legal

(11) It is illegal for value exceeds 255 or symbol is not in PAGE0 section for immediate address mode.

.PAGE0

Var1: .DS 1

.DATA

Var2: .DS 1

Count: EQU 156

Price: EQU 371

.CODE

```
LDA      #Var1      ; legal
LDA      #Count     ; legal
LDA      #Var2      ; illegal; rewrite to LDA #.LOW.Var2
LDA      #Price     ; illegal; count value over 255
```

(12). If a label does not have colon at the end of label name, the label must start in column 1. However, if an instruction set is placed in column 1 and intended to be used as a label, the compiler will identify it as an instruction automatically. That is, all reserved words cannot be used as labels.

(13). Not support forward reference. A symbol must be defined before it was used.

ex.

Incorrect:

```
D_EndAddress: EQU(D_StartAddress+D_Segment*D_Common/4)
D_Segment:    EQU    160
D_Common:     EQU    D_Const
D_Const:      EQU    120
```

Correct:

```
D_Const:      EQU 120
D_Segment:    EQU 160
D_Common:     EQU D_Const
D_EndAddress: EQU (D_StartAddress+D_Segment*D_Common/4)
```

(14). Not support multi defined symbol. A symbol can only been defined for one time in a file except command "VAR".

ex.

```
My_ADD .MACRO sum,p1,p2
    LDA p1
    CLC
    ADC p2
    STA sum
    .ENDM
My_ADD:
    LDA V1
    CLC
    ADC V2
    ADC V3
    ...
```

(15). Relative addressing label was not allowed in complicated expression operation except the below forms. Because the final address was decided in linking time, not compiling time.

Form 1: Label1-Label2

Form 2: Label1+immediate value

Form 3: Label1-immediate value

ex.

Tab1:

...

Tab2:

...

```
.DW (Tab1*Tab2/300) ;illegal  
.DW (Tab1+Tab2) ;illegal  
.DW (Tab1/2) ;illegal  
.DW (Tab2-Tab1) ;legal  
.DW (Tab1+17) ;legal  
.DW (Tab2-9) ;legal
```

Label wasn't such limitation in absolute addressing mode, because the address was specified in compiling time.

.Absolute

.ORG 600H

Tab1:

...

Tab2:

...

```
.DW (Tab1*Tab2/300) ;legal  
.DW (Tab1+Tab2) ;legal  
.DW (Tab1/2) ;legal  
.DW (Tab2-Tab1) ;legal  
.DW (Tab1+17) ;legal  
.DW (Tab2-9) ;legal
```

(16). Segment or offset operators should be the outmost operator on an expression.

ex.

Row: .EQU 12

Column: .EQU 5

```
.DW .HIGH8.Label1+$10000 ;illegal
```

```
.DW .LOW.Label2+Row*Column      ;illegal
LDA .LOW.Label1+.HIGH.Label2    ;illegal
.DW .HIGH8.(Tab1+$1)           ;legal
.DW .LOW.(Label2+Row*Column)   ;legal
```

(17). Local label is valid only within its own "local area", one bounded by labels.

ex.

```
%FG_PutStr:     .MACRO  STRPTR
%FP_SaveRegs
.IFSSEQ 1,'#',STRPTR
    jmp      ?stringgo#          ;illegal
    string#:   .VAR      $
    .DB      STRPTR,0
    ?stringgo#:
        mov2i   RG_StringAddr,string#
        exit#:
.ENDIF
...
.ENDM
```

Label "?stringgo#" is only valid between "string#" with "exit#".

(18). Xlink8 not support such as "group" and "range" commands.

24.8 Related Application Notes and Libraries

This section lists some application notes and libraries about various usage of SPMC65X family. The application notes may help user develop quickly and offer the concepts about the usage of this topic. Sunplus also supports some powerful libraries about general operation, mathematics, communication, etc. The purpose of library is to help user speed up development. The current application notes and libraries related to Assemble Tool are listed below.

Title	Application Note Series Number
No associated application notes at this time.	
Library:	
Title	Application Note Series Number
SPMC65X series software basics manual	AN_O0100.DOC
SPMC65X series software advances 1 manual (data processing operation)	AN_O0101.DOC



SUNPLUS

SPMC65X Family PROGRAMMING GUIDE

SPMC65X series software advances 2 manual (mathematical operation) AN_O0102.DOC

24.9 Revision History

Revision	Date	Remark
V0.1	03/31/2005	First edition

25 Instruction Set

25.1 Description

This manual intends to guide users through the 6502 Instruction sets. All 6502 instructions are listed in alphabetical order. They are different with general 6502 instruction sets. Besides 6502 instructions, we add some bit operation instructions such as CLR, INV, SET and TST, so-called 65B02.

25.2 Format of Assembly Language Instruction

There are four parts of assembly language instruction.

[label :] OP-code [operand] [; comment]

[] : represents optional item.

Label field It labels an instruction. Programmers are able to use the label as an address. Some rules should be applied:

- Start in column 1 or use a colon (:) at the end of a label.
- Start with a letter.
- Do not use the name of OP-code or register.
- 1 to 32 characters
- Avoid special symbols

Note: Letter is ‘_’ + ‘A~Z’ or ‘a~z’ and ‘?’.

OP-Code field It is an instruction field.

Operand field It can be data or addresses used in the program. When OP-Code is a single byte, operand field is omitted. When the address mode is immediate, it is a byte of data. It is a symbol for a location where a byte of data is found. It is a label when it refers to a program address.

Comment field The comment field will increase the program's readability. A semicolon (;) should be placed at the beginning of comment.

For example:

```
LDA #00      ; load data 00 to A  
STA Counter  ; load value of A into Counter
```

Note: A space is needed between two fields.

25.3 Instructions

n ADC

Add to Accumulator with Carry, (A+M+C) à A, C

Addressing mode	Assembly Language Form	Opcode	No. Bytes	No. Cycles
Immediate	ADC #dd	69H	2	2
Zero Page	ADC aa	65H	2	3
Zero Page, X	ADC aa, X	75H	2	4
Absolute	ADC aaaa	6DH	3	4
Absolute, X	ADC aaaa, X	7DH	3	4
Absolute, Y	ADC aaaa, Y	79H	3	4
(Indirect, X)	ADC (aa, X)	61H	2	6
(Indirect), Y	ADC (aa), Y	71H	2	6

* Add 1 clock cycle if page boundary is crossed.

N	V	D	I	Z	C
!	!		*	-	!

N: Set if result is negative

V: Set if arithmetic overflow occurs.

Z: Set if result is 0

C: Set if there is a carry from the most significant bit of the result.

D: * if set to 1, the ADC performs decimal operation.

n AND

AND memory data with Accumulator, (A^M) à A

Addressing mode	Assembly Language Form	Opcode	No. Bytes	No. Cycles
Immediate	AND #dd	29H	2	2
Zero Page	AND aa	25H	2	3
Zero Page, X	AND aa, X	35H	2	4
Absolute	AND aaaa	2DH	3	4
Absolute, X	AND aaaa, X	3DH	3	4
Absolute, Y	AND aaaa, Y	39H	3	4
(Indirect, X)	AND (aa, X)	21H	2	6
(Indirect), Y	AND (aa), Y	31H	2	6

* Add 1 clock cycle if page boundary is crossed.

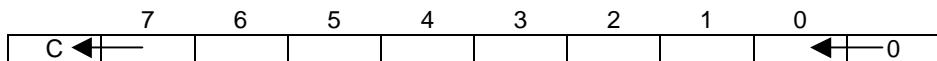
N	V	D	I	Z	C
!	-		-	-	!

N: Set if result is negative

Z: Set if result is 0

n ASL

Arithmetic Shift Left



Addressing mode	Assembly Language Form	Opcode	No. Bytes	No. Cycles
Accumulator	ASL A	0AH	1	2
Zero Page	ASL aa	06H	2	5
Zero Page, X	ASL aa, X	16H	2	6
Absolute	ASL aaaa	0EH	3	6
Absolute, X	ASL aaaa, X	1EH	3	6

* Add 1 clock cycle if page boundary is crossed.



N: Set if result is negative

Z: Set if result is 0

C: Set if the bit shifted from the most significant bit is 1 .

n BCC/BCS/BEQ/BMI/BNE/BPL/BVC/BVS

Branch to aa if condition is true.

The range of relative addressing is -128 (backward) and +127 (forward) bytes.

Assembly Language Form	Condition	Opcode	No. Bytes	No. Cycles
BCC aa	C=0	90H	2	2*
BCS aa	C=1	B0H	2	2*
BEQ aa	Z=1	F0H	2	2*
BMI aa	N=1	30H	2	2*
BNE aa	Z=0	D0H	2	2*
BPL aa	N=0	10H	2	2*
BVC aa	V=0	50H	2	2*
BVS aa	V=1	70H	2	2*

* Add 1 clock cycle if branch occurs.



n BIT

Test bit in memory with Accumulator

Addressing mode	Assembly Language Form	Opcode	No. Bytes	No. Cycles
Zero Page	BIT aa	24H	2	3
Absolute	BIT aaaa	2CH	3	4

N	V		D	I	Z	C
!	!		-	-	!	-

N: Set if memory bit7 of the result is 1

V: Set if memory bit 6 of the result is 1.

Z: Set if result is 0

n CLC

Clear Carry flag

Addressing mode	Assembly Language Form	Opcode	No. Bytes	No. Cycles
Implied	CLC	18H	1	2

N	V		D	I	Z	C
-	-		-	-	-	!

C: Unconditionally cleared.

n CLD

Clear Decimal mode

Addressing mode	Assembly Language Form	Opcode	No. Bytes	No. Cycles
Implied	CLD	D8H	1	2

N	V		D	I	Z	C
-	-		!	-	-	-

D: Unconditionally cleared.

n CLI

Clear Interrupt mask. (enable interrupt)

Addressing mode	Assembly Language Form	Opcode	No. Bytes	No. Cycles
Implied	CLI	58H	1	2

N	V		D	I	Z	C
-	-		-	!	-	-

I: Unconditionally cleared.

n CLR

Bit Clear.

Clear BITn of \$aa as "0".

Addressing mode	Assembly Language Form	Opcode	No. Bytes	No. Cycles
Zero Page	CLR aa, 0	0FH	2	5
Zero Page	CLR aa, 1	1FH	2	5
Zero Page	CLR aa, 2	2FH	2	5
Zero Page	CLR aa, 3	3FH	2	5
Zero Page	CLR aa, 4	4FH	2	5
Zero Page	CLR aa, 5	5FH	2	5
Zero Page	CLR aa, 6	6FH	2	5
Zero Page	CLR aa, 7	7FH	2	5

X: Not available.

N	V	D	I	Z	C
-	-		-	-	-

n CLV

Clear overflow

Addressing mode	Assembly Language Form	Opcode	No. Bytes	No. Cycles
Implied	CLV	B8H	1	2

N	V	D	I	Z	C
-	!		-	-	-

V: Unconditionally cleared.

n CMP

Compare memory data with Accumulator, A - M

Addressing mode	Assembly Language Form	Opcode	No. Bytes	No. Cycles
Immediate	CMP #dd	C9H	2	2
Zero Page	CMP aa	C5H	2	3
Zero Page, X	CMP aa, X	D5H	2	4
Absolute	CMP aaaa	CDH	3	4
Absolute, X	CMP aaaa, X	DDH	3	4
Absolute, Y	CMP aaaa, Y	D9H	3	4
(Indirect, X)	CMP (aa, X)	C1H	2	6
(Indirect), Y	CMP (aa), Y	D1H	2	6

* Add 1 clock cycle if page boundary is crossed.

N	V	D	I	Z	C
!	-		-	!	!

N: Set if result is negative

Z: Set if result is 0

C: Set if a “borrow” not occurred. (A > = M)

n CPX

Compare memory data with Register X, X - data

Addressing mode	Assembly Language Form	Opcode	No. Bytes	No. Cycles
Immediate	CPX #dd	E0H	2	2
Zero Page	CPX aa	E4H	2	3
Absolute	CPX aaaa	ECH	3	4

N	V	D	I	Z	C
!	-		-	!	!

N: Set if result is negative

Z: Set if result is 0

C: Set if a “borrow” not occurred. (X > = data)

n CPY

Compare memory data with Register Y, Y - data

Addressing mode	Assembly Language Form	Opcode	No. Bytes	No. Cycles
Immediate	CPY #dd	C0H	2	2
Zero Page	CPY aa	C4H	2	3
Absolute	CPY aaaa	CCH	3	4

N	V	D	I	Z	C
!	-		-	!	!

N: Set if result is negative

Z: Set if result is 0

C: Set if a “borrow” not occurred (Y > = data)

n DEC

Decrement memory by one

Addressing mode	Assembly Language Form	Opcode	No. Bytes	No. Cycles
Zero Page	DEC aa	C6H	2	5
Zero Page, X	DEC aa, X	D6H	2	6
Absolute	DEC aaaa	CEH	3	6
Absolute, X	DEC aaaa, X	DEH	3	6

N	V	D	I	Z	C

!	-			-	-	!	-
---	---	--	--	---	---	---	---

N: Set if result is negative

Z: Set if result is 0

n DEX

Decrement Register X by one

Addressing mode	Assembly Language Form	Opcode	No. Bytes	No. Cycles
Implied	DEX	CAH	1	2

N	V	D	I	Z	C
!	-			-	-

N: Set if result is negative

Z: Set if result is 0

n DEY

Decrement Register Y by one

Addressing mode	Assembly Language Form	Opcode	No. Bytes	No. Cycles
Implied	DEY	88H	1	2

N	V	D	I	Z	C
!	-			-	-

N: Set if result is negative

Z: Set if result is 0

n EOR

Exclusive-OR memory with Accumulator, A \oplus A XOR memory

Addressing mode	Assembly Language Form	Opcode	No. Bytes	No. Cycles
Immediate	EOR #dd	49H	2	2
Zero Page	EOR aa	45H	2	3
Zero Page, X	EOR aa, X	55H	2	4
Absolute	EOR aaaa	4DH	3	4
Absolute, X	EOR aaaa, X	5DH	3	4
Absolute, Y	EOR aaaa, Y	59H	3	4
(Indirect, X)	EOR (aa, X)	41H	2	6
(Indirect), Y	EOR (aa), Y	51H	2	6

* Add 1 clock cycle if page boundary is crossed.

N	V		D	I	Z	C	
!	-		-	-	!	-	

N: Set if result is negative

Z: Set if result is 0

n INC

Increment memory by one

Addressing mode	Assembly Language Form	Opcode	No. Bytes	No. Cycles
Zero Page	INC aa	E6H	2	5
Zero Page, X	INC aa, X	F6H	2	6
Absolute	INC aaaa	EEH	3	6
Absolute, X	INC aaaa, X	FEH	3	6

N	V		D	I	Z	C	
!	-		-	-	!	-	

N: Set if result is negative

Z: Set if result is 0

n INV

Bit Inverse.

Toggle BITn of \$aa.

Addressing mode	Assembly Language Form	Opcode	No. Bytes	No. Cycles
Zero Page	INV aa, 0	87H	2	5
Zero Page	INV aa, 1	97H	2	5
Zero Page	INV aa, 2	A7H	2	5
Zero Page	INV aa, 3	B7H	2	5
Zero Page	INV aa, 4	C7H	2	5
Zero Page	INV aa, 5	D7H	2	5
Zero Page	INV aa, 6	E7H	2	5
Zero Page	INV aa, 7	F7H	2	5

N	V		D	I	Z	C	
-	-		-	-	-	-	

n INX

Increment Register X by one

Addressing mode	Assembly Language Form	Opcode	No. Bytes	No. Cycles
Implied	INX	E8H	1	2

N	V		D	I	Z	C	
!	-			-	-	!	-

N: Set if result is negative

Z: Set if result is 0

n INY

Increment Register Y by one

Addressing mode	Assembly Language Form	Opcode	No. Bytes	No. Cycles
Implied	INY	C8H	1	2

N	V		D	I	Z	C	
!	-			-	-	!	-

N: Set if result is negative

Z: Set if result is 0

n JMP

Jump to specified location

Addressing mode	Assembly Language Form	Opcode	No. Bytes	No. Cycles
Absolute	JMP aaaa	4CH	3	3
Indirect	JMP (aaaa)	6CH	3	5

N	V		D	I	Z	C	
-	-			-	-	-	-

n JSR

Jump to subroutine

Addressing mode	Assembly Language Form	Opcode	No. Bytes	No. Cycles
Absolute	JSR aaaa	20H	3	6

With JSR instruction, the current address will be pushed on stack and then jumps to the specified subroutine. At the end of subroutine procedure, the RTS (return from subroutine) instruction can be used to return to the original program flow by popping saved address from stack.

N	V		D	I	Z	C	
-	-			-	-	-	-

n LDA

Load memory data or data into Accumulator, A **B** data

Addressing mode	Assembly Language Form	Opcode	No. Bytes	No. Cycles
Immediate	LDA #dd	A9H	2	2
Zero Page	LDA aa	A5H	2	3
Zero Page, X	LDA aa, X	B5H	2	4
Absolute	LDA aaaa	ADH	3	4
Absolute, X	LDA aaaa, X	BDH	3	4
Absolute, Y	LDA aaaa, Y	B9H	3	4
(Indirect, X)	LDA (aa, X)	A1H	2	6
(Indirect), Y	LDA (aa), Y	B1H	2	6

* Add 1 clock cycle if page boundary is crossed.

X: Not available.

N	V	D	I	Z	C
!	-		-	!	-

N: Set if result is negative

Z: Set if result is 0

n LDX

Load memory data or data into Register X, X **B** data

Addressing mode	Assembly Language Form	Opcode	No. Bytes	Available Instruction & No. Cycles
Immediate	LDX #dd	A2H	2	2
Zero Page	LDX aa	A6H	2	3
Zero Page, Y	LDX aa, Y	B6H	2	4
Absolute	LDX aaaa	AEH	3	4
Absolute, Y	LDX aaaa, Y	BEH	3	4

* Add 1 clock cycle if page boundary is crossed.

N	V	D	I	Z	C
!	-		-	!	-

N: Set if result is negative

Z: Set if result is 0

n LDY

Load memory data or data into Register Y, Y **B** data

Addressing mode	Assembly Language Form	Opcode	No. Bytes	No. Cycles
Immediate	LDY #dd	A0H	2	2
Zero Page	LDY aa	A4H	2	3
Zero Page, X	LDY aa, X	B4H	2	4
Absolute	LDY aaaa	ACH	3	4
Absolute, X	LDY aaaa, X	BCH	3	4

* Add 1 clock cycle if page boundary is crossed.

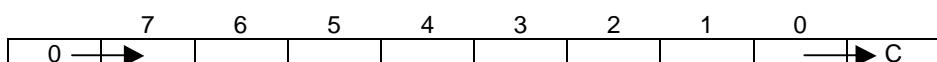
N	V		D	I	Z	C
!	-		-	-	!	-

N: Set if result is negative

Z: Set if result is 0

n LSR

Logical Shift Right



Addressing mode	Assembly Language Form	Opcode	No. Bytes	No. Cycles
Accumulator	LSR A	4AH	1	2
Zero Page	LSR aa	46H	2	5
Zero Page, X	LSR aa, X	56H	2	6
Absolute	LSR aaaa	4EH	3	6
Absolute, X	LSR aaaa, X	5EH	3	6

* Add 1 clock cycle if page boundary is crossed.

N	V		D	I	Z	C
!	-		-	-	!	!

N: Set if result is negative

Z: Set if result is 0

C: Set if the bit shifted from the least significant bit is 1.

n NOP

No operation

Addressing mode	Assembly Language Form	Opcode	No. Bytes	No. Cycles
Implied	NOP	EAH	1	2

N	V		D	I	Z	C
-	-		-	-	-	-

n ORA

OR memory with Accumulator, A OR A OR memory

Addressing mode	Assembly Language Form	Opcode	No. Bytes	Available Instruction & No. Cycles
Immediate	ORA #dd	09H	2	2

Zero Page	ORA aa	05H	2	3
Zero Page, X	ORA aa, X	15H	2	4
Absolute	ORA aaaa	0DH	3	4
Absolute, X	ORA aaaa, X	1DH	3	4
Absolute, Y	ORA aaaa, Y	19H	3	4
(Indirect, X)	ORA (aa, X)	01H	2	6
(Indirect), Y	ORA (aa), Y	11H	2	6

* Add 1 clock cycle if page boundary is crossed.

N	V	D	I	Z	C
!	-		-	!	-

N: Set if result is negative

Z: Set if result is 0

n PHA

Push Accumulator on Stack

Addressing mode	Assembly Language Form	Opcode	No. Bytes	No. Cycles
Implied	PHA	48H	1	3

N	V	D	I	Z	C
-	-		-	-	-

n PHP

Push Status Flag on Stack

Addressing mode	Assembly Language Form	Opcode	No. Bytes	No. Cycles
Implied	PHP	08H	1	3

N	V	D	I	Z	C
-	-		-	-	-

n PLA

Pull Accumulator from Stack

Addressing mode	Assembly Language Form	Opcode	No. Bytes	No. Cycles
Implied	PLA	68H	1	4

N	V	D	I	Z	C
!	-		-	!	-

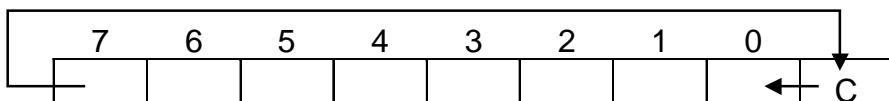
n PLP

Pull Status Flag from Stack

Addressing mode	Assembly Language Form	Opcode	No. Bytes	No. Cycles
Implied	PLP	28H	1	4


n ROL

Rotate Left



Addressing mode	Assembly Language Form	Opcode	No. Bytes	No. Cycles
Accumulator	ROL A	2AH	1	2
Zero Page	ROL aa	26H	2	5
Zero Page, X	ROL aa, X	36H	2	6
Absolute	ROL aaaa	2EH	3	6
Absolute, X	ROL aaaa, X	3EH	3	6

* Add 1 clock cycle if page boundary is crossed.



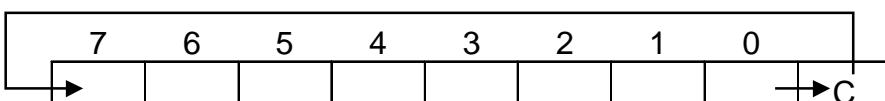
N: Set if result is negative

Z: Set if result is 0

C: Set if the bit shifted from the most significant bit position is 1.

n ROR

Rotate Right



Addressing mode	Assembly Language Form	Opcode	No. Bytes	No. Cycles
Accumulator	ROR A	6AH	1	2
Zero Page	ROR aa	66H	2	5
Zero Page, X	ROR aa, X	76H	2	6
Absolute	ROR aaaa	6EH	3	6
Absolute, X	ROR aaaa, X	7EH	3	6

* Add 1 clock cycle if page boundary is crossed.

N	V		D	I	Z	C	
!	-			-	-	!	!

N: Set if result is negative

Z: Set if result is 0

C: Set if the bit shifted from the least significant bit position is 1.

n RTI

Return from Interrupt

Addressing mode	Assembly Language Form	Opcode	No. Bytes	No. Cycles
Implied	RTI	40H	1	6

N	V		D	I	Z	C	

N: Restored from stack

V: Restored from stack

D, I: Restored from stack

Z: Restored from stack

C: Restored from stack

n RTS

Return from Subroutine

Addressing mode	Assembly Language Form	Opcode	No. Bytes	No. Cycles
Implied	RTS	60H	1	6

N	V		D	I	Z	C	
-	-			-	-	-	-

n SBC

Subtract from Accumulator with Carry's complement, (A-M - \bar{C}) \rightarrow A, C

Addressing mode	Assembly Language Form	Opcode	No. Bytes	No. Cycles
Immediate	SBC #dd	E9H	2	2
Zero Page	SBC aa	E5H	2	3
Zero Page, X	SBC aa, X	F5H	2	4
Absolute	SBC aaaa	EDH	3	4
Absolute, X	SBC aaaa, X	FDH	3	4
Absolute, Y	SBC aaaa, Y	F9H	3	4
(Indirect, X)	SBC (aa, X)	E1H	2	6
(Indirect), Y	SBC (aa), Y	F1H	2	6

* Add 1 clock cycle if page boundary is crossed.

N	V		D	I	Z	C	
!	!		*	-	!	!	

N: Set if result is negative

V: Set if arithmetic overflow occurs.

Z: Set if result is 0

C: Set if there is no “borrow” occurred. (A > M).

D: * if set to 1, the ADC performs decimal operation.

n SEC

Set Carry Flag to 1, C B1

Addressing mode	Assembly Language Form	Opcode	No. Bytes	No. Cycles
Implied	SEC	38H	1	2

N	V		D	I	Z	C	
-	-		-	-	-	!	

C: Unconditionally Set

n SED

Set Decimal Mode to 1, D B1

Addressing mode	Assembly Language Form	Opcode	No. Bytes	No. Cycles
Implied	SED	F8H	1	2

N	V		D	I	Z	C	
-	-		!	-	-	-	

D: Unconditionally Set

n SEI

Set Interrupt Disable flag to 1, I B1 (Disable Interrupt)

Addressing mode	Assembly Language Form	Opcode	No. Bytes	No. Cycles
Implied	SEI	78H	1	2

N	V		D	I	Z	C	
-	-		-	!	-	-	

I: Unconditionally Set

n SET

Bit Set.

Set BITn of \$aa as "1".

Addressing mode	Assembly Language Form	Opcode	No. Bytes	No. Cycles
Zero Page	SET aa, 0	8FH	2	5
Zero Page	SET aa, 1	9FH	2	5
Zero Page	SET aa, 2	AFH	2	5
Zero Page	SET aa, 3	BFH	2	5
Zero Page	SET aa, 4	CFH	2	5
Zero Page	SET aa, 5	DFH	2	5
Zero Page	SET aa, 6	EFH	2	5
Zero Page	SET aa, 7	FFH	2	5

X: Not available.

N	V	D	I	Z	C
-	-		-	-	-

n STA

Store Accumulator in memory, M \leftarrow A

Addressing mode	Assembly Language Form	Opcode	No. Bytes	No. Cycles
Zero Page	STA aa	85H	2	3
Zero Page, X	STA aa, X	95H	2	4
Absolute	STA aaaa	8DH	3	4
Absolute, X	STA aaaa, X	9DH	3	4
Absolute, Y	STA aaaa, Y	99H	3	4
(Indirect, X)	STA (aa, X)	81H	2	6
(Indirect), Y	STA (aa), Y	91H	2	6

N	V	D	I	Z	C
-	-		-	-	-

n STX

Store Register X in memory, M \leftarrow X

Addressing mode	Assembly Language Form	Opcode	No. Bytes	No. Cycles
Zero Page	STX aa	86H	2	3
Zero Page, Y	STX aa, Y	96H	2	4
Absolute	STX aaaa	8EH	3	4

N	V	D	I	Z	C
-	-		-	-	-

n STY

Store Register Y in memory, M \rightarrow Y

Addressing mode	Assembly Language Form	Opcode	No. Bytes	No. Cycles
Zero Page	STY aa	84H	2	3
Zero Page, X	STY aa, X	94H	2	4
Absolute	STY aaaa	8CH	3	4


n TAX

Transfer Accumulator to Index X, X \rightarrow A

Addressing mode	Assembly Language Form	Opcode	No. Bytes	No. Cycles
Implied	TAX	AAH	1	2



N: Set if the result is negative

Z: Set if the result is 0

n TAY

Transfer Accumulator to Index Y, Y \rightarrow A

Addressing mode	Assembly Language Form	Opcode	No. Bytes	No. Cycles
Implied	TAY	A8H	1	2



N: Set if the result is negative

Z: Set if the result is 0

n TST

Bit Test.

Read and judge BITn of \$aa.

Addressing mode	Assembly Language Form	Opcode	No. Bytes	No. Cycles
Zero Page	TST aa, 0	07H	2	3
Zero Page	TST aa, 1	17H	2	3

Zero Page	TST aa, 2	27H	2	3
Zero Page	TST aa, 3	37H	2	3
Zero Page	TST aa, 4	47H	2	3
Zero Page	TST aa, 5	57H	2	3
Zero Page	TST aa, 6	67H	2	3
Zero Page	TST aa, 7	77H	2	3

N	V	D	I	Z	C
-	-		-	!	-

Z: Set if BITn of \$aa is 0.

n TSX

Transfer Stack to Index X, X **B** S

Addressing mode	Assembly Language Form	Opcode	No. Bytes	No. Cycles
Implied	TSX	BAH	1	2

N	V	D	I	Z	C
!	-		-	!	-

N: Set if the result is negative

Z: Set if the result is 0

n TXA

Transfer Register X to Accumulator, A **B** X

Addressing mode	Assembly Language Form	Opcode	No. Bytes	No. Cycles
Implied	TXA	8AH	1	2

N	V	D	I	Z	C
!	-		-	!	-

N: Set if the result is negative

Z: Set if the result is 0

n TXS

Transfer Register X to Stack, S **B** X

Addressing mode	Assembly Language Form	Opcode	No. Bytes	No. Cycles
Implied	TXS	9AH	1	2

N	V	D	I	Z	C
-	-		-	-	-

**n TYA**Transfer Register Y to Accumulator, A \leftarrow Y

Addressing mode	Assembly Language Form	Opcode	No. Bytes	No. Cycles
Implied	TYA	98H	1	2

N	V	D	I	Z	C
!	-		-	!	-

N: Set if the result is negative

Z: Set if the result is 0

25.4 Summary of Available Instruction sets

Assembly Form	Addressing mode	Opcode	No. Bytes	No. Cycles	Flags							
					N	V			D	I	Z	C
ADC #dd	Immediate	69H	2	2	!	!			*	-	!	!
ADC aa	Zero Page	65H	2	3	!	!			*	-	!	!
ADC aa, X	Zero Page, X	75H	2	4	!	!			*	-	!	!
ADC aaaa	Absolute	6DH	3	4	!	!			*	-	!	!
ADC aaaa, X	Absolute, X	7DH	3	4	!	!			*	-	!	!
ADC aaaa, Y	Absolute, Y	79H	3	4	!	!			*	-	!	!
ADC (aa, X)	(Indirect, X)	61H	2	6	!	!			*	-	!	!
ADC (aa), Y	(Indirect), Y	71H	2	6	!	!			*	-	!	!
AND #dd	Immediate	29H	2	2	!	-			-	-	!	-
AND aa	Zero Page	25H	2	3	!	-			-	-	!	-
AND aa, X	Zero Page, X	35H	2	4	!	-			-	-	!	-
AND aaaa	Absolute	2DH	3	4	!	-			-	-	!	-
AND aaaa, X	Absolute, X	3DH	3	4	!	-			-	-	!	-
AND aaaa, Y	Absolute, Y	39H	3	4	!	-			-	-	!	-
AND (aa, X)	(Indirect, X)	21H	2	6	!	-			-	-	!	-
AND (aa), Y	(Indirect), Y	31H	2	6	!	-			-	-	!	-
ASL A	Accumulator	0AH	1	2	!	-			-	-	!	!
ASL aa	Zero Page	06H	2	5	!	-			-	-	!	!
ASL aa, X	Zero Page, X	16H	2	6	!	-			-	-	!	!
ASL aaaa	Absolute	0EH	3	6	!	-			-	-	!	!
ASL aaaa, X	Absolute, X	1EH	3	6	!	-			-	-	!	!
BCC aa	Relative	90H	2	2*	-	-			-	-	-	-
BCS aa	Relative	B0H	2	2*	-	-			-	-	-	-
BEQ aa	Relative	F0H	2	2*	-	-			-	-	-	-
BMI aa	Relative	30H	2	2*	-	-			-	-	-	-
BNE aa	Relative	D0H	2	2*	-	-			-	-	-	-
BPL aa	Relative	10H	2	2*	-	-			-	-	-	-
BVC aa	Relative	50H	2	2*	-	-			-	-	-	-
BVS aa	Relative	70H	2	2*	-	-			-	-	-	-
BIT aa	Zero Page	24H	2	3	!	!			-	-	!	-
BIT aaaa	Absolute	2CH	3	4	!	!			-	-	!	-
CLC	Implied	18H	1	2	-	-			-	-	-	!
CLD	Implied	D8H	1	2	-	-			!	-	-	-
CLI	Implied	58H	1	2	-	-			-	!	-	-
CLR aa, 0	Zero Page	0FH	2	5	-	-			-	-	-	-
CLR aa, 1	Zero Page	1FH	2	5	-	-			-	-	-	-
CLR aa, 2	Zero Page	2FH	2	5	-	-			-	-	-	-
CLR aa, 3	Zero Page	3FH	2	5	-	-			-	-	-	-
CLR aa, 4	Zero Page	4FH	2	5	-	-			-	-	-	-
CLR aa, 5	Zero Page	5FH	2	5	-	-			-	-	-	-
CLR aa, 6	Zero Page	6FH	2	5	-	-			-	-	-	-
CLR aa, 7	Zero Page	7FH	2	5	-	-			-	-	-	-
CLV	Implied	B8H	1	2	-	!			-	-	-	-
CMP #dd	Immediate	C9H	2	2	!	-			-	-	!	!
CMP aa	Zero Page	C5H	2	3	!	-			-	-	!	!
CMP aa, X	Zero Page, X	D5H	2	4	!	-			-	-	!	!

CMP aaaa	Absolute	CDH	3	4	!	-	-	-	-	!	!
CMP aaaa, X	Absolute, X	DDH	3	4	!	-	-	-	-	!	!
CMP aaaa, Y	Absolute, Y	D9H	3	4	!	-	-	-	-	!	!
CMP (aa, X)	(Indirect, X)	C1H	2	6	!	-	-	-	-	!	!
CMP (aa), Y	(Indirect), Y	D1H	2	6	!	-	-	-	-	!	!
CPX #dd	Immediate	E0H	2	2	!	-	-	-	-	!	!
CPX aa	Zero Page	E4H	2	3	!	-	-	-	-	!	!
CPX aaaa	Absolute	ECH	3	4	!	-	-	-	-	!	!
CPY #dd	Immediate	C0H	2	2	!	-	-	-	-	!	!
CPY aa	Zero Page	C4H	2	3	!	-	-	-	-	!	!
CPY aaaa	Absolute	CCH	3	4	!	-	-	-	-	!	!
DEC aa	Zero Page	C6H	2	5	!	-	-	-	-	!	-
DEC aa, X	Zero Page, X	D6H	2	6	!	-	-	-	-	!	-
DEC aaaa	Absolute	CEH	3	6	!	-	-	-	-	!	-
DEC aaaa, X	Absolute, X	DEH	3	6	!	-	-	-	-	!	-
DEX	Implied	CAH	1	2	!	-	-	-	-	!	-
DEY	Implied	88H	1	2	!	-	-	-	-	!	-
EOR #dd	Immediate	49H	2	2	!	-	-	-	-	!	-
EOR aa	Zero Page	45H	2	3	!	-	-	-	-	!	-
EOR aa, X	Zero Page, X	55H	2	4	!	-	-	-	-	!	-
EOR aaaa	Absolute	4DH	3	4	!	-	-	-	-	!	-
EOR aaaa, X	Absolute, X	5DH	3	4	!	-	-	-	-	!	-
EOR aaaa, Y	Absolute, Y	59H	3	4	!	-	-	-	-	!	-
EOR (aa, X)	(Indirect, X)	41H	2	6	!	-	-	-	-	!	-
EOR (aa), Y	(Indirect), Y	51H	2	6	!	-	-	-	-	!	-
INC aa	Zero Page	E6H	2	5	!	-	-	-	-	!	-
INC aa, X	Zero Page, X	F6H	2	6	!	-	-	-	-	!	-
INC aaaa	Absolute	EEH	3	6	!	-	-	-	-	!	-
INC aaaa, X	Absolute, X	FEH	3	6	!	-	-	-	-	!	-
INV aa, 0	Zero Page	87H	2	5	-	-	-	-	-	-	-
INV aa, 1	Zero Page	97H	2	5	-	-	-	-	-	-	-
INV aa, 2	Zero Page	A7H	2	5	-	-	-	-	-	-	-
INV aa, 3	Zero Page	B7H	2	5	-	-	-	-	-	-	-
INV aa, 4	Zero Page	C7H	2	5	-	-	-	-	-	-	-
INV aa, 5	Zero Page	D7H	2	5	-	-	-	-	-	-	-
INV aa, 6	Zero Page	E7H	2	5	-	-	-	-	-	-	-
INV aa, 7	Zero Page	F7H	2	5	-	-	-	-	-	-	-
INX	Implied	E8H	1	2	!	-	-	-	-	!	-
INY	Implied	C8H	1	2	!	-	-	-	-	!	-
JMP aaaa	Absolute	4CH	3	3	-	-	-	-	-	-	-
JMP (aaaa)	Indirect	6CH	3	5	-	-	-	-	-	-	-
JSR aaaa	Absolute	20H	3	6	-	-	-	-	-	-	-
LDA #dd	Immediate	A9H	2	2	!	-	-	-	-	!	-
LDA aa	Zero Page	A5H	2	3	!	-	-	-	-	!	-
LDA aa, X	Zero Page, X	B5H	2	4	!	-	-	-	-	!	-
LDA aaaa	Absolute	ADH	3	4	!	-	-	-	-	!	-
LDA aaaa, X	Absolute, X	BDH	3	4	!	-	-	-	-	!	-
LDA aaaa, Y	Absolute, Y	B9H	3	4	!	-	-	-	-	!	-
LDA (aa, X)	(Indirect, X)	A1H	2	6	!	-	-	-	-	!	-
LDA (aa), Y	(Indirect), Y	B1H	2	6	!	-	-	-	-	!	-
LDX #dd	Immediate	A2H	2	2	!	-	-	-	-	!	-
LDX aa	Zero Page	A6H	2	3	!	-	-	-	-	!	-
LDX aa, Y	Zero Page, Y	B6H	2	4	!	-	-	-	-	!	-

LDX aaaa	Absolute	AEH	3	4	!	-	-	-	-	!	-
LDX aaaa, Y	Absolute, Y	BEH	3	4	!	-	-	-	-	!	-
LDY #dd	Immediate	A0H	2	2	!	-	-	-	-	!	-
LDY aa	Zero Page	A4H	2	3	!	-	-	-	-	!	-
LDY aa, X	Zero Page, X	B4H	2	4	!	-	-	-	-	!	-
LDY aaaa	Absolute	ACH	3	4	!	-	-	-	-	!	-
LDY aaaa, X	Absolute, X	BCH	3	4	!	-	-	-	-	!	-
LSR A	Accumulator	4AH	1	2	!	-	-	-	-	!	!
LSR aa	Zero Page	46H	2	5	!	-	-	-	-	!	!
LSR aa, X	Zero Page, X	56H	2	6	!	-	-	-	-	!	!
LSR aaaa	Absolute	4EH	3	6	!	-	-	-	-	!	!
LSR aaaa, X	Absolute, X	5EH	3	6	!	-	-	-	-	!	!
NOP	Implied	EAH	1	2	-	-	-	-	-	-	-
ORA #dd	Immediate	09H	2	2	!	-	-	-	-	!	-
ORA aa	Zero Page	05H	2	3	!	-	-	-	-	!	-
ORA aa, X	Zero Page, X	15H	2	4	!	-	-	-	-	!	-
ORA aaaa	Absolute	0DH	3	4	!	-	-	-	-	!	-
ORA aaaa, X	Absolute, X	1DH	3	4	!	-	-	-	-	!	-
ORA aaaa, Y	Absolute, Y	19H	3	4	!	-	-	-	-	!	-
ORA (aa, X)	(Indirect, X)	01H	2	6	!	-	-	-	-	!	-
ORA (aa), Y	(Indirect), Y	11H	2	6	!	-	-	-	-	!	-
PHA	Implied	48H	1	3	-	-	-	-	-	-	-
PHP	Implied	08H	1	3	-	-	-	-	-	-	-
PLA	Implied	68H	1	4	!	-	-	-	-	!	-
PLP	Implied	28H	1	4	!	!	-	!	!	!	!
ROL A	Accumulator	2AH	1	2	!	-	-	-	-	!	-
ROL aa	Zero Page	26H	2	5	!	-	-	-	-	!	-
ROL aa, X	Zero Page, X	36H	2	6	!	-	-	-	-	!	-
ROL aaaa	Absolute	2EH	3	6	!	-	-	-	-	!	-
ROL aaaa, X	Absolute, X	3EH	3	6	!	-	-	-	-	!	-
ROR A	Accumulator	6AH	1	2	!	-	-	-	-	!	-
ROR aa	Zero Page	66H	2	5	!	-	-	-	-	!	-
ROR aa, X	Zero Page, X	76H	2	6	!	-	-	-	-	!	-
ROR aaaa	Absolute	6EH	3	6	!	-	-	-	-	!	-
ROR aaaa, X	Absolute, X	7EH	3	6	!	-	-	-	-	!	-
RTI	Implied	40H	1	6	-	-	-	-	-	-	-
RTS	Implied	60H	1	6	-	-	-	-	-	-	-
SBC #dd	Immediate	E9H	2	2	!	!	-	*	-	!	!
SBC aa	Zero Page	E5H	2	3	!	!	-	*	-	!	!
SBC aa, X	Zero Page, X	F5H	2	4	!	!	-	*	-	!	!
SBC aaaa	Absolute	EDH	3	4	!	!	-	*	-	!	!
SBC aaaa, X	Absolute, X	FDH	3	4	!	!	-	*	-	!	!
SBC aaaa, Y	Absolute, Y	F9H	3	4	!	!	-	*	-	!	!
SBC (aa, X)	(Indirect, X)	E1H	2	6	!	!	-	*	-	!	!
SBC (aa), Y	(Indirect), Y	F1H	2	6	!	!	-	*	-	!	!
SEC	Implied	38H	1	2	-	-	-	-	-	-	!
SED	Implied	F8H	1	2	-	-	-	!	-	-	-
SEI	Implied	78H	1	2	-	-	-	-	!	-	-
SET aa, 0	Zero Page	8FH	2	5	-	-	-	-	-	-	-
SET aa, 1	Zero Page	9FH	2	5	-	-	-	-	-	-	-
SET aa, 2	Zero Page	AFH	2	5	-	-	-	-	-	-	-
SET aa, 3	Zero Page	BFH	2	5	-	-	-	-	-	-	-
SET aa, 4	Zero Page	CFH	2	5	-	-	-	-	-	-	-

SET aa, 5	Zero Page	DFH	2	5	-	-	-	-	-	-
SET aa, 6	Zero Page	EFH	2	5	-	-	-	-	-	-
SET aa, 7	Zero Page	FFH	2	5	-	-	-	-	-	-
STA aa	Zero Page	85H	2	3	-	-	-	-	-	-
STA aa, X	Zero Page, X	95H	2	4	-	-	-	-	-	-
STA aaaa	Absolute	8DH	3	4	-	-	-	-	-	-
STA aaaa, X	Absolute, X	9DH	3	4	-	-	-	-	-	-
STA aaaa, Y	Absolute, Y	99H	3	4	-	-	-	-	-	-
STA (aa, X)	(Indirect, X)	81H	2	6	-	-	-	-	-	-
STA (aa), Y	(Indirect), Y	91H	2	6	-	-	-	-	-	-
STX aa	Zero Page	86H	2	3	-	-	-	-	-	-
STX aa, Y	Zero Page, Y	96H	2	4	-	-	-	-	-	-
STX aaaa	Absolute	8EH	3	4	-	-	-	-	-	-
STY aa	Zero Page	84H	2	3	-	-	-	-	-	-
STY aa, X	Zero Page, X	94H	2	4	-	-	-	-	-	-
STY aaaa	Absolute	8CH	3	4	-	-	-	-	-	-
TAX	Implied	AAH	1	2	!	-	-	-	!	-
TAY	Implied	A8H	1	2	!	-	-	-	!	-
TST aa, 0	Zero Page	07H	2	3	-	-	-	-	!	-
TST aa, 1	Zero Page	17H	2	3	-	-	-	-	!	-
TST aa, 2	Zero Page	27H	2	3	-	-	-	-	!	-
TST aa, 3	Zero Page	37H	2	3	-	-	-	-	!	-
TST aa, 4	Zero Page	47H	2	3	-	-	-	-	!	-
TST aa, 5	Zero Page	57H	2	3	-	-	-	-	!	-
TST aa, 6	Zero Page	67H	2	3	-	-	-	-	!	-
TST aa, 7	Zero Page	77H	2	3	-	-	-	-	!	-
TSX	Implied	BAH	1	2	!	-	-	-	!	-
TXA	Implied	8AH	1	2	!	-	-	-	!	-
TXS	Implied	9AH	1	2	-	-	-	-	-	-
TYA	Implied	98H	1	2	!	-	-	-	!	-

25.5 Revision History

Revision	Date	Remark
V0.1	03/31/2005	First edition

26 Include File

26.1 Description

This chapter lists SPMC65X family hardware registers located from \$00 to \$5F. All of them are defined in 'SPMC65PxxxxA.inc' file in FortisIDE tool. Each body owned 'SPMC65PxxxxA.inc' file of itself to define its hardware resources. When user opens a new project and selects a suitable body in FortisIDE tool, the 'SPMC65PxxxxA.inc' file would be created in the project. In the content of 'SPMC65PxxxxA.inc' file, SUNPLUS not only defines normal hardware control registers but also defines constant labels for the purpose of coding style. Using C_XXX_XXX instead of numeral could increase readable abilities of program and maintain it easily. CB_XXX_XXX type is offered user for bit operation. For detailed information about both types using, please refer to example in FortisIDE tool.

26.2 Include File

```
;*****  
;; *** Copyright 2005 Sunplus Technology Co., Ltd. All right reserved. ***  
;; *** Header Name : ECMC653.inc ***  
;; *** Applied Body : ECMC653A ***  
;; *** Data : 2005/07/04 ***  
;; *** Revision : V1.0.0A ***  
;*****  
;  
;===== ;ECMC653 Input/Output Ports and Data Direction Registers ;=====;  
  
P_IOA_Data: EQU $00 ; Port A data b0~b7.(A)  
P_IOB_Data: EQU $01 ; Port B data b0~b7.(A)  
P_IOC_Data: EQU $02 ; Port C data b0~b7.(A)  
P_IOD_Data: EQU $03 ; Port D data b0~b7.(A)  
P_IOA_Dir: EQU $04 ; Port A direction control b0~b7.(W), 0=In, 1=Out  
P_IOB_Dir: EQU $05 ; Port B direction control b0~b7.(W)  
P_IOC_Dir: EQU $06 ; Port C direction control b0~b7.(W)  
P_IOD_Dir: EQU $07 ; Port D direction control b0~b7.(W)  
P_IOA_Attrib: EQU $08 ; Port A attribute register b0~b7.(W)  
P_IOB_Attrib: EQU $09 ; Port B attribute register b0~b7.(W)  
P_IOC_Attrib: EQU $0A ; Port C attribute register b0~b7.(W)  
P_IOD_Attrib: EQU $0B ; Port D attribute register b0~b7.(W)  
;
```

```

;-----  

P_INT_Flag0:      EQU    $0C          ; Interrupt Flag 0.(A)  

C_INT_ADIF:       EQU    %10000000  ; A/D INT flag bit.(A)  

C_INT_WDIF:       EQU    %01000000  ; WDT INT flag bit.(A)  

C_INT_IRQ5IF:     EQU    %00100000  ; IRQ5 INT flag bit.(A)  

C_INT_IRQ4IF:     EQU    %00010000  ; IRQ4 INT flag bit.(A)  

C_INT_IRQ3IF:     EQU    %00001000  ; IRQ3 INT flag bit.(A)  

C_INT_IRQ2IF:     EQU    %00000100  ; IRQ2 INT flag bit.(A)  

C_INT_IRQ1IF:     EQU    %00000010  ; IRQ1 INT flag bit.(A)  

C_INT_IRQ0IF:     EQU    %00000001  ; IRQ0 INT flag bit.(A)  

C_INT_CAP5IF:     EQU    %00100000  ; CAP5 INT flag bit.(A)  

C_INT_CAP4IF:     EQU    %00010000  ; CAP4 INT flag bit.(A)  

C_INT_CAP3IF:     EQU    %00000010  ; CAP3 INT flag bit.(A)  

C_INT_CAP2IF:     EQU    %00000001  ; CAP2 INT flag bit.(A)  

CB_INT_ADIF:      EQU    7           ; A/D INT flag bit for bit mode.(A)  

CB_INT_WDIF:       EQU    6           ; WDT INT flag bit for bit mode.(A)  

CB_INT_IRQ5IF:     EQU    5           ; IRQ5 INT flag bit for bit mode.(A)  

CB_INT_IRQ4IF:     EQU    4           ; IRQ4 INT flag bit for bit mode.(A)  

CB_INT_IRQ3IF:     EQU    3           ; IRQ3 INT flag bit for bit mode.(A)  

CB_INT_IRQ2IF:     EQU    2           ; IRQ2 INT flag bit for bit mode.(A)  

CB_INT_IRQ1IF:     EQU    1           ; IRQ1 INT flag bit for bit mode.(A)  

CB_INT_IRQ0IF:     EQU    0           ; IRQ0 INT flag bit for bit mode.(A)  

CB_INT_CAP5IF:     EQU    5           ; CAP5 INT flag bit for bit mode.(A)  

CB_INT_CAP4IF:     EQU    4           ; CAP4 INT flag bit for bit mode.(A)  

CB_INT_CAP3IF:     EQU    1           ; CAP3 INT flag bit for bit mode.(A)  

CB_INT_CAP2IF:     EQU    0           ; CAP2 INT flag bit for bit mode.(A)  

;  

P_INT_Ctrl0:       EQU    $0D        ; Interrupt control 0.(A)  

C_INT_ADIE:        EQU    %10000000  ; A/D INT enable bit.(A)  

C_INT_WDIE:        EQU    %01000000  ; WDT INT enable bit.(A)  

C_INT_IRQ5IE:      EQU    %00100000  ; IRQ5 INT enable bit.(A)  

C_INT_IRQ4IE:      EQU    %00010000  ; IRQ4 INT enable bit.(A)  

C_INT_IRQ3IE:      EQU    %00001000  ; IRQ3 INT enable bit.(A)  

C_INT_IRQ2IE:      EQU    %00000100  ; IRQ2 INT enable bit.(A)  

C_INT_IRQ1IE:      EQU    %00000010  ; IRQ1 INT enable bit.(A)  

C_INT_IRQ0IE:      EQU    %00000001  ; IRQ0 INT enable bit.(A)

```

C_INT_CAP5IE:	EQU	%00100000	; CAP5 INT enable bit.(A)
C_INT_CAP4IE:	EQU	%00010000	; CAP4 INT enable bit.(A)
C_INT_CAP3IE:	EQU	%00000010	; CAP3 INT enable bit.(A)
C_INT_CAP2IE:	EQU	%00000001	; CAP2 INT enable bit.(A)
CB_INT_ADIE:	EQU	7	; A/D INT enable bit for bit mode.(A)
CB_INT_WDIE:	EQU	6	; WDT INT enable bit for bit mode.(A)
CB_INT_IRQ5IE:	EQU	5	; IRQ5 INT enable bit for bit mode.(A)
CB_INT_IRQ4IE:	EQU	4	; IRQ4 INT enable bit for bit mode.(A)
CB_INT_IRQ3IE:	EQU	3	; IRQ3 INT enable bit for bit mode.(A)
CB_INT_IRQ2IE:	EQU	2	; IRQ2 INT enable bit for bit mode.(A)
CB_INT_IRQ1IE:	EQU	1	; IRQ1 INT enable bit for bit mode.(A)
CB_INT_IRQ0IE:	EQU	0	; IRQ0 INT enable bit for bit mode.(A)
CB_INT_CAP5IE:	EQU	5	; CAP5 INT enable bit for bit mode.(A)
CB_INT_CAP4IE:	EQU	4	; CAP4 INT enable bit for bit mode.(A)
CB_INT_CAP3IE:	EQU	1	; CAP3 INT enable bit for bit mode.(A)
CB_INT_CAP2IE:	EQU	0	; CAP2 INT enable bit for bit mode.(A)
 ;			
P_INT_Flag1:	EQU	\$0E	; Interrupt flag 1.
C_INT_CAP1IF:	EQU	%10000000	; CAP1 INT flag bit.(A)
C_INT_CAP0IF:	EQU	%01000000	; CAP0 INT flag bit.(A)
C_INT_T5OIF:	EQU	%00100000	; Timer5 overflow INT flag bit.(A)
C_INT_T4OIF:	EQU	%00010000	; Timer4 overflow INT flag bit.(A)
C_INT_T3OIF:	EQU	%00001000	; Timer3 overflow INT flag bit.(A)
C_INT_T2OIF:	EQU	%00000100	; Timer2 overflow INT flag bit.(A)
C_INT_T1OIF:	EQU	%00000010	; Timer1 overflow INT flag bit.(A)
C_INT_T0OIF:	EQU	%00000001	; Timer0 overflow INT flag bit.(A)
CB_INT_CAP1IF:	EQU	7	; CAP1 INT flag bit for bit mode.(A)
CB_INT_CAP0IF:	EQU	6	; CAP0 INT flag bit for bit mode.(A)
CB_INT_T5OIF:	EQU	5	; Timer5 overflow INT flag bit for bit mode.(A)
CB_INT_T4OIF:	EQU	4	; Timer4 overflow INT flag bit for bit mode.(A)
CB_INT_T3OIF:	EQU	3	; Timer3 overflow INT flag bit for bit mode.(A)
CB_INT_T2OIF:	EQU	2	; Timer2 overflow INT flag bit for bit mode.(A)
CB_INT_T1OIF:	EQU	1	; Timer1 overflow INT flag bit for bit mode.(A)
CB_INT_T0OIF:	EQU	0	; Timer0 overflow INT flag bit for bit mode.(A)
 ;			

P_INT_Ctrl1:	EQU	\$0F	; Interrupt control 1.
C_INT_CAP1IE:	EQU	%10000000	; CAP1 INT enable bit.(A)
C_INT_CAP0IE:	EQU	%01000000	; CAP0 INT enable bit.(A)
C_INT_T5OIE:	EQU	%00100000	; Timer5 overflow INT enable bit.(A)
C_INT_T4OIE:	EQU	%00010000	; Timer4 overflow INT enable bit.(A)
C_INT_T3OIE:	EQU	%00001000	; Timer3 overflow INT enable bit.(A)
C_INT_T2OIE:	EQU	%00000100	; Timer2 overflow INT enable bit.(A)
C_INT_T1OIE:	EQU	%00000010	; Timer1 overflow INT enable bit.(A)
C_INT_T0OIE:	EQU	%00000001	; Timer0 overflow INT enable bit.(A)
CB_INT_CAP1IE:	EQU	7	; CAP1 INT enable bit for bit mode.(A)
CB_INT_CAP0IE:	EQU	6	; CAP0 INT enable bit for bit mode.(A)
CB_INT_T5OIE:	EQU	5	; Timer5 overflow INT enable bit for bit mode.(A)
CB_INT_T4OIE:	EQU	4	; Timer4 overflow INT enable bit for bit mode.(A)
CB_INT_T3OIE:	EQU	3	; Timer3 overflow INT enable bit for bit mode.(A)
CB_INT_T2OIE:	EQU	2	; Timer2 overflow INT enable bit for bit mode.(A)
CB_INT_T1OIE:	EQU	1	; Timer1 overflow INT enable bit for bit mode.(A)
CB_INT_T0OIE:	EQU	0	; Timer0 overflow INT enable bit for bit mode.(A)
;			
P_INT_Flag2:	EQU	\$26	; Interrupt flag 2.
C_INT_ITVALIF:	EQU	%00100000	; Timer Base interrupt flag.
C_INT_IICIF:	EQU	%00010000	; IIC interrupt flag.
C_INT_UARTIF:	EQU	%00001000	; UART interrupt flag.
C_INT_SPIIF:	EQU	%00000100	; SPI interrupt flag.
C_INT_CMP1IF:	EQU	%00000010	; Comparator 1 interrupt flag.
C_INT_CMP0IF:	EQU	%00000001	; Comparator 0 interrupt flag.
CB_INT_ITVALIF:	EQU	5	; Timer Base interrupt flag for bit mode.
CB_INT_IICIF:	EQU	4	; IIC interrupt flag for bit mode.
CB_INT_UARTIF:	EQU	3	; UART interrupt flag for bit mode.
CB_INT_SPIIF:	EQU	2	; SPI interrupt flag for bit mode.
CB_INT_CMP1IF:	EQU	1	; Comparator 1 interrupt flag for bit mode.
CB_INT_CMP0IF:	EQU	0	; Comparator 0 interrupt flag for bit mode.
;			
P_INT_Ctrl2:	EQU	\$27	; Interrupt control 2.
C_INT_ITVALIE:	EQU	%00100000	; Timer Base interrupt enable bit.

```

C_INT_CMP1IE:      EQU    %00000010 ; Comparator 1 interrupt enable bit.
C_INT_CMP0IE:      EQU    %00000001 ; Comparator 0 interrupt enable bit.

CB_INT_ITVALIE:   EQU    5          ; Timer Base interrupt enable bit for bit mode.
CB_INT_CMP1IE:     EQU    1          ; Comparator 1 interrupt enable bit for bit mode.
CB_INT_CMP0IE:     EQU    0          ; Comparator 0 interrupt enable bit for bit mode.
;

P_WDT_Clr:         EQU    $10        ; Watchdog clear register.(W), $55= clear
C_WDT_Clr:         EQU    $55        ; Write '55' to clear this register.
;

-----

P_TMR0_1_Ctrl0:    EQU    $11        ; Timer0/1 control 0.
C_T112B_PWM:       EQU    %01110000 ; Timer1 Function as 12 Bit PWM.
C_T116B_CAP:        EQU    %01100000 ; Timer1 Function as 16 Bit Capture(Width).
C_T116B_COMP:       EQU    %01010000 ; Timer1 Function as 16 Bit Compare.
C_T116B_Timer:      EQU    %01000000 ; Timer1 Function as 16 Bit Timer.
C_T18B_CAP:         EQU    %00110000 ; Timer1 Function as 8Bit Capture(Width,Cycle).
C_T18B_COMP:        EQU    %00100000 ; Timer1 Function as 8Bit Compare.
C_T18B_Timer:       EQU    %00010000 ; Timer1 Function as 8 Bit Timer.
C_T08B_PWM:         EQU    %00000111 ; Timer0 Function as 8 Bit PWM.
C_T016B_CAP:        EQU    %00000110 ; Timer0 Function as 16 Bit Capture(Width).
C_T016B_COMP:       EQU    %00000101 ; Timer0 Function as 16 Bit Compare.
C_T016B_Timer:      EQU    %00000100 ; Timer0 Function as 16 Bit Timer.
C_T08B_CAP:         EQU    %00000011 ; Timer0 Function as 8 Bit Capture(Width,Cycle).
C_T08B_COMP:        EQU    %00000010 ; Timer0 Function as 8Bit Compare.
C_T08B_Timer:       EQU    %00000001 ; Timer0 Function as 8 Bit Timer.

P_TMR0_1_Ctrl1:    EQU    $12        ; Timer0/1 control 1.
C_T1EXT_EN:         EQU    %01110000 ; External Event
C_T1FCS_Div_512:   EQU    %01100000 ; Timer1 Clock= FCS/512.
C_T1FCS_Div_128:   EQU    %01010000 ; Timer1 Clock= FCS/128.
C_T1FCS_Div_32:    EQU    %01000000 ; Timer1 Clock= FCS/32.
C_T1FCS_Div_8:     EQU    %00110000 ; Timer1 Clock= FCS/8.
C_T1FCS_Div_4:     EQU    %00100000 ; Timer1 Clock= FCS/4.
C_T1FCS_Div_2:     EQU    %00010000 ; Timer1 Clock= FCS/2.
C_T1FCS_Div_1:     EQU    %00000000 ; Timer1 Clock= FCS/1.
C_T0EXT_EN:         EQU    %00000111 ; External Event

```

C_T0FCS_Div_512:	EQU	%00000110	; Timer0 Clock= FCS/512.
C_T0FCS_Div_128:	EQU	%00000101	; Timer0 Clock= FCS/128.
C_T0FCS_Div_32:	EQU	%00000100	; Timer0 Clock= FCS/32.
C_T0FCS_Div_8:	EQU	%00000011	; Timer0 Clock= FCS/8.
C_T0FCS_Div_4:	EQU	%00000010	; Timer0 Clock= FCS/4.
C_T0FCS_Div_2:	EQU	%00000001	; Timer0 Clock= FCS/2.
C_T0FCS_Div_1:	EQU	%00000000	; Timer0 Clock= FCS/1.
P_TMR0_Count:	EQU	\$13	; Timer0 8/16-bit counter register.(R)
P_TMR0_Preload:	EQU	\$13	; Timer0 8/16-bit preload register.(W)
P_TMR0_Comp:	EQU	\$13	; Timer0 8/16-bit compare low byte value.
P_TMR0_Cap:	EQU	\$13	; Timer0 8/16-bit capture low byte width value.
P_TMR0_PWMPeriod:	EQU	\$13	; Timer0 8 bit PWM period value.
;			
P_TMR0_CountHi:	EQU	\$14	; Timer0 16-bit counter register.(R)
P_TMR0_PreloadHi:	EQU	\$14	; Timer0 16-bit preload register.(W)
P_TMR0_CompHi:	EQU	\$14	; Timer0 16-bit compare high byte value.
P_TMR0_CapHi:	EQU	\$14	; Timer0 16-bit capture high byte width value.
P_TMR0_CapCycle8:	EQU	\$14	; Timer0 8-bit capture cycle value.(R)
P_TMR0_PWMduty:	EQU	\$14	; Timer0 8-bit PWM duty value.
;			
P_TMR1_Count:	EQU	\$15	; Timer1 8/16-bit Counter register.(R)
P_TMR1_Preload:	EQU	\$15	; Timer1 8/16-bit preload register.(W)
P_TMR1_Comp:	EQU	\$15	; Timer1 8/16-bit compare low byte value.
P_TMR1_Cap:	EQU	\$15	; Timer1 8/16-bit capture low byte width value.
P_TMR1_PWMPeriod:	EQU	\$15	; Timer1 12-bit PWM peroid low byte register.
;			
P_TMR1_CountHi:	EQU	\$16	; Timer1 16-bit Counter register.(R)
P_TMR1_PreloadHi:	EQU	\$16	; Timer1 16-bit preload register.(W)
P_TMR1_CompHi:	EQU	\$16	; Timer1 16-bit compare high byte value.
P_TMR1_CapHi:	EQU	\$16	; Timer1 16-bit capture high byte width value.
P_TMR1_CapCycle8:	EQU	\$16	; Timer1 8-bit capture cycle value.(R)
P_TMR1_DutyPeriod:	EQU	\$16	; Timer1 12-bit PWM high byte register.
;			
P_TMR1_PWMduty:	EQU	\$17	; Timer1 12-bit PWM duty low byte register.
;			
P_TMR2_3_Ctrl0:	EQU	\$18	; Timer2/3 control 0.

C_T312B_PWM:	EQU	%01110000	; Timer3 Function as 12 Bit PWM.
C_T316B_CAP:	EQU	%01100000	; Timer3 Function as 16 Bit Capture(Width).
C_T316B_COMP:	EQU	%01010000	; Timer3 Function as 16 Bit Compare.
C_T316B_Timer:	EQU	%01000000	; Timer3 Function as 16 Bit Timer.
C_T38B_CAP:	EQU	%00110000	; Timer3 Function as 8Bit Capture(Width,Cycle).
C_T38B_COMP:	EQU	%00100000	; Timer3 Function as 8Bit Compare.
C_T38B_Timer:	EQU	%00010000	; Timer3 Function as 8 Bit Timer.
C_T28B_PWM:	EQU	%00000111	; Timer2 Function as 8 Bit PWM.
C_T216B_CAP:	EQU	%00000110	; Timer2 Function as 16 Bit Capture(Width).
C_T216B_COMP:	EQU	%00000101	; Timer2 Function as 16 Bit Compare.
C_T216B_Timer:	EQU	%00000100	; Timer2 Function as 16 Bit Timer.
C_T28B_CAP:	EQU	%00000011	; Timer2 Function as 8 Bit Capture(Width,Cycle).
C_T28B_COMP:	EQU	%00000010	; Timer2 Function as 8Bit Compare.
C_T28B_Timer:	EQU	%00000001	; Timer2 Function as 8 Bit Timer.
;			
P_TMR2_3_Ctrl1:	EQU	\$19	; Timer2/3 control 1.
C_T3EXT_EN:	EQU	%01110000	; External Event.
C_T3FCS_Div_512:	EQU	%01100000	; Timer3 Clock= FCS/512.
C_T3FCS_Div_128:	EQU	%01010000	; Timer3 Clock= FCS/128.
C_T3FCS_Div_32:	EQU	%01000000	; Timer3 Clock= FCS/32.
C_T3FCS_Div_8:	EQU	%00110000	; Timer3 Clock= FCS/8.
C_T3FCS_Div_4:	EQU	%00100000	; Timer3 Clock= FCS/4.
C_T3FCS_Div_2:	EQU	%00010000	; Timer3 Clock= FCS/2.
C_T3FCS_Div_1:	EQU	%00000000	; Timer3 Clock= FCS/1.
C_T2EXT_EN:	EQU	%00000111	; External Event.
C_T2FCS_Div_512:	EQU	%00000110	; Timer2 Clock= FCS/512.
C_T2FCS_Div_128:	EQU	%00000101	; Timer2 Clock= FCS/128.
C_T2FCS_Div_32:	EQU	%00000100	; Timer2 Clock= FCS/32.
C_T2FCS_Div_8:	EQU	%00000011	; Timer2 Clock= FCS/8.
C_T2FCS_Div_4:	EQU	%00000010	; Timer2 Clock= FCS/4.
C_T2FCS_Div_2:	EQU	%00000001	; Timer2 Clock= FCS/2.
C_T2FCS_Div_1:	EQU	%00000000	; Timer2 Clock= FCS/1.
;			
P_TMR2_Count:	EQU	\$1A	; Timer2 8/16-bit counter register.(R)
P_TMR2_Preload:	EQU	\$1A	; Timer2 8/16-bit preload register.(W)
P_TMR2_Comp:	EQU	\$1A	; Timer2 8/16-bit compare low byte value.
P_TMR2_Cap:	EQU	\$1A	; Timer2 8/16-bit capture low byte width value.

P_TMR2_PWMPeriod:	EQU	\$1A	; Timer2 8-bit PWM period value.
;			
P_TMR2_CountHi:	EQU	\$1B	; Timer2 16-bit counter register.(R)
P_TMR2_PreloadHi:	EQU	\$1B	; Timer2 16-bit preload register.(W)
P_TMR2_CompHi:	EQU	\$1B	; Timer2 16-bit compare high byte value.
P_TMR2_CapHi:	EQU	\$1B	; Timer2 16-bit capture high byte width value.
P_TMR2_CapCycle8:	EQU	\$1B	; Timer2 8-bit capture cycle value.(R)
P_TMR2_PWMduty:	EQU	\$1B	; Timer2 8-bit PWM duty value.
;			
P_TMR3_Count:	EQU	\$1C	; Timer3 8/16-bit Counter register.(R)
P_TMR3_Preload:	EQU	\$1C	; Timer3 8/16-bit preload register.(W)
P_TMR3_Comp:	EQU	\$1C	; Timer3 8/16-bit compare low byte value.
P_TMR3_Cap:	EQU	\$1C	; Timer3 8-bit capture low byte width value.
P_TMR3_PWMPeriod:	EQU	\$1C	; Timer3 12-bit PWM peroid low byte register.
;			
P_TMR3_CountHi:	EQU	\$1D	; Timer3 16-bit Counter register.(R)
P_TMR3_PreloadHi:	EQU	\$1D	; Timer3 16-bit preload register.(W)
P_TMR3_CompHi:	EQU	\$1D	; Timer3 16-bit compare high byte value.
P_TMR3_CapHi:	EQU	\$1D	; Timer3 16-bit capture high byte width value.
P_TMR3_CapCycle8:	EQU	\$1D	; Timer3 8-bit capture cycle value.(R)
P_TMR3_DutyPeriod:	EQU	\$1D	; Timer3 12-bit PWM high byte register.
;			
P_TMR3_PWMduty:	EQU	\$1E	; Timer3 12-bit PWM duty low byte register.
;			
P_TMR4_5_Ctrl0:	EQU	\$1F	; Timer4/5 control 0.
C_T516B_PWM:	EQU	%01110000	; Timer5 Function as 16 Bit PWM.
C_T516B_CAP:	EQU	%01100000	; Timer5 Function as 16 Bit Capture(Width,Cycle).
C_T516B_COMP:	EQU	%01010000	; Timer5 Function as 16 Bit Compare.
C_T516B_Timer:	EQU	%01000000	; Timer5 Function as 16 Bit Timer.
C_T58B_CAP:	EQU	%00110000	; Timer5 Function as 8Bit Capture(Width,Cycle).
C_T58B_COMP:	EQU	%00100000	; Timer5 Function as 8Bit Compare.
C_T58B_Timer:	EQU	%00010000	; Timer5 Function as 8 Bit Timer.
C_T48B_PWM:	EQU	%00000111	; Timer4 Function as 8 Bit PWM.
C_T416B_CAP:	EQU	%00000110	; Timer4 Function as 16 Bit Capture(Width).
C_T416B_COMP:	EQU	%00000101	; Timer4 Function as 16 Bit Compare.
C_T416B_Timer:	EQU	%00000100	; Timer4 Function as 16 Bit Timer.
C_T48B_CAP:	EQU	%00000011	; Timer4 Function as 8 Bit Capture(Width,Cycle).

C_T48B_COMP:	EQU	%00000010	; Timer4 Function as 8Bit Compare.
C_T48B_Timer:	EQU	%00000001	; Timer4 Function as 8 Bit Timer.
;			
P_TMR4_5_Ctrl1:	EQU	\$20	; Timer4/5 control 1.
C_T5EXT_EN:	EQU	%01110000	; External Event.
C_T5FCS_Div_512:	EQU	%01100000	; Timer5 Clock= FCS/512.
C_T5FCS_Div_128:	EQU	%01010000	; Timer5 Clock= FCS/128.
C_T5FCS_Div_32:	EQU	%01000000	; Timer5 Clock= FCS/32.
C_T5FCS_Div_8:	EQU	%00110000	; Timer5 Clock= FCS/8.
C_T5FCS_Div_4:	EQU	%00100000	; Timer5 Clock= FCS/4.
C_T5FCS_Div_2:	EQU	%00010000	; Timer5 Clock= FCS/2.
C_T5FCS_Div_1:	EQU	%00000000	; Timer5 Clock= FCS/1.
C_T4EXT_EN:	EQU	%00000111	; External Event.
C_T4FCS_Div_512:	EQU	%00000110	; Timer4 Clock= FCS/512.
C_T4FCS_Div_128:	EQU	%00000101	; Timer4 Clock= FCS/128.
C_T4FCS_Div_32:	EQU	%00000100	; Timer4 Clock= FCS/32.
C_T4FCS_Div_8:	EQU	%00000011	; Timer4 Clock= FCS/8.
C_T4FCS_Div_4:	EQU	%00000010	; Timer4 Clock= FCS/4.
C_T4FCS_Div_2:	EQU	%00000001	; Timer4 Clock= FCS/2.
C_T4FCS_Div_1:	EQU	%00000000	; Timer4 Clock= FCS/1.
;			
P_TMR4_Count:	EQU	\$21	; Timer4 8/16-bit counter register.(R)
P_TMR4_Preload:	EQU	\$21	; Timer4 8/16-bit preload register.(W)
P_TMR4_Comp:	EQU	\$21	; Timer4 8/16-bit compare low byte value.
P_TMR4_Cap:	EQU	\$21	; Timer4 8/16-bit capture low byte width value.
P_TMR4_PWMPeriod:	EQU	\$21	; Timer4 8-bit PWM period value.
;			
P_TMR4_CountHi:	EQU	\$22	; Timer4 16-bit counter register.(R)
P_TMR4_PreloadHi:	EQU	\$22	; Timer4 16-bit preload register.(W)
P_TMR4_CompHi:	EQU	\$22	; Timer4 16-bit compare high byte value.
P_TMR4_CapHi:	EQU	\$22	; Timer4 16-bit capture high byte width value.
P_TMR4_CapCycle8:	EQU	\$22	; Timer4 8-bit capture cycle value.(R)
P_TMR4_PWMduty:	EQU	\$22	; Timer4 8-bit PWM duty value.
;			
P_TMR5_Count:	EQU	\$23	; Timer5 8/16-bit Counter register.(R)
P_TMR5_Preload:	EQU	\$23	; Timer5 8/16-bit preload register.(W)
P_TMR5_Comp:	EQU	\$23	; Timer5 8/16-bit compare low byte value.

P_TMR5_Cap:	EQU	\$23	; Timer5 8/16-bit capture low byte width value.
P_TMR5_PWMPeriod:	EQU	\$23	; Timer5 16-bit PWM peroid low byte register.
;			
P_TMR5_CountHi:	EQU	\$24	; Timer5 16-bit Counter register.(R)
P_TMR5_PreloadHi:	EQU	\$24	; Timer5 16-bit preload register.(W)
P_TMR5_CompHi:	EQU	\$24	; Timer5 16-bit compare high byte value.
P_TMR5_CapHi:	EQU	\$24	; Timer5 16-bit capture high byte width value.
P_TMR5_CapCycle8:	EQU	\$24	; Timer5 8-bit capture cycle value.(R)
P_TMR5_DutyPeriod:	EQU	\$24	; Timer5 16-bit PWM period high byte register.
;			
P_TMR5_CapCycleLo:	EQU	\$25	; Timer5 16-bit capture cycle high byte value.(R)
P_TMR5_PWMduty:	EQU	\$25	; Timer5 16-bit PWM duty low byte register.
;			
P_TMR5_CapCycleHi:	EQU	\$5F	; Timer5 16-bit capture cycle high byte value.(R)
P_TMR5_PWMdutyHi:	EQU	\$5F	; Timer5 16-bit PWM duty high byte register.
;			
<hr/>			
P_AD_Ctrl0:	EQU	\$28	; A/D converter control 0.
C_AD_EN:	EQU	%10000000	; ADC enable control.(A)
C_AD_VRT:	EQU	%01000000	; ADC bottom voltage source select bit.(A)
C_AD_CS_2:	EQU	%00000000	; 000=Fcpu/2.
C_AD_CS_4:	EQU	%00000010	; 001=Fcpu/4.
C_AD_CS_8:	EQU	%00000100	; 010=Fcpu/8.
C_AD_CS_16:	EQU	%00000110	; 011=Fcpu/16.
C_AD_CS_32:	EQU	%00001000	; 100=Fcpu/32.
C_AD_CS_64:	EQU	%00001010	; 101=Fcpu/64.
C_AD_CS_128:	EQU	%00001100	; 110=Fcpu/128.
C_AD_CS_256:	EQU	%00001110	; 111=Fcpu/256.
C_AD_RDY:	EQU	%00000001	; ADC status bit.(R)
C_AD_Start:	EQU	%00000000	; ADC conversion start bit.(W)
;			
CB_AD_EN:	EQU	7	; ADC enable control for bit mode.(A)
CB_AD_VRT:	EQU	6	; ADC top voltage source select bit for bit mode.(A)
CB_AD_RDY:	EQU	0	; ADC status bit for bit mode.(R)
CB_AD_Start:	EQU	0	; ADC conversion start bit for bit mode.(W)
;			
P_AD_Ctrl1:	EQU	\$29	; A/D converter control 1.

```

C_AD_Pin0:      EQU    %00000001 ; Analog input Port Configuration: channel 0.
C_AD_Pin1:      EQU    %00000010 ; Analog input Port Configuration: channel 1.
C_AD_Pin2:      EQU    %00000100 ; Analog input Port Configuration: channel 2.
C_AD_Pin3:      EQU    %00001000 ; Analog input Port Configuration: channel 3.
C_AD_Pin4:      EQU    %00010000 ; Analog input Port Configuration: channel 4.
C_AD_Pin5:      EQU    %00100000 ; Analog input Port Configuration: channel 5.
C_AD_Pin6:      EQU    %01000000 ; Analog input Port Configuration: channel 6.
C_AD_Pin7:      EQU    %10000000 ; Analog input Port Configuration: channel 7.

CB_AD_Pin0:     EQU    0      ; Analog input Configuration: channel 0 for bit mode.
CB_AD_Pin1:     EQU    1      ; Analog input Configuration: channel 1 for bit mode.
CB_AD_Pin2:     EQU    2      ; Analog input Configuration: channel 2 for bit mode.
CB_AD_Pin3:     EQU    3      ; Analog input Configuration: channel 3 for bit mode.
CB_AD_Pin4:     EQU    4      ; Analog input Configuration: channel 4 for bit mode.
CB_AD_Pin5:     EQU    5      ; Analog input Configuration: channel 5 for bit mode.
CB_AD_Pin6:     EQU    6      ; Analog input Configuration: channel 6 for bit mode.
CB_AD_Pin7:     EQU    7      ; Analog input Configuration: channel 7 for bit mode.

;

P_AD_Ctrl2:     EQU    $2A   ; A/D converter control 2.
C_AD_CE:        EQU    %10000000 ; ADC power control bit.(A)
C_AD_Ch0:        EQU    %00000000 ; 0000:channel 0.
C_AD_Ch1:        EQU    %00001000 ; 0001:channel 1.
C_AD_Ch2:        EQU    %00010000 ; 0010:channel 2.
C_AD_Ch3:        EQU    %00011000 ; 0011:channel 3.
C_AD_Ch4:        EQU    %00100000 ; 0100:channel 4.
C_AD_Ch5:        EQU    %00101000 ; 0101:channel 5.
C_AD_Ch6:        EQU    %00110000 ; 0110:channel 6.
C_AD_Ch7:        EQU    %00111000 ; 0111:channel 7.
C_AD_Ch8:        EQU    %01000000 ; 1000:channel 8.
C_AD_Pin8:       EQU    %00000001 ; Analog input : channel 8.

CB_AD_CE:        EQU    7      ; ADC power control bit for bit mode.(A)

;

P_AD_DataHi:    EQU    $2B   ; Converted A/D data[9:2] hi.(R)
P_AD_DataLo:    EQU    $2C   ; Converted A/D data[1:0] low.(R)

;
;
```

P_BUZ_Ctrl:	EQU	\$2D	; Buzzer & Timer base Control. ; Interval Timer Period selection bits.
C_TBASE_Dis:	EQU	%00000000	; Time Base disable
C_TBASE_Div_128:	EQU	%00010000	; Time Base Clk: Fto/2^7.
C_TBASE_Div_256:	EQU	%00100000	; Time Base Clk: Fto/2^8.
C_TBASE_Div_512:	EQU	%00110000	; Time Base Clk: Fto/2^9.
C_TBASE_Div_1k:	EQU	%01000000	; Time Base Clk: Fto/2^10.
C_TBASE_Div_2k:	EQU	%01010000	; Time Base Clk: Fto/2^11.
C_TBASE_Div_4k:	EQU	%01100000	; Time Base Clk: Fto/2^12.
C_TBASE_Div_8k:	EQU	%01110000	; Time Base Clk: Fto/2^13.
C_TBASE_Div_16k:	EQU	%10000000	; Time Base Clk: Fto/2^14.
C_TBASE_Div_32k:	EQU	%10010000	; Time Base Clk: Fto/2^15.
C_TBASE_Div_64k:	EQU	%10100000	; Time Base Clk: Fto/2^16.
C_TBASE_Div_128k:	EQU	%10110000	; Time Base Clk: Fto/2^17.
C_TBASE_Div_256k:	EQU	%11000000	; Time Base Clk: Fto/2^18.
C_TBASE_Div_512k:	EQU	%11010000	; Time Base Clk: Fto/2^19.
C_TBASE_Div_2M:	EQU	%11100000	; Time Base Clk: Fto/2^21.
C_TBASE_Div_8M:	EQU	%11110000	; Time Base Clk: Fto/2^23. ; Buzzer frequency selection bits.
C_BUZ_Dis:	EQU	%00000000	; Buzzer disable
C_BUZ_Div_64:	EQU	%00000001	; Buzzer:Fto/2^6.
C_BUZ_Div_128:	EQU	%00000010	; Buzzer:Fto/2^7.
C_BUZ_Div_256:	EQU	%00000011	; Buzzer:Fto/2^8.
C_BUZ_Div_512:	EQU	%00000100	; Buzzer:Fto/2^9.
C_BUZ_Div_1k:	EQU	%00000101	; Buzzer:Fto/2^10.
C_BUZ_Div_2k:	EQU	%00000110	; Buzzer:Fto/2^11.
C_BUZ_Div_4k:	EQU	%00000111	; Buzzer:Fto/2^12.
C_BUZ_Div_8k:	EQU	%00001000	; Buzzer:Fto/2^13.
C_BUZ_Div_4:	EQU	%00001001	; Buzzer:Fto/2^2.
C_BUZ_Div_8:	EQU	%00001010	; Buzzer:Fto/2^3.
C_BUZ_Div_16:	EQU	%00001011	; Buzzer:Fto/2^4.
C_BUZ_Div_32:	EQU	%00001100	; Buzzer:Fto/2^5.
;			

P_CMP_Ctrl:	EQU	\$2E	; Comparator control register.
C_CMP1_EN:	EQU	%10000000	; comparator1 function enable.
C_CMP1_RS:	EQU	%01000000	; comparator1 reference source selection.

```

C_CMP1_LOG:      EQU    %00100000 ; comparator1 logic event direction selection.
C_CMP1_OUT:      EQU    %00010000 ; comparator1 result bit.
C_CMP0_EN:       EQU    %00001000 ; comparator0 function enable.
C_CMP0_RS:       EQU    %00000100 ; comparator0 reference source selection.
C_CMP0_LOG:      EQU    %00000010 ; comparator0 logic event direction selection.
C_CMP0_OUT:      EQU    %00000001 ; comparator0 result bit.

CB_CMP1_EN:      EQU    7          ; comparator1 function enable for bit mode.
CB_CMP1_RS:      EQU    6          ; comparator1 reference
                   ; source selection for bit mode.
CB_CMP1_LOG:     EQU    5          ; comparator1 logic event direction
                   ; selection for bit mode.
CB_CMP1_OUT:     EQU    4          ; comparator1 result bit for bit mode.
CB_CMP0_EN:      EQU    3          ; comparator0 function enable for bit mode.
CB_CMP0_RS:      EQU    2          ; comparator0 reference
                   ; source selection for bit mode.
CB_CMP0_LOG:     EQU    1          ; comparator0 logic event
                   ; direction selection for bit mode.
CB_CMP0_OUT:     EQU    0          ; comparator0 result bit for bit mode.

;

;-----;
; Double Write Register

P_SYS_Ctrl:      EQU    $30        ; System control.
C_SCR_POR:       EQU    %10000000 ; Power On Reset Flag.(A)
C_SCR_ERST:      EQU    %01000000 ; External Reset Flag.(A)
C_SCR_LVR:       EQU    %00100000 ; Low Voltage Reset Flag.(A)
C_SCR_WDTR:      EQU    %00001000 ; WDT Reset Flag.(A)
C_SCR_IAR:       EQU    %00000100 ; Illegal Address Reset Flag.(A)
C_SCR_IIR:       EQU    %00000001 ; Illegal instruction reset(A)

CB_SCR_POR:      EQU    7          ; Power On Reset Flag for bit mode.(A)
CB_SCR_ERST:      EQU    6          ; External Reset Flag for bit mode.(A)
CB_SCR_LVR:       EQU    5          ; Low Voltage Reset Flag for bit mode.(A)
CB_SCR_WDTR:      EQU    3          ; WDT Reset Flag for bit mode.(A)
CB_SCR_IAR:       EQU    2          ; Illegal Address Reset Flag for bit mode.(A)
CB_SCR_IIR:       EQU    0          ; Illegal instruction reset for bit mode.(A)

;

```

P_MODE_Ctrl:	EQU	\$31	; Operation mode control register.(W)
C_MODE_STOP:	EQU	\$5A	; Enter STOP mode.(W)
C_MODE_HALT:	EQU	\$A5	; Enter HALT mode.(W)
C_MODE_Reset:	EQU	\$66	; Reset all of internal modules except CPU.(W)
;			
P_WDT_Ctrl:	EQU	\$32	; Watchdog control register.
C_WDT_SCKEN:	EQU	%10000000	; Slow Clock enable bit in stop mode.(A) ; Selection bits of watchdog interrupt rate.(A)
C_WDT_RTO:	EQU	%00000000	; RTO from Timer0 is selected.
C_WDT_Div_256:	EQU	%00010000	; WDI clock = /256.
C_WDT_Div_512:	EQU	%00100000	; WDI clock = /512.
C_WDT_Div_1024:	EQU	%00110000	; WDI clock = /1024.
C_WDT_Div_2048:	EQU	%01000000	; WDI clock = /2048.
C_WDT_Div_4096:	EQU	%01010000	; WDI clock = /4096.
C_WDT_Div_8192:	EQU	%01100000	; WDI clock = /8192.
C_WDT_Div_16384:	EQU	%01110000	; WDI clock = /16384.
CB_WDT_EN:	EQU	7	; Watchdog enable bit in stop mode for bit mode.(A)
;			
P_IRQ_Opt0:	EQU	\$33	; IRQ Option 0 register.
C IRQOpt0_IRQ5ES:	EQU	%00001000	; Polarity control of INT5.(A)
C IRQOpt0_IRQM5:	EQU	%00000100	; INT5 trigger mode selection.(A)
C IRQOpt0_IRQ4ES:	EQU	%00000010	; Polarity control of INT4.(A)
C IRQOpt0_IRQM4:	EQU	%00000001	; INT4 trigger mode selection.(A)
C IRQOpt0_CAP5ES:	EQU	%00001000	; Polarity control of CAP5.(A)
C IRQOpt0_CAP4ES:	EQU	%00000010	; Polarity control of CAP4.(A)
CB IRQOpt0_IRQ5ES:	EQU	3	; Polarity control of INT5 for bit mode.(A)
CB IRQOpt0_IRQM5:	EQU	2	; INT5 trigger mode selection for bit mode.(A)
CB IRQOpt0_IRQ4ES:	EQU	1	; Polarity control of INT4 for bit mode.(A)
CB IRQOpt0_IRQM4:	EQU	0	; INT4 trigger mode selection for bit mode.(A)
CB IRQOpt0_CAP5ES:	EQU	3	; Polarity control of CAP5 for bit mode.(A)
CB IRQOpt0_CAP4ES:	EQU	1	; Polarity control of CAP4 for bit mode.(A)
;			
P_IRQ_Opt1:	EQU	\$34	; IRQ Option 1 register.
C IRQOpt1_IRQ3ES:	EQU	%10000000	; Polarity control of INT3.(A)
C IRQOpt1_IRQM3:	EQU	%01000000	; INT3 trigger mode selection.(A)

C IRQOpt1_IRQ2ES:	EQU	%00100000	; Polarity control of INT2.(A)
C IRQOpt1_IRQM2:	EQU	%00010000	; INT2 trigger mode selection.(A)
C IRQOpt1_IRQ1ES:	EQU	%00001000	; Polarity control of INT1.(A)
C IRQOpt1_IRQM1:	EQU	%00000100	; INT1 trigger mode selection.(A)
C IRQOpt1_IRQ0ES:	EQU	%00000010	; Polarity control of INT0.(A)
C IRQOpt1_IRQM0:	EQU	%00000001	; INT0 trigger mode selection.(A)
C IRQOpt1_CAP3ES:	EQU	%00001000	; Polarity control of CAP3.(A)
C IRQOpt1_CAP2ES:	EQU	%00000010	; Polarity control of CAP2.(A)
 CB IRQOpt1_IRQ3ES:	EQU	7	; Polarity control of INT3 for bit mode.(A)
CB IRQOpt1_IRQM3:	EQU	6	; INT3 trigger mode selection for bit mode.(A)
CB IRQOpt1_IRQ2ES:	EQU	5	; Polarity control of INT2 for bit mode.(A)
CB IRQOpt1_IRQM2:	EQU	4	; INT2 trigger mode selection for bit mode.(A)
CB IRQOpt1_IRQ1ES:	EQU	3	; Polarity control of INT1 for bit mode.(A)
CB IRQOpt1_IRQM1:	EQU	2	; INT1 trigger mode selection for bit mode.(A)
CB IRQOpt1_IRQ0ES:	EQU	1	; Polarity control of INT0 for bit mode.(A)
CB IRQOpt1_IRQM0:	EQU	0	; INT0 trigger mode selection for bit mode.(A)
CB IRQOpt1_CAP3ES:	EQU	3	; Polarity control of CAP3 for bit mode.(A)
CB IRQOpt1_CAP2ES:	EQU	1	; Polarity control of CAP2 for bit mode.(A)
 ;			
P_IO_Opt:	EQU	\$35	; I/O slew rate control register.
C_IO_SLOWE:	EQU	%00000001	; PB[7:6] slew rate enable selection.(A)
 CB_IO_SLOWE:	EQU	0	; PB[7:6] slew rate enable selection for bit mode.(A)
 ;			
P_LVR_Opt:	EQU	\$36	; LVR Option
C_LVR_V40:	EQU	%00000001	; LVR level select bit.(A)
 CB_LVR_V40:	EQU	0	; LVR level select bit for bit mode.(A)
 ;			

P_SPI_Ctrl0:	EQU	\$38	; SPI control register 0.
C_SPI_EN:	EQU	%10000000	; enable Control bit.
C_SPI_MOD:	EQU	%01000000	; operation Mode Master/Slave Mode.
C_SPI_SCKPHA:	EQU	%00100000	; clock phase.
C_SPI_SCKPOL:	EQU	%00010000	; clock polarity.
C_SPI_SPISMPS:	EQU	%00001000	; sample mode selection bit for master mode.

C_SPICS_Div_128:	EQU	%00000101	; CPU Clock Selection/128.
C_SPICS_Div_64:	EQU	%00000100	; CPU clock Selection/64.
C_SPICS_Div_32:	EQU	%00000011	; CPU clock Selection/32.
C_SPICS_Div_16:	EQU	%00000010	; CPU clock Selection/16.
C_SPICS_Div_8:	EQU	%00000001	; CPU clock Selection/8.
C_SPICS_Div_4:	EQU	%00000000	; CPU clock Selection/4.
CB_SPI_EN:	EQU	7	; enable Control bit for bit mode.
CB_SPI_MOD:	EQU	6	; operation Mode Master/Slave Mode for bit mode.
CB_SPI_SCKPHA:	EQU	5	; clock phase for bit mode.
CB_SPI_SCKPOL:	EQU	4	; clock polarity for bit mode.
CB_SPI_SPISMPS:	EQU	3	; sample mode selection bit for master mode for bit mode.
;			
P_SPI_Ctrl1:	EQU	\$39	; SPI control register 1.
C_SPI_SMSEN:	EQU	%10000000	; SPI Slave Mode Selection enable bit.
C_SPI_SWRST:	EQU	%01000000	; software reset bit.
C_SPISPC_Div_4:	EQU	%00000011	; sampling clock Div/4.
C_SPISPC_Div_2:	EQU	%00000010	; sampling clock Div/2.
C_SPISPC_Div_1:	EQU	%00000001	; sampling clock Div/1.
C_SPISPC_Dis:	EQU	%00000000	; no sampling.
CB_SPI_SMSEN:	EQU	7	; SPI Slave Mode Selection enable bit for bit mode.
CB_SPI_SWRST:	EQU	6	; software reset bit for bit mode.
;			
P_SPI_Status:	EQU	\$3A	; SPI status register.
C_SPI_INTIF:	EQU	%10000000	; SPI interrupt flag active.
C_SPI_INTEN:	EQU	%01000000	; SPI interrupt enable/disable.
C_SPI_TXBF:	EQU	%00100000	; transmission buffer full flag.
C_SPI_BUFFull:	EQU	%00000001	; buffer full and overwrite.
CB_SPI_INTIF:	EQU	7	; SPI interrupt flag active for bit mode.
CB_SPI_INTEN:	EQU	6	; SPI interrupt enable/disable for bit mode.
CB_SPI_TXBF:	EQU	5	; transmission buffer full flag for bit mode.
CB_SPI_BUFFull:	EQU	0	; buffer full and overwrite for bit mode.
;			
P_SPI_TxData:	EQU	\$3B	; SPI Transmit data buffer.

P_SPI_RxData:	EQU	\$3C	; SPI Receive data buffer.
;			

P_IOE_Data:	EQU	\$40	; Port E data b0~b7.(A)
P_IOF_Data:	EQU	\$41	; Port F data b0~b7.(A)
P_IOE_Dir:	EQU	\$42	; Port E direction control b0~b7.(W)
P_IOF_Dir:	EQU	\$43	; Port F direction control b0~b7.(W)
P_IOE_Attrib:	EQU	\$44	; Port E attribute register b0~b7.(W)
P_IOF_Attrib:	EQU	\$45	; Port F attribute register b0~b7.(W)
;			

P_UART_Ctrl:	EQU	\$46	; UART control register.
C_UART_RXIE:	EQU	%10000000	; receive interrupt enable bit.
C_UART_TXIE:	EQU	%01000000	; transmit interrupt enable bit.
C_UART_RXEN:	EQU	%00100000	; receive function enable bit.
C_UART_TXEN:	EQU	%00010000	; transmit function enable bit.
C_UART_SOFRST:	EQU	%00001000	; software reset.
C_UART_STOPSEL:	EQU	%00000100	; stop bit length selection bit.
C_UART_PSEL:	EQU	%00000010	; parity type selection bit.
C_UART_PEN:	EQU	%00000001	; parity check or generation enable bit.
;			
CB_UART_RXIE:	EQU	7	; receive interrupt enable bit for bit mode.
CB_UART_TXIE:	EQU	6	; transmit interrupt enable bit for bit mode.
CB_UART_RXEN:	EQU	5	; receive function enable bit for bit mode.
CB_UART_TXEN:	EQU	4	; transmit function enable bit for bit mode.
CB_UART_SOFRST:	EQU	3	; software reset for bit mode.
CB_UART_STOPSEL:	EQU	2	; stop bit length selection bit for bit mode.
CB_UART_PSEL:	EQU	1	; parity type selection bit for bit mode.
CB_UART_PEN:	EQU	0	; parity check or generation enable bit for bit mode.
;			
P_UART_Baud:	EQU	\$47	; UART baud rate divisor.
;			
P_UART_Status:	EQU	\$48	; UART status register.
C_UART_RXIF:	EQU	%10000000	; receive interrupt flag.
C_UART_TXIF:	EQU	%01000000	; transmit interrupt flag.
C_UART_BUSY:	EQU	%00100000	; transmission in progress flag bit.
C_UART_OERR:	EQU	%00000100	; overrun error flag bit.

```

C_UART_PERR:      EQU    %00000010 ; parity error flag bit.
C_UART_FERR:      EQU    %00000001 ; frame error flag bit.

CB_UART_RXIF:     EQU    7          ; receive interrupt flag for bit mode.
CB_UART_TXIF:     EQU    6          ; transmit interrupt flag for bit mode.
CB_UART_BUSY:      EQU    5          ; transmission in progress flag bit for bit mode.
CB_UART_OERR:      EQU    2          ; overrun error flag bit for bit mode.
CB_UART_PERR:      EQU    1          ; parity error flag bit for bit mode.
CB_UART_FERR:      EQU    0          ; frame error flag bit for bit mode.

;

P_UART_Data:      EQU    $49        ; UART data register.

;

;-----



P_IIC_Ctrl:        EQU    $4A        ; IIC series control register.
C_IIC_IICEN:       EQU    %10000000 ; IIC series interface enable bit.
C_IIC_TXRXEN:      EQU    %00100000 ; IIC bus data output enable bit.
C_IIC_INTEN:       EQU    %00010000 ; IIC interface interrupt enable bit.
C_IIC_ACKEN:       EQU    %00001000 ; IIC acknowledge enable bit.
C_IIC_SCK_1024:    EQU    %00000111 ; interface clock rate. Fcs/1024.
C_IIC_SCK_512:     EQU    %00000110 ; interface clock rate. Fcs/512.
C_IIC_SCK_256:     EQU    %00000101 ; interface clock rate. Fcs/256.
C_IIC_SCK_128:     EQU    %00000100 ; interface clock rate. Fcs/128.
C_IIC_SCK_64:      EQU    %00000011 ; interface clock rate. Fcs/64.
C_IIC_SCK_32:      EQU    %00000010 ; interface clock rate. Fcs/32.
C_IIC_SCK_16:      EQU    %00000001 ; interface clock rate. Fcs/16.
C_IIC_SCK_8:       EQU    %00000001 ; interface clock rate. Fcs/8.

CB_IIC_IICEN:      EQU    7          ; IIC series interface enable bit for bit mode.
CB_IIC_TXRXEN:     EQU    5          ; IIC bus data output enable bit for bit mode.
CB_IIC_INTEN:      EQU    4          ; IIC interface interrupt enable bit for bit mode.
CB_IIC_ACKEN:      EQU    3          ; IIC acknowledge enable bit for bit mode.

;

P_IIC_Status:      EQU    $4B        ; IIC series status register.
C_IIC_MXT:         EQU    %10000000 ; IIC master/slave mode selection.
C_IIC_TXRXSEL:     EQU    %01000000 ; Tx/Rx selection bit.
C_IIC_BUSY:         EQU    %00100000 ; bus busy signal.
C_IIC_SIGGEN:      EQU    %00100000 ; IIC start/stop signal generation.

```

C_IIC_INTIF:	EQU	%00010000	; IIC series interrupt flag bit.
C_IIC_ARBITRAT:	EQU	%00001000	; IIC bus arbitration status.
C_IIC_AAS:	EQU	%00000100	; I2C bus address-as-slave status flag data foramt bit0.
C_IIC_AZERO:	EQU	%00000010	; IIC address zero status flag.
C_IIC_LRB:	EQU	%00000001	; IIC last-received bit status flag.
CB_IIC_MXT:	EQU	7	; IIC master/slave mode selection for bit mode.
CB_IIC_TXRXSEL:	EQU	6	; Tx/Rx selection bit for bit mode.
CB_IIC_BUSY:	EQU	5	; bus busy signal for bit mode.
CB_IIC_SIGGEN:	EQU	5	; IIC start/stop signal generation for bit mode.
CB_IIC_INTIF:	EQU	4	; IIC series interrupt flag bit for bit mode.
CB_IIC_ARBITRAT:	EQU	3	; IIC bus arbitration status for bit mode.
CB_IIC_AAS:	EQU	2	; I2C bus address-as-slave status flag data foramt bit0 for bit mode.
CB_IIC_AZERO:	EQU	1	; IIC address zero status flag for bit mode.
CB_IIC_LRB:	EQU	0	; IIC last-received bit status flag for bit mode.
;			
P_IIC_Data:	EQU	\$4C	; IIC series data register.
P_IIC_Address:	EQU	\$4D	; IIC series address bits.
;			

P_DA_Ctrl:	EQU	\$55	; DA converter control register.
C_DA_EN:	EQU	%10000000	; DA converter enable bit.
;			
CB_DA_EN:	EQU	7	; DA converter enable bit for bit mode.
;			
P_DA_DataLo:	EQU	\$56	; Converted D/A data[1:0] low.(W)
P_DA_DataHi:	EQU	\$57	; Converted D/A data[9:2] hi.(W)
;			

P_CAP_Ctrl:	EQU	\$58	; Capture control.
C_CAP_OPT:	EQU	%10000000	; Capture option control bit.(A)
C_CAP_IP4:	EQU	%01000000	; CAP4 interrupt evoke polarity.
C_CAP_IP3:	EQU	%00100000	; CAP3 interrupt evoke polarity.
C_CAP_IP2:	EQU	%00010000	; CAP2 interrupt evoke polarity.
C_CAP_IP1:	EQU	%00001000	; CAP1 interrupt evoke polarity.
C_CAP_IP0:	EQU	%00000100	; CAP0 interrupt evoke polarity.

```

C_CAP1_ES:      EQU    %00000010 ; Polarity control of capture1 interrupt.
C_CAP0_ES:      EQU    %00000001 ; Polarity control of capture0 interrupt.

CB_CAP_OPT:     EQU    7          ; Capture option control bit for bit mode.(A)
CB_CAP_IP4:     EQU    6          ; CAP4 interrupt evoke polarity for bit mode.
CB_CAP_IP3:     EQU    5          ; CAP3 interrupt evoke polarity for bit mode.
CB_CAP_IP2:     EQU    4          ; CAP2 interrupt evoke polarity for bit mode.
CB_CAP_IP1:     EQU    3          ; CAP1 interrupt evoke polarity for bit mode.
CB_CAP_IP0:     EQU    2          ; CAP0 interrupt evoke polarity for bit mode.
CB_CAP1_ES:     EQU    1          ; Polarity control of capture1 interrupt for bit mode.
CB_CAP0_ES:     EQU    0          ; Polarity control of capture0 interrupt for bit mode.

;

;-----
```

P_IOA_Buf:	EQU	\$59	
C_IOA_Buf7:	EQU	%10000000	; Output data Latch of PORTA bit7.
C_IOA_Buf6:	EQU	%01000000	; Output data Latch of PORTA bit6.
C_IOA_Buf5:	EQU	%00100000	; Output data Latch of PORTA bit5.
C_IOA_Buf4:	EQU	%00010000	; Output data Latch of PORTA bit4.
C_IOA_Buf3:	EQU	%00001000	; Output data Latch of PORTA bit3.
C_IOA_Buf2:	EQU	%00000100	; Output data Latch of PORTA bit2.
C_IOA_Buf1:	EQU	%00000010	; Output data Latch of PORTA bit1.
C_IOA_Buf0:	EQU	%00000001	; Output data Latch of PORTA bit0.
P_IOB_Buf:	EQU	\$5A	
C_IOB_Buf7:	EQU	%10000000	; Output data Latch of PORTB bit7.
C_IOB_Buf6:	EQU	%01000000	; Output data Latch of PORTB bit6.
C_IOB_Buf5:	EQU	%00100000	; Output data Latch of PORTB bit5.
C_IOB_Buf4:	EQU	%00010000	; Output data Latch of PORTB bit4.
C_IOB_Buf3:	EQU	%00001000	; Output data Latch of PORTB bit3.
C_IOB_Buf2:	EQU	%00000100	; Output data Latch of PORTB bit2.
C_IOB_Buf1:	EQU	%00000010	; Output data Latch of PORTB bit1.
C_IOB_Buf0:	EQU	%00000001	; Output data Latch of PORTB bit0.
P_IOC_Buf:	EQU	\$5B	
C_IOC_Buf7:	EQU	%10000000	; Output data Latch of PORTC bit7.
C_IOC_Buf6:	EQU	%01000000	; Output data Latch of PORTC bit6.
C_IOC_Buf5:	EQU	%00100000	; Output data Latch of PORTC bit5.
C_IOC_Buf4:	EQU	%00010000	; Output data Latch of PORTC bit4.
C_IOC_Buf3:	EQU	%00001000	; Output data Latch of PORTC bit3.

C_IOC_Buf2:	EQU	%00000100	; Output data Latch of PORTC bit2.
C_IOC_Buf1:	EQU	%00000010	; Output data Latch of PORTC bit1.
C_IOC_Buf0:	EQU	%00000001	; Output data Latch of PORTC bit0.
P_IOD_Buf:	EQU	\$5C	
C_IOD_Buf7:	EQU	%10000000	; Output data Latch of PORTD bit7.
C_IOD_Buf6:	EQU	%01000000	; Output data Latch of PORTD bit6.
C_IOD_Buf5:	EQU	%00100000	; Output data Latch of PORTD bit5.
C_IOD_Buf4:	EQU	%00010000	; Output data Latch of PORTD bit4.
C_IOD_Buf3:	EQU	%00001000	; Output data Latch of PORTD bit3.
C_IOD_Buf2:	EQU	%00000100	; Output data Latch of PORTD bit2.
C_IOD_Buf1:	EQU	%00000010	; Output data Latch of PORTD bit1.
C_IOD_Buf0:	EQU	%00000001	; Output data Latch of PORTD bit0.
P_IOE_Buf:	EQU	\$5D	
C_IOE_Buf7:	EQU	%10000000	; Output data Latch of PORTE bit7.
C_IOE_Buf6:	EQU	%01000000	; Output data Latch of PORTE bit6.
C_IOE_Buf5:	EQU	%00100000	; Output data Latch of PORTE bit5.
C_IOE_Buf4:	EQU	%00010000	; Output data Latch of PORTE bit4.
C_IOE_Buf3:	EQU	%00001000	; Output data Latch of PORTE bit3.
C_IOE_Buf2:	EQU	%00000100	; Output data Latch of PORTE bit2.
C_IOE_Buf1:	EQU	%00000010	; Output data Latch of PORTE bit1.
C_IOE_Buf0:	EQU	%00000001	; Output data Latch of PORTE bit0.
P_IOF_Buf:	EQU	\$5E	
C_IOF_Buf7:	EQU	%10000000	; Output data Latch of PORTF bit7.
C_IOF_Buf6:	EQU	%01000000	; Output data Latch of PORTF bit6.
C_IOF_Buf5:	EQU	%00100000	; Output data Latch of PORTF bit5.
C_IOF_Buf4:	EQU	%00010000	; Output data Latch of PORTF bit4.
C_IOF_Buf3:	EQU	%00001000	; Output data Latch of PORTF bit3.
C_IOF_Buf2:	EQU	%00000100	; Output data Latch of PORTF bit2.
C_IOF_Buf1:	EQU	%00000010	; Output data Latch of PORTF bit1.
C_IOF_Buf0:	EQU	%00000001	; Output data Latch of PORTF bit0.
C_RAM_ADDR:	EQU	\$60	; RAM Start Address.
C_STACK_BOTTOM:	EQU	\$0FF	; Stack Bottom Address.
CB_Bit7	EQU	7	; for bit mode
CB_Bit6	EQU	6	;
CB_Bit5	EQU	5	;
CB_Bit4	EQU	4	;

CB_Bit3	EQU	3	;
CB_Bit2	EQU	2	;
CB_Bit1	EQU	1	;
CB_Bit0	EQU	0	;

26.3 Revision History

Revision	Date	Remark
V1.0	03/31/2005	First edition
V1.1	01/05/2005	Second edition