



PMC234/PMS234 Series

12-bit ADC Enhanced Field Programmable Processor Array (FPPA™) 8-bit Controller

Data Sheet

Version 0.02 – Oct. 30, 2015

Copyright © 2015 by PADAUK Technology Co., Ltd., all rights reserved

10F-2, No. 1, Sec. 2, Dong-Da Road, Hsin-Chu 300, Taiwan, R.O.C.

TEL: 886-3-532-7598  www.padauk.com.tw



PMC234/PMS234 Series 12-bit ADC Enhanced FPPA™ 8-bit Controller

IMPORTANT NOTICE

PADAUK Technology reserves the right to make changes to its products or to terminate production of its products at any time without notice. Customers are strongly recommended to contact PADAUK Technology for the latest information and verify whether the information is correct and complete before placing orders.

PADAUK Technology products are not warranted to be suitable for use in life-support applications or other critical applications. PADAUK Technology assumes no liability for such applications. Critical applications include, but are not limited to, those which may involve potential risks of death, personal injury, fire or severe property damage.

PADAUK Technology assumes no responsibility for any issue caused by a customer's product design. Customers should design and verify their products within the ranges guaranteed by PADAUK Technology. In order to minimize the risks in customers' products, customers should design a product with adequate operating safeguards.

Table of Contents

1. Features	9
1-1. Special Features	9
1-2. High Performance RISC CPU Array	9
1-3. System Functions	9
1-4. Package Information	10
2. General Description and Block Diagram	11
3. Pin Assignment and Description	12
4. Device Characteristics	17
4-1. AC/DC Device Characteristics.....	17
4-2. Absolute Maximum Ratings	19
4-3. Typical ILRC frequency vs. VDD and temperature.....	20
4-4. Typical IHRC frequency deviation vs. VDD and temperature.....	20
4-5. Typical operating current vs. VDD @ system clock = ILRC/n	21
4-6. Typical operating current vs. VDD @ system clock = IHRC/n.....	21
4-7. Typical operating current vs. VDD @ system clock = 1MHz EOSC / n	22
4-8. Typical operating current vs. VDD @ system clock = 4MHz EOSC / n	22
4-9. Typical operating current vs. VDD @ system clock = 32KHz EOSC / n.....	23
4-10. Typical IO driving current (I_{OH}) and sink current (I_{OL}).....	23
4-11. Typical IO input high/low threshold voltage (V_{IH}/V_{IL}).....	24
4-12. Typical resistance of IO pull high device	24
4-13. Typical VDD/2 Bias output voltage	25
4-14. Timing charts for boot up conditions.....	25
4-15. Typical Comparator Responsive Time (Use $V_{internal R}$).....	26
4-16. Typical Comparator Responsive Time (Use External Inputs).....	27
5. Functional Description	28
5-1. Processing Units.....	28
5-1-1. Program Counter	29
5-1-2. Stack Pointer	29
5-1-3. Single FPP mode.....	30
5-2. Program Memory -- OTP	31
5-2-1. Program Memory Assignment	31
5-2-2. Example of Using Program Memory for Two FPP mode.....	32
5-2-3. Example of Using Program Memory for Single FPP mode.....	32
5-3. Program Structure	33

5-3-1. Program structure of two FPP units mode.....	33
5-3-2. Program structure of single FPP mode	33
5-4. Boot Procedure.....	34
5-5. Data Memory -- SRAM	35
5-6. Arithmetic and Logic Unit.....	35
5-7. Oscillator and clock	36
5-7-1. Internal High RC oscillator and Internal Low RC oscillator	36
5-7-2. Chip calibration.....	36
5-7-3. IHRC Frequency Calibration and System Clock.....	36
5-7-4. External Crystal Oscillator	38
5-7-5. System Clock and LVR level	39
5-7-6. System Clock Switching	40
5-8. 16-bit Timer (Timer16).....	41
5-9. 8-bit Timer (Timer2) with PWM generation.....	43
5-9-1. Using the Timer2 to generate periodical waveform	44
5-9-2. Using the Timer2 to generate 8-bit PWM waveform.....	46
5-9-3. Using the Timer2 to generate 6-bit PWM waveform.....	47
5-10. WatchDog Timer.....	48
5-11. Interrupt	49
5-12. Power-Save and Power-Down.....	51
5-12-1. Power-Save mode (“stopexe”).....	51
5-12-2. Power-Down mode (“stopsys”)	52
5-12-3. Wake-up	53
5-13. IO Pins.....	54
5-14. Reset and LVR	55
5-14-1. Reset	55
5-14-2. LVR (low voltage reset)	55
5-15. Comparator.....	56
5-15-1. Comparator Hardware Diagram	56
5-15-2. Analog Inputs.....	57
5-15-3. Internal reference voltage ($V_{\text{internal R}}$)	57
5-15-4. Comparator Interrupt Operation	60
5-15-5. Synchronizing Comparator Output to Timer2	60
5-15-6. Comparator Response Time	61
5-15-7. Using the comparator	61
5-15-8. Using the comparator and band-gap 1.20V.....	62
5-16. LCD Bias Voltage Generator	62
5-17. Analog-to-Digital Conversion (ADC) module	63
5-17-1. The input requirement for AD conversion.....	64

5-17-2. Select the ADC bit resolution.....	65
5-17-3. ADC clock selection.....	65
5-17-4. AD conversion	65
5-17-5. Configure the analog pins.....	65
5-17-6. Using the ADC.....	66
6. IO Registers	67
6-1. ACC Status Flag Register (<i>flag</i>), IO address = 0x00.....	67
6-2. FPP unit Enable Register (<i>fpen</i>), IO address = 0x01	67
6-3. Stack Pointer Register (<i>sp</i>), IO address = 0x02	67
6-4. Clock Mode Register (<i>clkmd</i>), IO address = 0x03	68
6-5. Interrupt Enable Register (<i>inten</i>), IO address = 0x04	69
6-6. Interrupt Request Register (<i>intrq</i>), IO address = 0x05.....	69
6-7. Timer16 mode Register (<i>t16m</i>), IO address = 0x06	70
6-8. General Data register for IO (<i>gdio</i>), IO address = 0x07.....	70
6-9. External Oscillator setting Register (<i>eoscr</i>), IO address = 0x0a	71
6-10. Internal High RC oscillator control Register (<i>ihrcr</i>), IO address = 0x0b	71
6-11. Interrupt Edge Select Register (<i>integs</i>), IO address = 0x0c	71
6-12. Port A Digital Input Enable Register (<i>padier</i>), IO address = 0x0d	72
6-13. Port B Digital Input Enable Register (<i>pbdiar</i>), IO address = 0x0e	73
6-14. Port A Data Register (<i>pa</i>), IO address = 0x10.....	73
6-15. Port A Control Register (<i>pac</i>), IO address = 0x11	73
6-16. Port A Pull-High Register (<i>paph</i>), IO address = 0x12.....	73
6-17. Port B Data Register (<i>pb</i>), IO address = 0x14.....	74
6-18. Port B Control Register (<i>pbc</i>), IO address = 0x15	74
6-19. Port B Pull-High Register (<i>pbph</i>), IO address = 0x16.....	74
6-20. Port C Data Register (<i>pc</i>), IO address = 0x17.....	74
6-21. Port C Control Register (<i>pcc</i>), IO address = 0x18.....	74
6-22. Port C Pull-High Register (<i>pcph</i>), IO address = 0x19.....	74
6-23. Port D Data Register (<i>pd</i>), IO address = 0x1a.....	74
6-24. Port D Control Register (<i>pdc</i>), IO address = 0x1b.....	75
6-25. Port D Pull-High Register (<i>pdph</i>), IO address = 0x1c.....	75
6-26. ADC Control Register (<i>adcc</i>), IO address = 0x20.....	75
6-27. ADC Mode Register (<i>adcm</i>), IO address = 0x21	76
6-28. ADC Result High Register (<i>adcrh</i>), IO address = 0x22	76
6-29. ADC Result Low Register (<i>adclr</i>), IO address = 0x23	76
6-30. Miscellaneous Register (<i>misc</i>), IO address = 0x3b	77
6-31. Timer2 Control Register (<i>tm2c</i>), IO address = 0x3c.....	78
6-32. Timer2 Counter Register (<i>tm2ct</i>), IO address = 0x3d.....	78

6-33. Timer2 Scalar Register (tm2s), IO address = 0x37	78
6-34. Timer2 Bound Register (tm2b), IO address = 0x09.....	79
6-35. Comparator Control Register (gpcc), IO address = 0x3e	79
6-36. Comparator Selection Register (gps), IO address = 0x22	79
7. Instructions	80
7-1. Data Transfer Instructions	80
7-2. Arithmetic Operation Instructions.....	84
7-3. Shift Operation Instructions	87
7-4. Logic Operation Instructions.....	88
7-5. Bit Operation Instructions	91
7-6. Conditional Operation Instructions	92
7-7. System control Instructions	94
7-8. Summary of Instructions Execution Cycle	96
7-9. Summary of affected flags by Instructions.....	97
8. Special Notes	98
8-1. Using IC.....	98
8-1-1. IO pin usage and setting	98
8-1-2. Interrupt.....	99
8-1-3. System clock switching	99
8-1-4. Power down mode, wakeup and watchdog.....	100
8-1-5. TIMER time out	101
8-1-6. Using ADC	101
8-1-7. LVR	101
8-1-8. Differences in command timing between single / double FPPA mode.....	102
8-1-9. Programming 28pin package	102
8-2. Using ICE.....	103
8-2-1. Emulating PMC234/PMS234 series IC on ICE PDK3S-I-001/002/003	103
8-2-2. Important Notice for ICE operation.....	104



PMC234/PMS234 Series 12-bit ADC Enhanced FPPA™ 8-bit Controller

Revision History:

Revision	Date	Description
0.01	2015/8/1	1st version
0.02	2015/10/30	Add 1-4. Package Information: PMC234-Y24 Add 3. PMC234-Y24 Pin Assignment and Description

Major Differences between P234C and PMC234/PMS234

There are many differences between P234C and PMC234/PMS234. The table below only shows the major differences between them.

Item	Function	P234C	PMC234/PMS234
1	IO Pin	22 IO pins PA[7:0], PB[7:0], PC[5:0]	26 IO pins PA[7:0], PB[7:0], PC[7:0], PD[1:0]
2	IO capability	12mA@5.0V	10mA@5.0V
3	SRAM	200 bytes	208 bytes
4	Band-gap	+/- 200mV(@1.20V)	+/- 60mV(@1.20V) after calibration
5	LVR	4 levels LVR setting	8 levels LVR setting
6	Single FPPA mode	N/A	Yes
7	ADC reference high voltage	VDD and PB1	VDD
8	ADC resolution	8bit to 12bit selectable	Only 12bit available.
9	Port digital/analog input configure registers	<i>padidr , pbdidr, pcdidr</i>	<i>padier , pbdier</i>
10	IHRC option command	.ADJUST_OTP_IHRCR	.ADJUST_IC
11	WDT time out	512 ILRC clock cycles	4 selectable periods

Procedure for converting code from P234C to PMC234/PMS234

Please follow the below steps for converting code from P234C to PMC234/PMS234:

1. Go through the PMC234/PMS234 datasheet and user guide;
2. Modify the source code engineering file “.pre”; change “.chip P234CXXX” to “.chip PMC234” or “.chip PMS234”
3. Press “Build” and then IDE will show some errors and warnings.
4. Modify the source code correspondingly until all errors have been solved.
5. Save and build the project files again.
6. Write to a real chip and test its functions in detail.
7. Back to the step 3 if necessary.
8. Contact our FAE at fae@padauk.com.tw if you still have any problems.

1. Features

1-1. Special Features

- ◆ PMC234 series:
 - ◇ High EFT series
 - ◇ Operating temperature range: -40°C ~ 85°C
- ◆ PMS234 series:
 - ◇ General purpose series
 - ◇ Please don't apply to AC RC step-down powered, high power ripple or high EFT requirement application
 - ◇ Operating temperature range: -20°C ~ 70°C

1-2. High Performance RISC CPU Array

- ◆ Operating modes: Two processing units FPPA™ mode or Traditional one processing unit mode
- ◆ 4Kx16 bits OTP program memory for both FPP units
- ◆ 208 Bytes data RAM for both FPP units
- ◆ 100 powerful instructions
- ◆ Most instructions are 1T execution cycle
- ◆ Programmable stack pointer to provide adjustable stack level
- ◆ Direct and indirect addressing modes for data and instructions
- ◆ All data memories are available for use as an index pointer
- ◆ Separated IO and memory space

1-3. System Functions

- ◆ Clock sources: internal high RC oscillator, internal low RC oscillator and external crystal oscillator
- ◆ Band-gap circuit to provide 1.20V reference voltage
- ◆ One hardware 16-bit timer
- ◆ One hardware 8-bit timer with PWM generator
- ◆ Up to 11-channel 12-bit resolution ADC with 1-channel for internal band-gap reference voltage
- ◆ Provide software configurable LCD driver IO with optional VDD/2 LCD bias voltage
- ◆ Provide maximum 4x21 dots LCD display
- ◆ Support fast wake-up
- ◆ Eight levels of LVR ~ 4.1V, 3.6V, 3.1V, 2.8V, 2.5V, 2.2V, 2.0V, 1.8V
- ◆ Provide one hardware comparator
- ◆ 26 IO pins with 10mA driving / sink capability
- ◆ Two external interrupt pins
- ◆ Every IO pin can be configured to enable wake-up function
- ◆ Operating voltage range: 2.2V ~ 5.5V
- ◆ Operating frequency and voltage for both crystal mode and IHRC mode

DC ~ 8MHz@VDD ≥ 4V	DC ~ 4MHz@VDD ≥ 3V
DC ~ 2MHz@VDD ≥ 2.5V	DC ~ 1MHz@VDD ≥ 2.2V
- ◆ Low power consumption

I _{operating} ~ 1.7mA@1MIPS, VDD=5.0V;	I _{operating} ~ 15uA@VDD=3.3V, ILRC ~ 12KHz
I _{powerdown} ~ 1uA@VDD=5.0V;	I _{powerdown} ~ 0.5uA@VDD=3.3V

1-4. Package Information

◆ PMC234 series

PMC234 - Y24: SSOP24 (150mil);
PMC234 - S20: SOP20 (300mil);
PMC234 - S24: SOP24 (300mil);
PMC234 - S28: SOP28 (300mil);
PMC234 - D20: DIP20 (300mil);
PMC234 - K24: Skinny DIP24 (300mil);
PMC234 - K28: Skinny DIP28 (300mil);

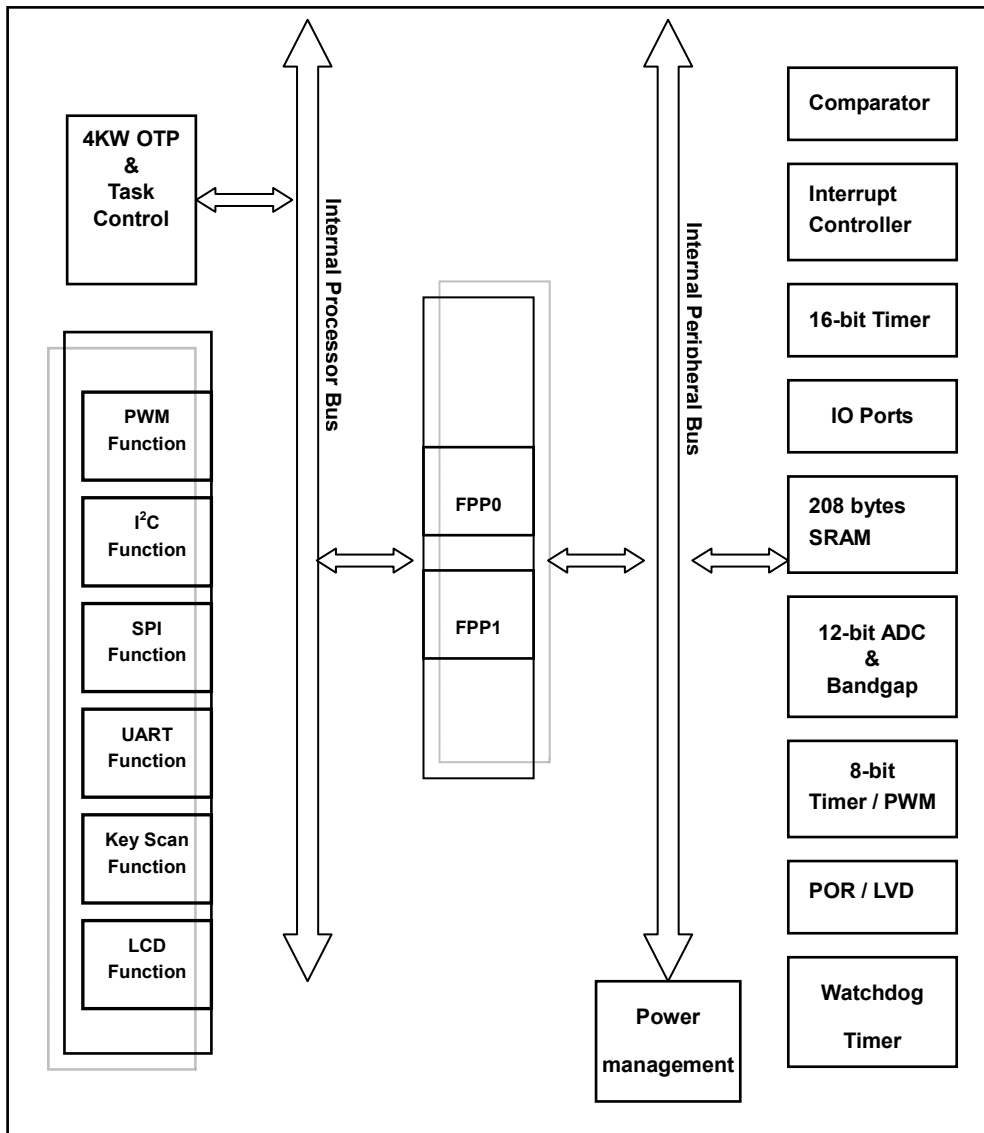
◆ PMS234 series

PMS234 - S20: SOP20 (300mil);
PMS234 - S24: SOP24 (300mil);
PMS234 - S28: SOP28 (300mil);
PMS234 - D20: DIP20 (300mil);
PMS234 - K24: Skinny DIP24 (300mil);
PMS234 - K28: Skinny DIP28 (300mil);

2. General Description and Block Diagram

The PMC234/PMS234 series is an ADC-Type of PADAUK's parallel processing, fully static, OTP-based CMOS 2x8 bit processor array that can execute two peripheral functions in parallel. It employs RISC architecture based on patent pending FPPA™ (Field Programmable Processor Array) technology and all the instructions are executed in one cycle except that some instructions are two cycles that handle indirect memory access.

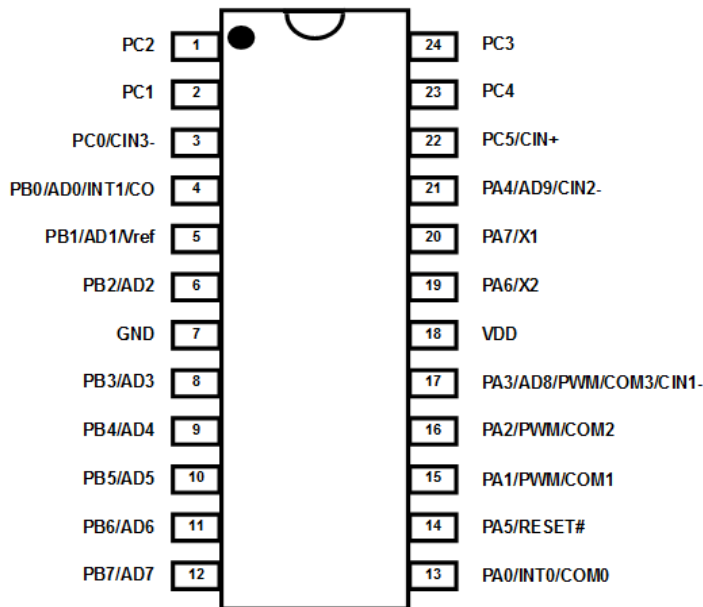
4Kx16 bits OTP program memory and 208 bytes data SRAM are inside for two FPP units using, one up to 11 channels 12-bit ADC is built inside the chip with one channel for internal band-gap reference voltage; one comparator is also provided. PMC234/PMS234 also provides two hardware timers: one is 16-bit timer and the other one is 8-bit with PWM generation.



3. Pin Assignment and Description

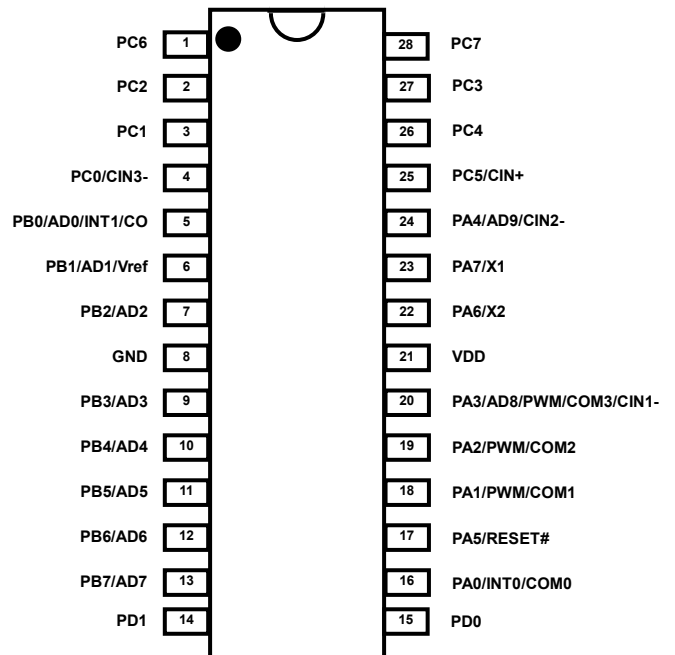
◆ PMC234 series

PMC234-Y24 (SSOP24-150mil)
PMC234-S24 (SOP24-300mil)
PMC234-K24 (Skinny DIP24-300mil)



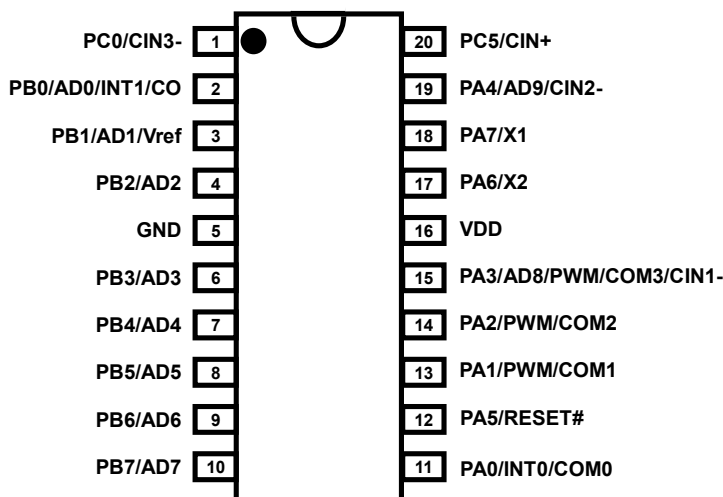
PMC234-Y24 (SSOP24-150mil)
 PMC234-S24 (SOP24-300mil)
 PMC234-K24 (Skinny DIP24-300mil)

PMC234-S28 (SOP28-300mil)
PMC234-K28 (Skinny DIP28-300mil)



PMC234-S28 (SOP28-300mil)
 PMC234-K28 (Skinny DIP-300mil)

PMC234-S20 (SOP20-300mil)
PMC234-D20 (DIP20-300mil)

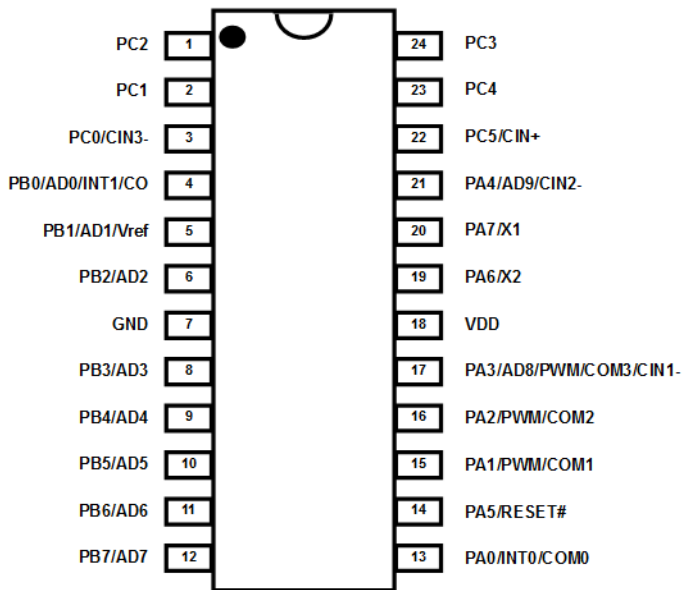


PMC234-S20 (SOP20-300mil)
 PMC234-D20 (DIP20-300mil)

◆ **PMS234 series**

PMS234-S24 (SOP24-300mil)

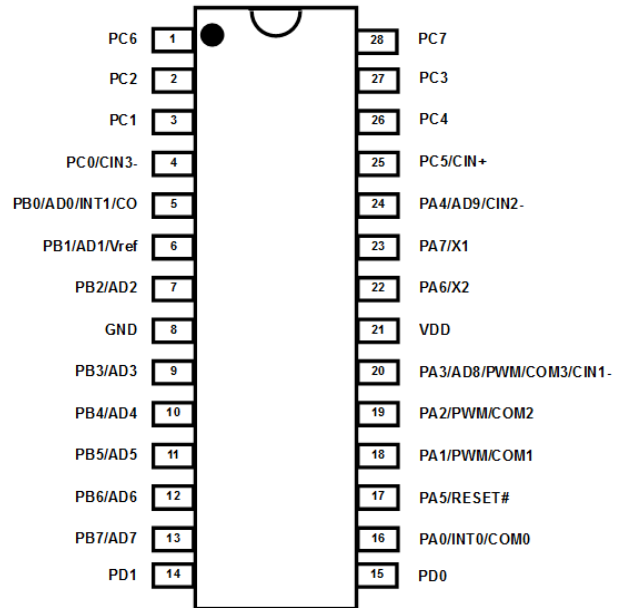
PMS234-K24 (Skinny DIP24-300mil)



PMS234-S24 (SOP24 300mil)
PMS234-K24 (Skinny DIP24 300mil)

PMS234-S28 (SOP28-300mil)

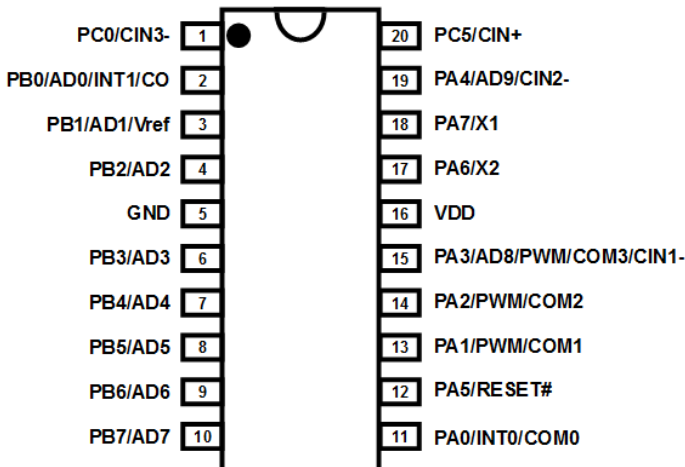
PMS234-K28 (Skinny DIP28-300mil)



PMS234-S28 (SOP28-300mil)
PMS234-K28 (Skinny DIP-300mil)

PMS234-S20 (SOP20-300mil)

PMS234-D20 (DIP20-300mil)



PMS234-S20 (SOP-300mil)
PMS234-D20 (DIP-300mil)

Pin Description

Pin Name	Pin Type & Buffer Type	Description
PA7 / X1	IO ST / CMOS	<p>The functions of this pin can be:</p> <p>(1) Bit 7 of port A. It can be configured as digital input and two-state output with pull-up resistor by software independently</p> <p>(2) X1 when crystal oscillator is used.</p> <p>If this pin is used for crystal oscillator, bit 7 of padier register must be programmed "0" to avoid leakage current. This pin can be used to wake-up system during sleep mode; however, wake-up function is also disabled if bit 7 of padier register is "0".</p>
PA6 / X2	IO ST / CMOS	<p>The functions of this pin can be:</p> <p>(1) Bit 6 of port A. It can be configured as digital input, and two-state output with pull-up resistor by software independently</p> <p>(2) X2 when crystal oscillator is used.</p> <p>If this pin is used for crystal oscillator, bit 6 of padier register must be programmed "0" to avoid leakage current. This pin can be used to wake-up system during sleep mode; however, wake-up function is also disabled if bit 6 of padier register is "0".</p>
PA5 / RESET#	IO (OC) ST / CMOS	<p>The functions of this pin can be:</p> <p>(1) Bit 5 of port A. It can be configured as input or open-drain output pin. <u>Please notice that there is no pull-up resistor in this pin.</u></p> <p>(2) Hardware reset.</p> <p>This pin can be used to wake-up system during sleep mode; however, wake-up function is also disabled if bit 5 of padier register is "0". <u>Please put 33Ω resistor in series to have high noise immunity when this pin is in input mode. Please set it output low or input with external pull-up when not used to prevent leakage current.</u></p>
PA4 / AD9 / CIN2-	IO ST / CMOS / Analog	<p>The functions of this pin can be:</p> <p>(1) Bit 4 of port A. It can be configured as digital input and two-state output with pull-up resistor by software independently</p> <p>(2) Channel 9 of ADC analog input</p> <p>(3) Minus input source of comparator.</p> <p>When this pin is configured as analog input, please use bit 4 of register padier to disable the digital input to prevent current leakage. The bit 4 of padier register can be set to "0" to disable digital input; wake-up from power-down by toggling this pin is also disabled.</p>

Pin Name	Pin Type & Buffer Type	Description
PA3 / AD8 / PWM / COM3 / CIN1-	IO ST / CMOS / Analog	<p>The functions of this pin can be:</p> <ol style="list-style-type: none"> (1) Bit 3 of port A. It can be configured as digital input and two-state output with pull-up resistor independently by software (2) Channel 8 of ADC analog input (3) Minus input source of comparator (4) PWM output from Timer2 (5) COM3 for LCD. It can provide VDD/2 LCD bias voltage. <p>When this pin is configured as analog input, please use bit 3 of register padier to disable the digital input to prevent current leakage. The bit 3 of padier register can be set to "0" to disable digital input; wake-up from power-down by toggling this pin is also disabled.</p>
PA2 / PWM / COM2	IO ST / CMOS	<p>The functions of this pin can be:</p> <ol style="list-style-type: none"> (1) Bit 2 of port A. It can be configured as digital input and two-state output with pull-up resistor independently by software (2) PWM output from Timer2 (3) COM2 for LCD. It can provide VDD/2 LCD bias voltage. <p>The bit 2 of padier register can be set to "0" to disable wake-up from power-down by toggling this pin.</p>
PA1 / PWM / COM1	IO ST / CMOS	<p>The functions of this pin can be:</p> <ol style="list-style-type: none"> (1) Bit 1 of port A. It can be configured as digital input and two-state output with pull-up resistor independently by software (2) PWM output from Timer2 (3) COM1 for LCD. It can provide VDD/2 LCD bias voltage. <p>The bit 1 of padier register can be set to "0" to disable wake-up from power-down by toggling this pin.</p>
PA0 / INT0 / COM0	IO ST / CMOS	<p>The functions of this pin can be:</p> <ol style="list-style-type: none"> (1) Bit 0 of port A. It can be configured as digital input and two-state output with pull-up resistor independently by software (2) External interrupt line 0. It can be used as an external interrupt line 0. <u>Both rising edge and falling edge are accepted to request interrupt service and configurable by register setting.</u> (3) COM0 for LCD. This pin is also the COM0 from LCD bias voltage generator. <p>The bit 0 of padier register can be set to "0" to disable wake-up from power-down by toggling this pin.</p>

Pin Name	Pin Type & Buffer Type	Description
PB7 / AD7	IO ST / CMOS / Analog	<p>The functions of these pins can be:</p> <p>(1) Bit 7~1 of port B. These seven pins can each be configured as analog input, digital input, and two-state output mode with pull-up resistor independently by software.</p> <p>(2) Channel 7~1 of ADC analog input.</p> <p>When any of these seven pins acts as analog inputs, please use register <i>pbdier</i> to disable the digital input to prevent current leakage. If the control bit in <i>pbdier</i> register is set to “0” to disable digital input, wake-up from power-down by toggling the corresponding pin is also disabled.</p>
PB6 / AD6		
PB5 / AD5		
PB4 / AD4		
PB3 / AD3		
PB2 / AD2		
PB1 / AD1		
PB0 / AD0 / INT1 / CO	IO ST / CMOS / Analog	<p>The functions of this pin can be:</p> <p>(1) Bit 0 of port B. It can be configured as analog input, digital input, and two-state output mode with pull-up resistor independently by software.</p> <p>(2) Channel 0 of ADC analog input. When this pin acts as analog input, please use bit 0 of register <i>pbdier</i> to disable the digital input to prevent current leakage.</p> <p>(3) External interrupt line 1. <u>Both rising edge and falling edge are accepted to request interrupt service and configurable by register setting.</u></p> <p>(4) Comparator output.</p> <p>If bit 0 of <i>pbdier</i> register is set to “0” to disable digital input, wake-up from power-down by toggling this pin is also disabled.</p>
PC5 / CIN+	IO ST / CMOS / Analog	<p>The functions of this pin can be:</p> <p>(1) Bit 5 of port C. It can be configured as analog input, digital input, and two-state output mode with pull-up resistor independently by software.</p> <p>(2) Plus or minus input source of comparator.</p>
PC7	IO ST / CMOS	<p>Bit 7, 6, 4, 3, 2, 1 of port C. These pins can be configured as digital input mode and two-state output mode with pull-up resistor independently by software.</p>
PC6		
PC4		
PC3		
PC2		
PC1		
PC0 / CIN3-	IO ST / CMOS / Analog	<p>The functions of this pin can be:</p> <p>(1) Bit 0 of port C. It can be configured as analog input, digital input, and two-state output mode with pull-up resistor independently by software.</p> <p>(2) Minus input source of comparator.</p>
PD1	IO	<p>Bit 1, 0 of port D. These pins can be configured as digital input mode and two-state output mode with pull-up resistor independently by software.</p>
PD0	ST / CMOS	
VDD	VDD	Positive power
GND	GND	Ground

Notes: IO: Input/Output; ST: Schmitt Trigger input; OC: Open Collector; Analog: Analog input pin;
CMOS: CMOS voltage level

4. Device Characteristics

4-1. AC/DC Device Characteristics

Symbol	Description	Min	Typ	Max	Unit	Conditions (Ta=25°C)
V _{DD}	Operating Voltage	2.2	5.0	5.5	V	
f _{SYS}	System clock					V _{DD} ≥ 4.0V
	IHRC	0		8M		V _{DD} ≥ 3.0V
	IHRC & crystal oscillator	0		4M		V _{DD} ≥ 2.5V
	IHRC & crystal oscillator	0		2M	Hz	V _{DD} ≥ 2.2V
	IHRC & crystal oscillator	0		1M		V _{DD} ≥ 5.0V
	Internal low RC oscillator		24K			
I _{OP}	Operating Current		1.7		mA	f _{SYS} =IHRC/16=1MIPS@5.0V
			15		μA	f _{SYS} =ILRC=12KHz@3.3V
I _{PD}	Power Down Current (by stopsys command)		1.0		μA	f _{SYS} = 0Hz, VDD=5.0V
			0.5		μA	f _{SYS} = 0Hz, VDD=3.3V
I _{PS}	Power Save Current (by stopexe command)		0.3		mA	VDD=5.0V; Band-gap, LVR, IHRC, ILRC, Timer16 modules are ON.
V _{IL}	Input low voltage for IO lines	0		0.3V _{DD}	V	
V _{IH}	Input high voltage for IO lines	0.7 V _{DD}		V _{DD}	V	
I _{OL}	IO lines sink current	7	10	13	mA	V _{DD} =5.0V, V _{OL} =0.5V
I _{OH}	IO lines drive current	-5	-7	-9	mA	V _{DD} =5.0V, V _{OH} =4.5V
V _{IN}	Input voltage	-0.3		VDD+0.3	V	
I _{INJ (PIN)}	Injected current on pin			1	mA	VDD+0.3 ≥ V _{IN} ≥ -0.3
R _{PH}	Pull-high Resistance		62			V _{DD} =5.0V
			100		KΩ	V _{DD} =3.3V
			210		KΩ	V _{DD} =2.2V
V _{LVR}	Low Voltage Reset Voltage *	3.86	4.15	4.44	V	
		3.35	3.60	3.85		
		2.84	3.05	3.26		
		2.61	2.80	3.00		
		2.37	2.55	2.73		
		2.04	2.20	2.35		
		1.86	2.00	2.14		
		1.67	1.80	1.93		
V _{BG}	Band-gap Reference Voltage (before calibration)	1.11	1.20	1.29	V	VDD=5V, 25°C
	Band-gap Reference Voltage * (after calibration)	1.140*	1.200*	1.260*		VDD=2.2V ~ 5.5V, -40°C < Ta < 85°C*
		1.145*	1.200*	1.255*		-20°C < Ta < 70°C*
f _{IHRC}	Frequency of IHRC after calibration *	15.76*	16*	16.24*	MHz	25°C, VDD=2.2V~5.5V
		15.04*	16*	16.96*		VDD=2.2V~5.5V, -40°C < Ta < 85°C*
		15.28*	16*	16.72*		VDD=2.2V~5.5V, -20°C < Ta < 70°C*

Symbol	Description	Min	Typ	Max	Unit	Conditions (Ta=25°C)
f _{ILRC}	Frequency of ILRC *	20.4*	24*	27.6*	KHz	VDD=5.0V, Ta=25°C
		15.6*	24*	32.4*		VDD=5.0V, -40°C <Ta<85°C*
		16.8*	24*	31.2*		VDD=5.0V, -20°C <Ta<70°C*
		10.2*	12*	13.8*		VDD=3.3V, Ta=25°C
		4.55*	7*	9.45*		VDD=2.2V, -40°C <Ta<85°C*
		5.55*	7*	8.45*		VDD=2.2V, -20°C <Ta<70°C*
t _{INT}	Interrupt pulse width	30			ns	V _{DD} = 5.0V
V _{ADC}	Supply voltage for workable ADC	2.5		5.0	V	
V _{AD}	AD Input Voltage	0		VDD	V	
ADrs	ADC resolution			12	bit	
ADcs	ADC current consumption		0.9 0.8		mA	@5V @3V
ADclk	ADC clock period		2		us	2.5V ~ 5.5V
t _{ADCONV}	ADC conversion time (T _{ADCLK} is the period of the selected AD conversion clock)		17		T _{ADCLK}	12-bit resolution
AD DNL	ADC Differential NonLinearity		±2*		LSB	
AD INL	ADC Integral NonLinearity		±4*		LSB	
ADos	ADC offset*		3		mV	
V _{DR}	RAM data retention voltage*	1.5			V	PMC234/PMS234 in stop mode.
CPos	Comparator offset*	-	±10	±20	mV	
CPcm	Comparator input common mode*	0		VDD-1.5	V	
CPspt	Comparator response time**		100	500	ns	Both Rising and Falling
CPmc	Stable time to change comparator mode		2.5	7.5	us	
CPcs	Comparator current consumption		40 20		uA	@5.0V @3.3V
R _(VDD/2)	(VDD/2) pull-up / pull-down resistance	2.5	5	10	KΩ	
ΔV _(VDD/2)	Deviation of (VDD/2) output voltage		±1%	±3%		@ V _{DD} = 5.0V
t _{WDT}	Watchdog timeout period		2408		ILRC clock period	misc[1:0]=00 (default)
			4096			misc[1:0]=01
			16384			misc[1:0]=10
			256			misc[1:0]=11
t _{SBP}	System boot-up period from power-on		1024		T _{ILRC}	Where T _{ILRC} is the clock period of ILRC

Symbol	Description	Min	Typ	Max	Unit	Conditions (Ta=25°C)
t _{WUP}	System wake-up period					
	Fast wake-up by IO toggle from STOPEXE suspend		128		T _{SYS}	Where T _{SYS} is the time period of system clock
	Fast wake-up by IO toggle from STOPSYS suspend, IHRC is the system clock		128 T _{SYS} + T _{SIHRC}			Where T _{SIHRC} is the stable time of IHRC from power-on. T _{SIHRC} = 5us@5V
	Fast wake-up by IO toggle from STOPSYS suspend, ILRC is the system clock		128 T _{SYS} + T _{SILRC}			Where T _{SILRC} is the stable time of ILRC from power-on. T _{SILRC} = 43ms@5V
	Normal wake-up from STOPEXE or STOPSYS suspend		1024		T _{ILRC}	Where T _{ILRC} is the clock period of ILRC
t _{RST}	External reset pulse width	120			us	@VDD=5V

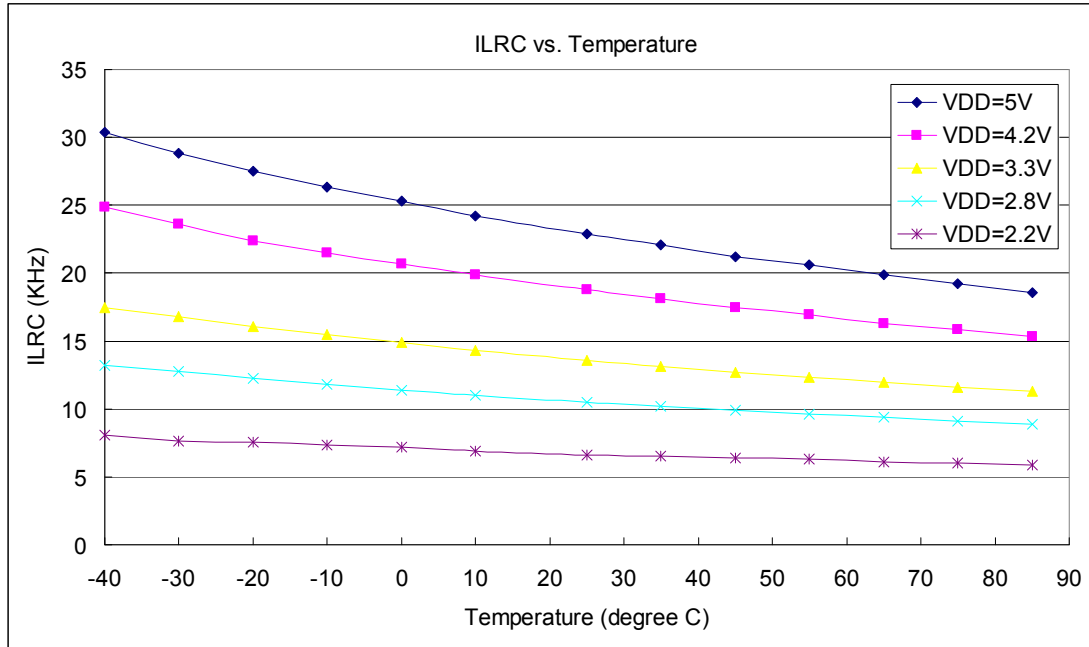
*These parameters are for design reference, not tested for each chip.

** Response time is measured with comparator input at (VDD-1.5)/2 -100mV, and (VDD-1.5)/2+100mV

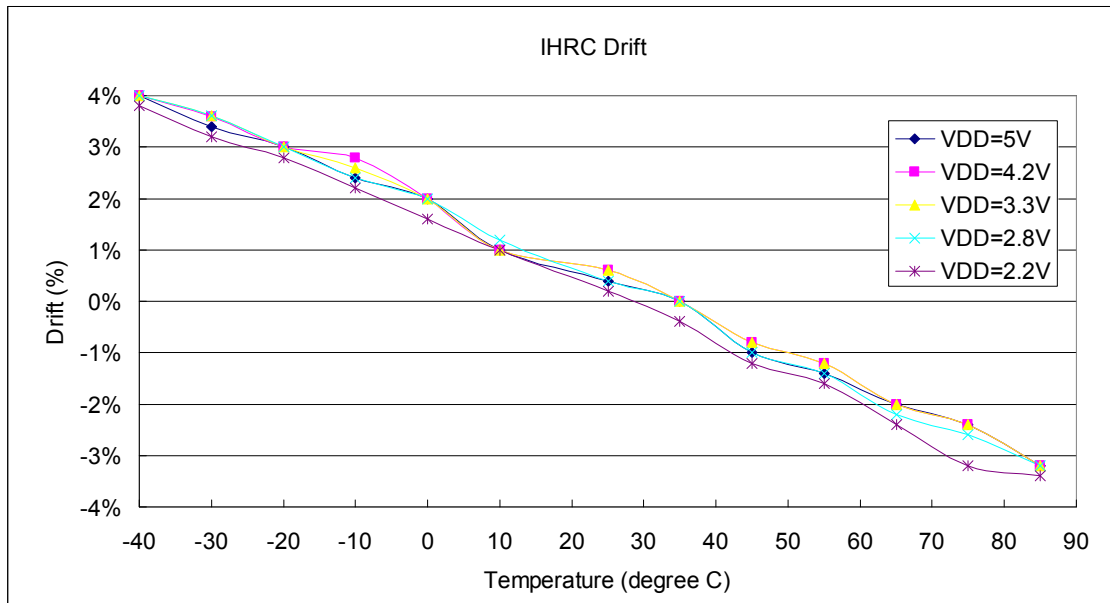
4-2. Absolute Maximum Ratings

- Supply Voltage 2.2V ~ 5.5V
- Input Voltage -0.3V ~ VDD + 0.3V
- Operating Temperature PMC Series: -40°C ~ 85°C
PMS Series: -20°C ~ 70°C
- Junction Temperature 150°C
- Storage Temperature -50°C ~ 125°C

4-3. Typical ILRC frequency vs. VDD and temperature



4-4. Typical IHRC frequency deviation vs. VDD and temperature



Note: IHRC is calibrated to 16MHz

4-5. Typical operating current vs. VDD @ system clock = ILRC/n

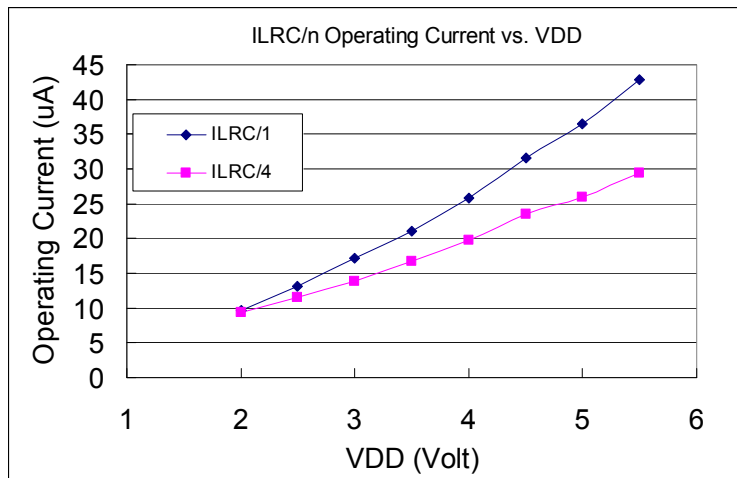
Conditions:

2-FPPA (FPPA0: tog PA0, FPPA1: idle)

ON: ILRC;

OFF: Band-gap, LVR, IHRC, EOSC, T16, TM2, ADC, Comparator modules;

IO: PA0:0.5Hz output toggle and no loading, others: input and no floating



4-6. Typical operating current vs. VDD @ system clock = IHRC/n

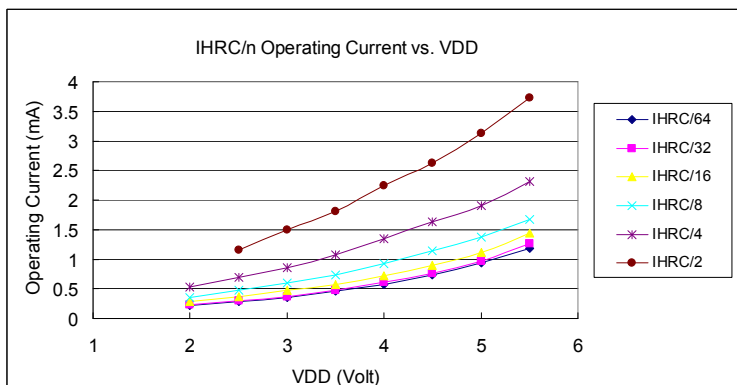
Conditions:

2-FPPA (FPPA0: tog PA0, FPPA1: idle)

ON: Band-gap, LVR, IHRC;

OFF: ILRC, EOSC, T16, TM2, ADC, Comparator modules;

IO: PA0:0.5Hz output toggle and no loading, others: input and no floating



4-7. Typical operating current vs. VDD @ system clock = 1MHz EOSC / n

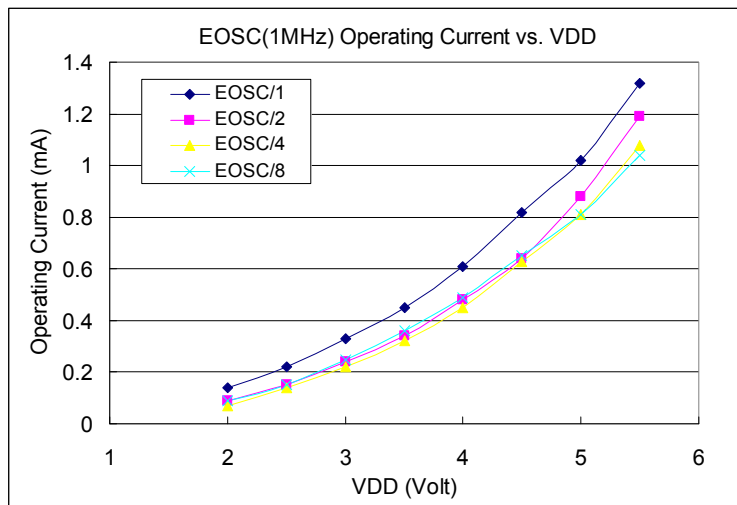
Conditions:

2-FPPA (FPPA0: tog PA0, FPPA1: idle)

ON: EOSC, MISC.6 = 1;

OFF: Band-gap, LVR, IHRC, ILRC, T16, TM2, ADC, Comparator modules;

IO: PA0:0.5Hz output toggle and no loading, others: input and no floating



4-8. Typical operating current vs. VDD @ system clock = 4MHz EOSC / n

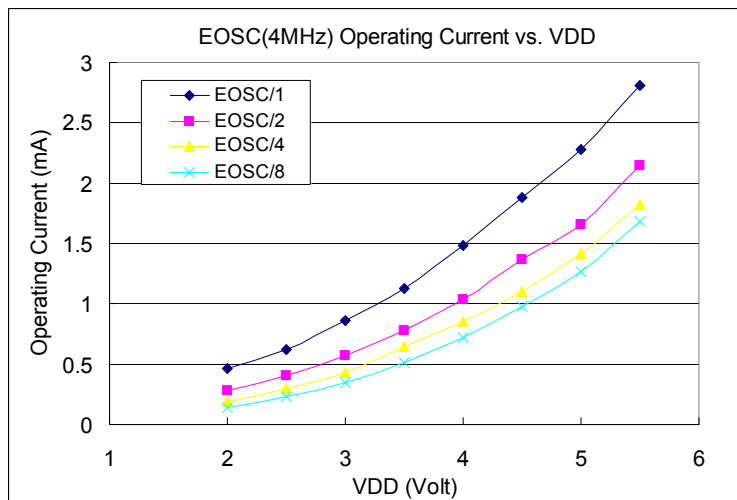
Conditions:

2-FPPA (FPPA0: tog PA0, FPPA1: idle)

ON: EOSC, MISC.6 = 1;

OFF: Band-gap, LVR, IHRC, ILRC, T16, TM2, ADC, Comparator modules;

IO: PA0:0.5Hz output toggle and no loading, others: input and no floating



4-9. Typical operating current vs. VDD @ system clock = 32KHz EOSC / n

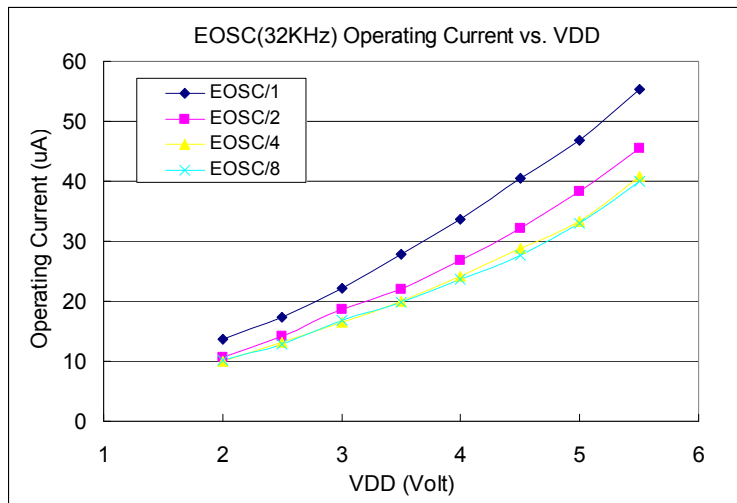
Conditions:

2-FPPA (FPPA0: tog PA0, FPPA1: idle)

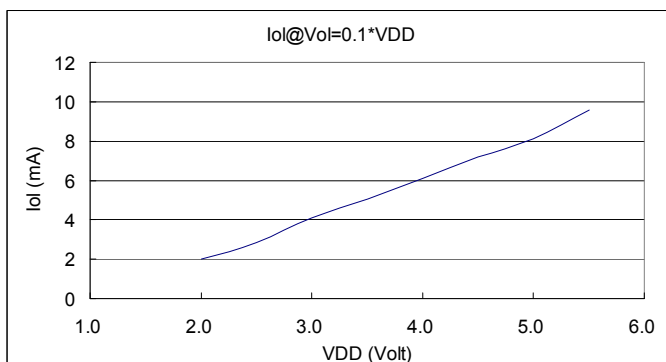
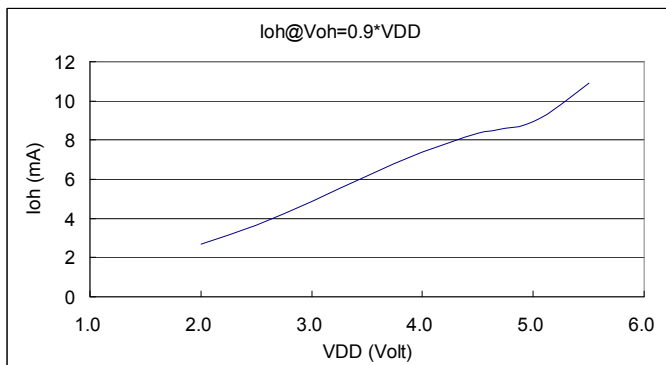
ON: EOSC, MISC.6 = 1;

OFF: Band-gap, LVR, IHRC, ILRC, T16, TM2, ADC, Comparator modules;

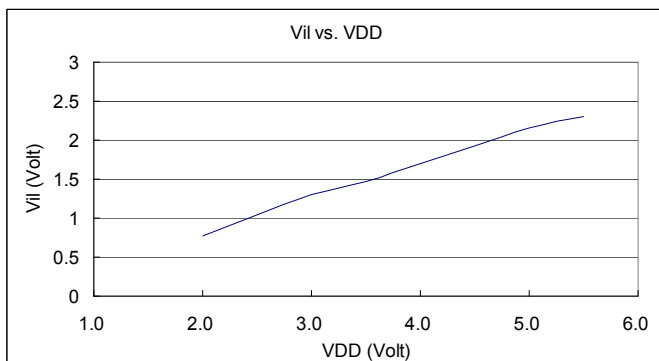
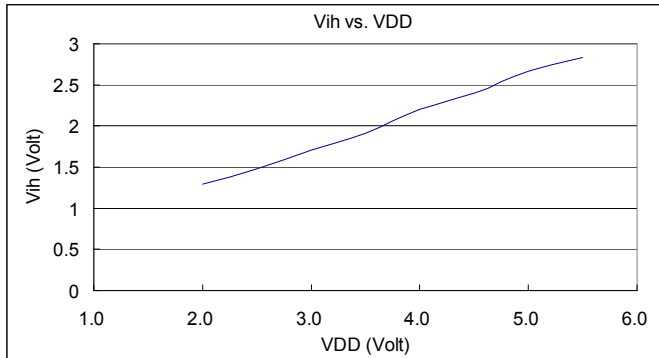
IO: PA0:0.5Hz output toggle and no loading, others: input and no floating



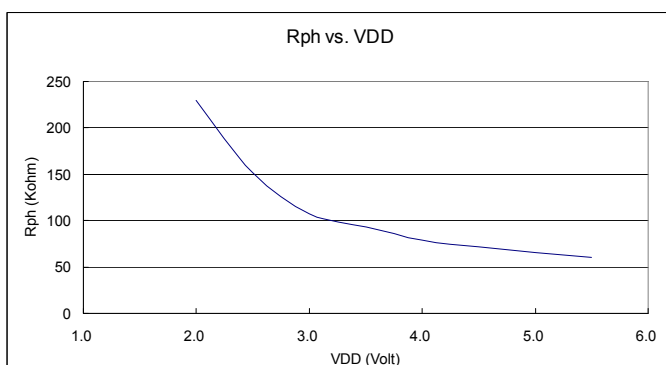
4-10. Typical IO driving current (I_{OH}) and sink current (I_{OL})



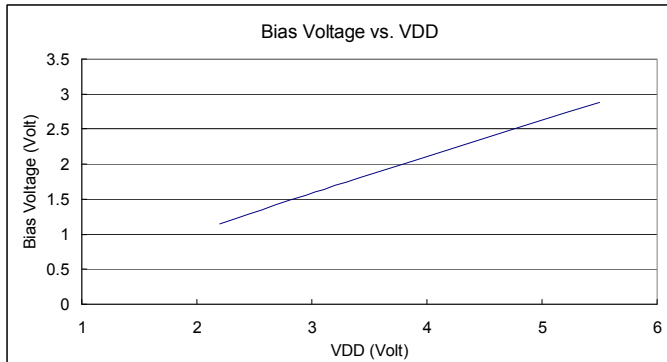
4-11. Typical IO input high/low threshold voltage (V_{IH}/V_{IL})



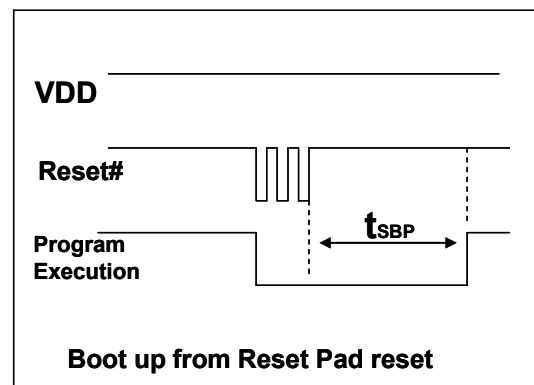
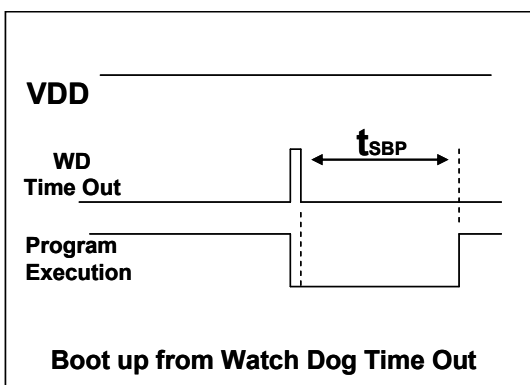
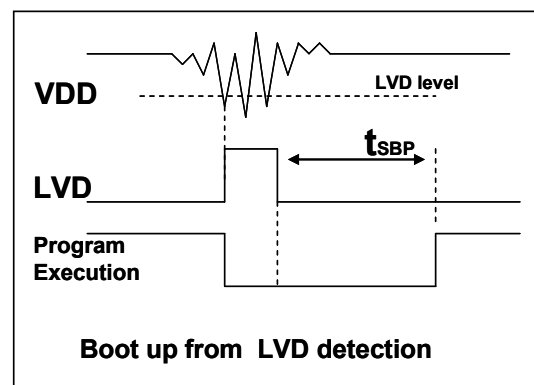
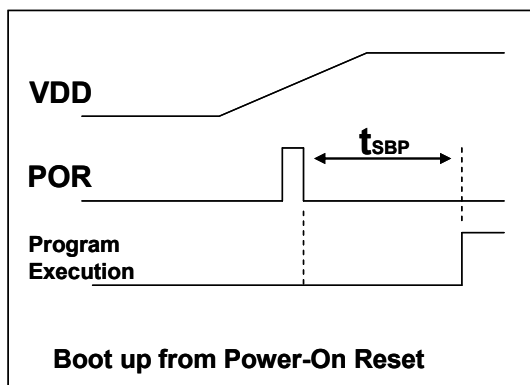
4-12. Typical resistance of IO pull high device



4-13. Typical VDD/2 Bias output voltage



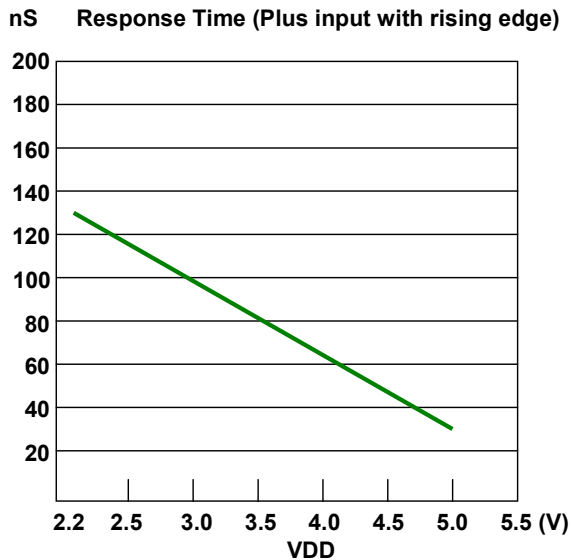
4-14. Timing charts for boot up conditions



4-15. Typical Comparator Responsive Time (Use $V_{\text{internal R}}$)

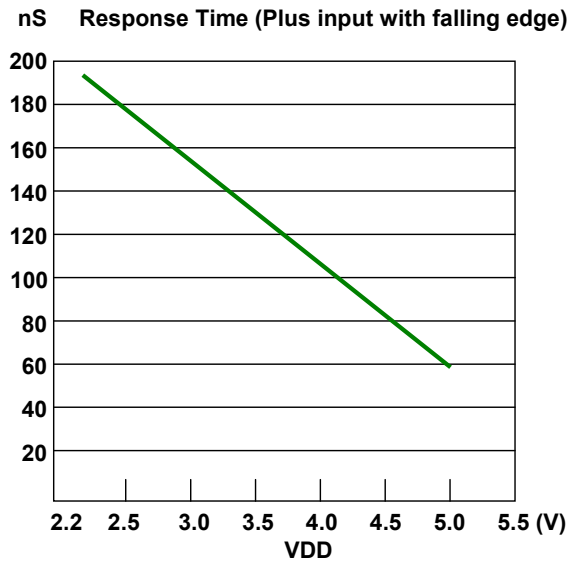
Measurement Conditions:

- ◆ Plus input: PC5
- ◆ Minus input: $V_{\text{internal R}} = (5/16)*VDD$
- ◆ $V_{\text{internal R}}$: use Case I, R1
- ◆ Fix $V_{\text{internal R}}$, use PC5 rising edge to measure response time



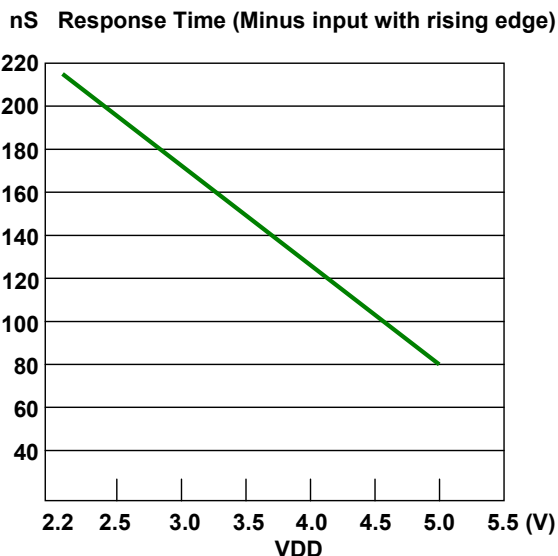
Measurement Conditions:

- ◆ Plus input: PC5
- ◆ Minus input: $V_{\text{internal R}} = (5/16)*VDD$
- ◆ $V_{\text{internal R}}$: use Case I, R1
- ◆ Fix $V_{\text{internal R}}$, use PC5 falling edge to measure response time



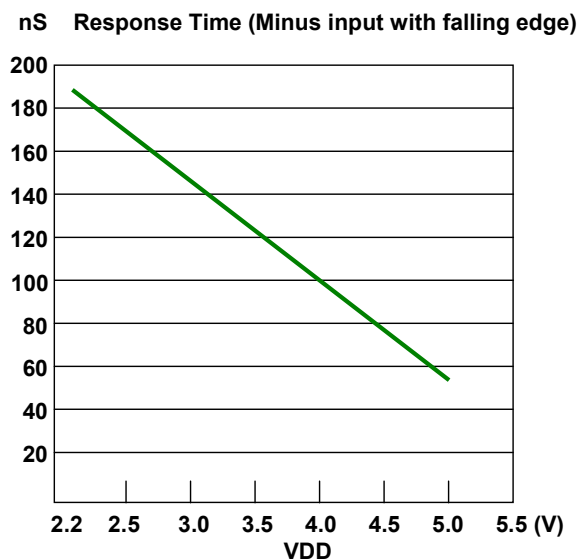
Measurement Conditions:

- ◆ Plus input: $V_{\text{internal R}} = (5/16)*VDD$
- ◆ Minus input: PC5
- ◆ $V_{\text{internal R}}$: use Case I, R1
- ◆ Fix $V_{\text{internal R}}$, use PC5 rising edge to measure response time



Measurement Conditions:

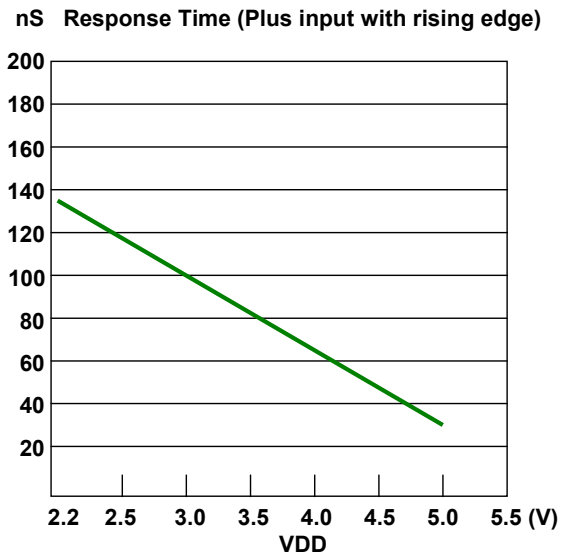
- ◆ Plus input: $V_{\text{internal R}} = (5/16)*VDD$
- ◆ Minus input: PC5
- ◆ $V_{\text{internal R}}$: use Case I, R1
- ◆ Fix $V_{\text{internal R}}$, use PC5 falling edge to measure response time



4-16. Typical Comparator Responsive Time (Use External Inputs)

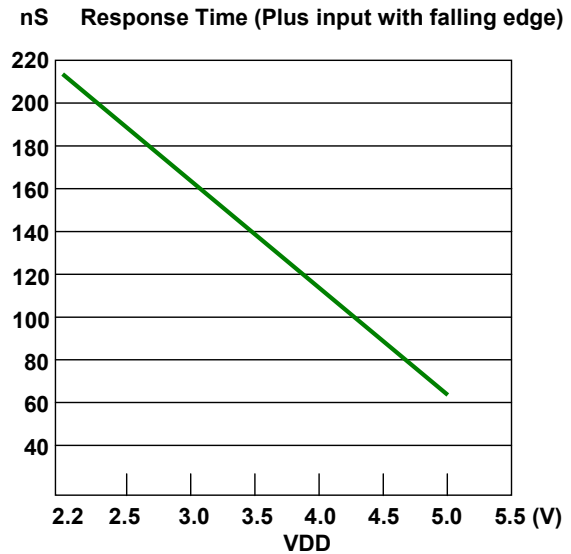
Measurement Conditions:

- ◆ Plus input: PC5
- ◆ Minus input: PC0 = (5/16)VDD
- ◆ Fix PC0, use PC5 rising edge to measure response time



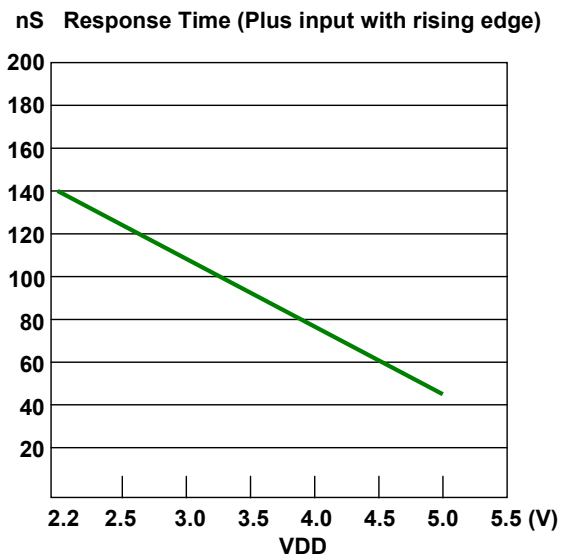
Measurement Conditions:

- ◆ Plus input: PC5
- ◆ Minus input: PC0 = (5/16)VDD
- ◆ Fix PC0, use PC5 falling edge to measure response time



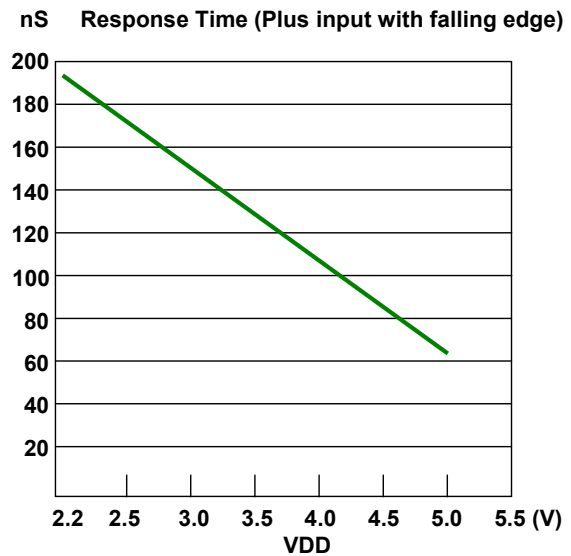
Measurement Conditions:

- ◆ Plus input: PC5 = (5/16)VDD
- ◆ Minus input: PC0
- ◆ Fix PC5, use PC0 rising edge to measure response time



Measurement Conditions:

- ◆ Plus input: PC5 = (5/16)VDD
- ◆ Minus input: PC0
- ◆ Fix PC5, use PC0 falling edge to measure response time



5. Functional Description

5-1. Processing Units

There are two processing units (FPP unit) inside the chip of PMC234/PMS234. In each processing unit, it includes (i) its own Program Counter to control the program execution sequence (ii) its own Stack Pointer to store or restore the program counter for program execution (iii) its own accumulator (iv) Status Flag to record the status of program execution. Each FPP unit has its own program counter and accumulator for program execution, flag register to record the status, and stack pointer for jump operation. Based on such architecture, FPP unit can execute its own program independently, thus parallel processing can be expected.

These two FPP units share the same 4Kx16 bits OTP program memory, 208 bytes data SRAM and all the IO ports, these two FPP units are operated at mutual exclusive clock cycles to avoid interference. One task switch is built inside the chip to decide which FPP unit should be active for the corresponding cycle. The hardware diagram and basic timing diagram of FPP units are illustrated in Fig. 1. For FPP0 unit, its program will be executed in sequence every other system clock, shown as $(M-1)_{th}$, M_{th} and $(M+1)_{th}$ instructions. For FPP1 unit, its program will be also executed in sequence every other system clock, shown as $(N-1)_{th}$, N_{th} and $(N+1)_{th}$ instructions.

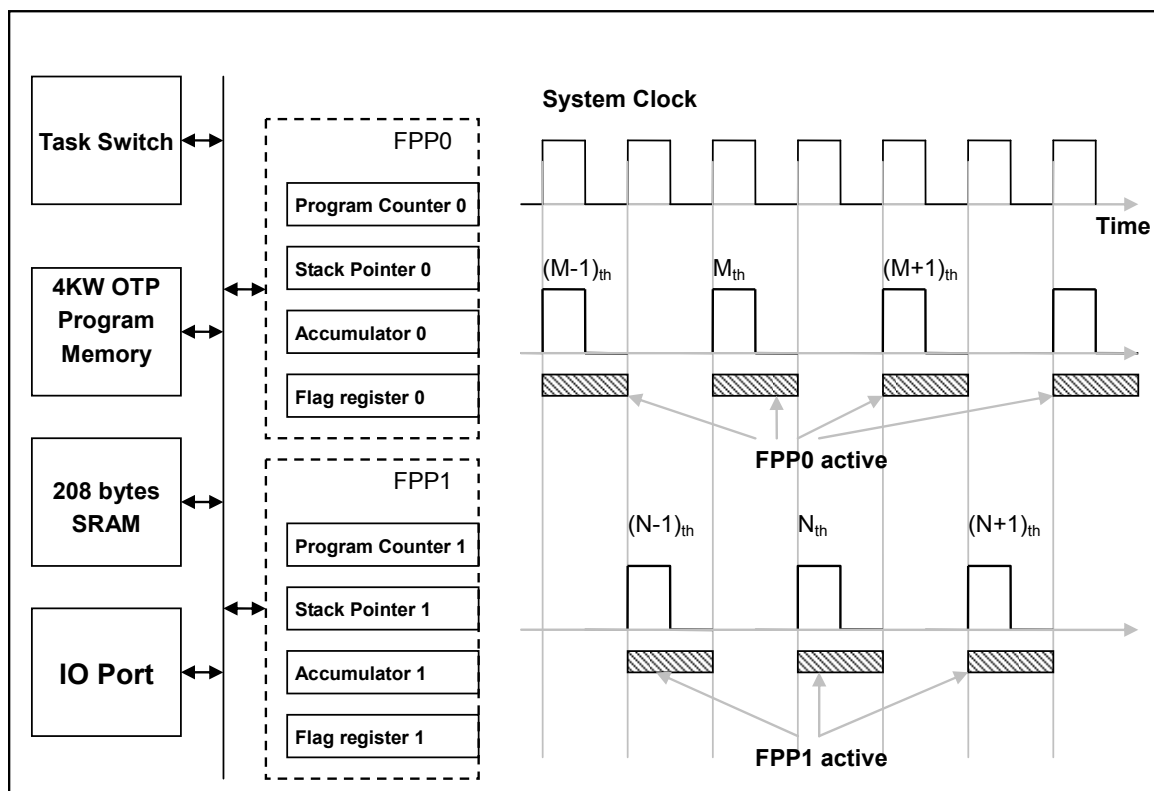


Fig. 1 : Hardware and Timing Diagram of FPP unit

Each FPP unit has half computing power of whole system; for example, FPP0 and FPP1 will be operated at 4MHz if system clock is 8MHz. The FPP unit can be enabled or disabled by programming the FPP unit Enable Register, only FPP0 is enabled after power-on reset. The system initialization will be started from FPP0 and FPP1 unit can be enabled by user's program if necessary. Both FPP0 and FPP1 can be enabled or disabled by using any one FPP unit.

5-1-1. Program Counter

Program Counter (PC) is the unit that contains the address of an instruction to be executed next. The program counter is automatically incremented at each instruction cycle so that instructions are retrieved sequentially from the program memory. Certain instructions, such as branches and subroutine calls, interrupt the sequence by placing a new value in the program counter. The bit length of the program counter is 12 for PMC234/PMS234. The program counter of FPP0 is 0 after hardware reset and 1 for FPP1. Whenever an interrupt happens in FPP0, the program counter will jump to 0x10 for interrupt service routine. Each FPP unit has its own program counter to control the program execution sequence.

5-1-2. Stack Pointer

The stack pointer in each processing unit is used to point the top of the stack area where the local variables and parameters to subroutines are stored; the stack pointer register (sp) is located in IO address 0x02h. The bit number of stack pointer is 8 bit; the stack memory cannot be accessed over 208 bytes and should be defined within 208 bytes from 0x00h address. The stack memory of PMC234/PMS234 for each FPP unit can be assigned by user via stack pointer register, means that the depth of stack pointer for each FPP unit is adjustable in order to optimize system performance. The following example shows how to define the stack in the ASM (assembly language) project:

```

    . ROMADR    0
    GOTO       FPPA0
    GOTO       FPPA1
    ...
    . RAMADR    0                                // Address must be less than 0x100
    WORD       Stack0 [1]                       // one WORD
    WORD       Stack1 [2]                       // two WORD
    ...
FPPA0:
    SP =       Stack0;                          // assign Stack0 for FPPA0,
                                                // one level call because of Stack0[1]
    ...
    call      function1
    ...
FPPA1:
    SP =       Stack1;                          // assign Stack1 for FPPA1,
                                                // two level call because of Stack1[2]
    ...
    call      function2
    ...

```

In Mini-C project, the stack calculation is done by system software, user will not have effort on it, and the example is shown as below:

```

void      FPPA0 (void)
{
    ...
}

```

User can check the stack assignment in the window of program disassembling, Fig. 2 shows that the status of stack before FPP0 execution, system has calculated the required stack space and has reserved for the program.

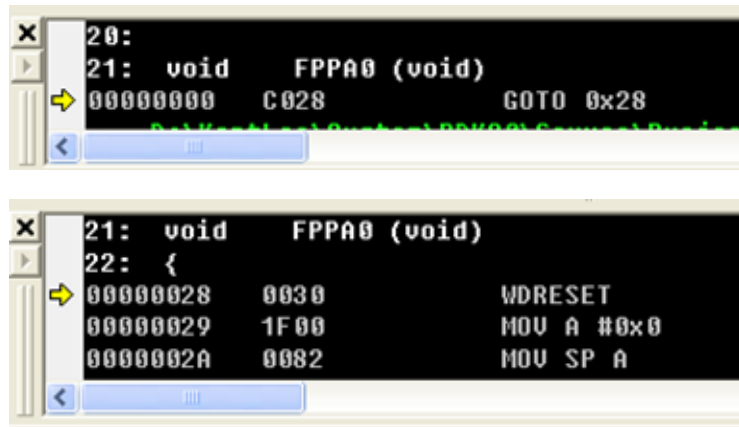


Fig. 2 : Stack Assignment in Mini-C project

5-1-3. Single FPP mode

For traditional MCU user who does not need parallel processing capability, PMC234/PMS234 provides single FPP mode optional to behave as traditional MCU. After single FPP mode is selected, FPP1 is always disabled and only FPP0 is active. Fig.3 shows the timing diagram for each FPP unit, FPP1 is always disabled and only FPP0 active. Please notice that wait and delay instructions are NOT supported when single FPP mode is chosen.

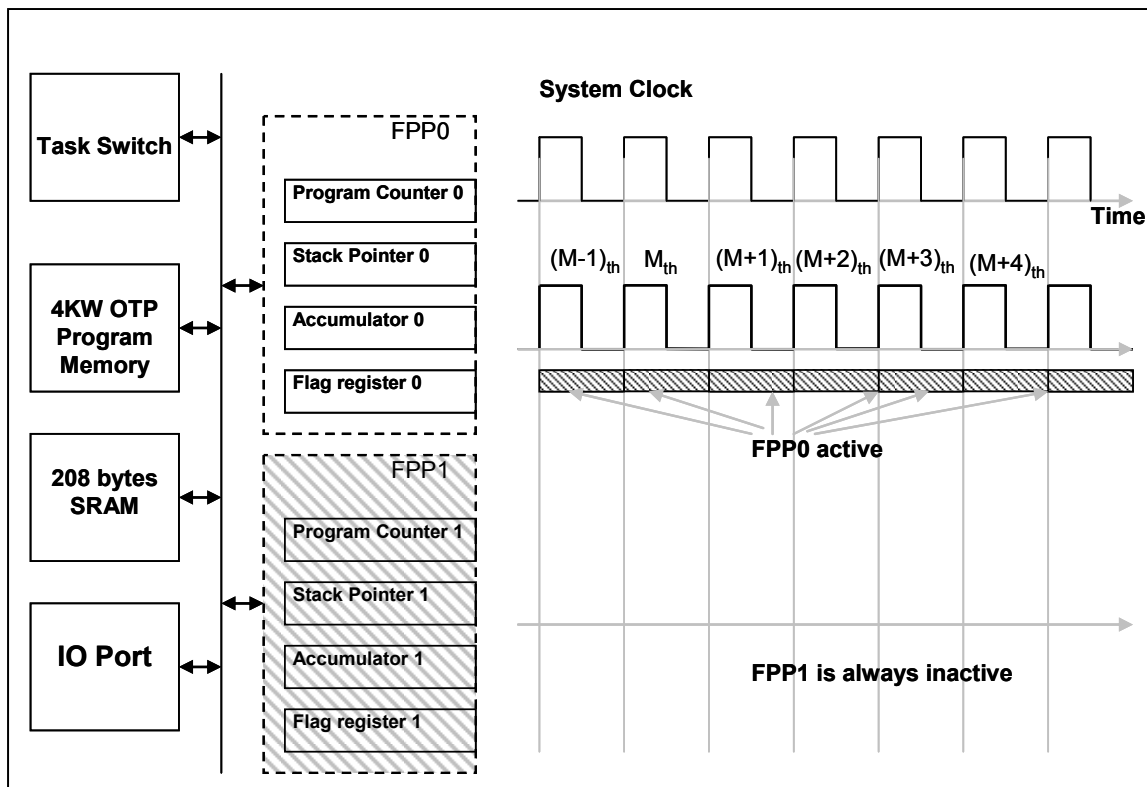


Fig. 3 : Timing Diagram of single FPP mode

5-2. Program Memory -- OTP

5-2-1. Program Memory Assignment

The OTP (One Time Programmable) program memory is used to store the program instructions to be executed. All program codes for each FPP unit are stored in this OTP memory for both FPP0 and FPP1. The OTP program memory may contains the data, tables and interrupt entry. After reset, the initial address for FPP0 is 0x0 and 0x1 for FPP1. The interrupt entry is 0x10 if used and interrupt function is for FPP0 only, the last eight addresses are reserved for system using, like checksum, serial number, etc. The OTP program memory for PMC234/PMS234 is a 4Kx16 bit that is partitioned as Table 1. The OTP memory from address 0xFF8 to 0xFFF is for system using, address space from 0x002 to 0x00F and from 0x011 to 0xFF7 are user program space. The address 0x001 is the FPP1 initial address for two FPP units mode and user program for single FPP unit mode.

Address	Function
0x000	FPP0 reset – goto instruction
0x001	FPP1 reset – goto instruction
0x002	User program
•	•
0x00F	User program
0x010	Interrupt entry address
0x011	User program
•	•
0xFF7	User program
0xFF8	System Using
•	•
0xFFF	System Using

Table 1: Program Memory Organization of PMC234/PMS234

5-2-2. Example of Using Program Memory for Two FPP mode

Table 2 shows one example of using program memory which two FPP units are active.

Address	Function
000	FPP0 reset – goto instruction (goto 0x020)
001	Begin of FPP1 program
•	•
00F	goto 0x1A1 to continue FPP1 program
010	Interrupt entry address (FPP0 only)
•	•
01F	End of ISR
020	Begin of FPP0 user program
•	•
1A0	End of FPP0 user program
1A1	Continuing FPP1 program
•	•
FF7	End of FPP1 program
FF8	System Using
•	•
FFF	System Using

Table 2: Example of using Program Memory for two FPP units mode

5-2-3. Example of Using Program Memory for Single FPP mode

The entire user's program memory can be assigned to FPP0 when single FPP mode is selected. Table 3 shows the example of program memory using for single FPP unit mode.

Address	Function
000	FPP0 reset
001	Begin of user program
002	user program
•	•
00F	goto instruction (goto 0x020)
010	Interrupt entry address
011	ISR
•	•
01F	End of ISR
020	user program
•	•
•	•
FF7	user program
FF8	System Using
•	•
FFF	System Using

Table 3: Example of using Program Memory for single FPP mode

5-3. Program Structure

5-3-1. Program structure of two FPP units mode

After power-up, the program starting address of FPP0 is 0x000 and 0x001 for FPP1. The 0x010 is the entry address of interrupt service routine, which belongs to FPP0 only. The basic firmware structure for PMC234/PMS234 is shown as Fig. 4, the program codes of two FPP units are placed in one whole program space. Except for the initial addresses of processing units and entry address of interrupt, the memory location is not specially specified; the program codes of processing unit can be resided at any location no matter what the processing unit is. After power-up, the fpp0Boot will be executed first, which will include the system initialization and other FPP units enabled.

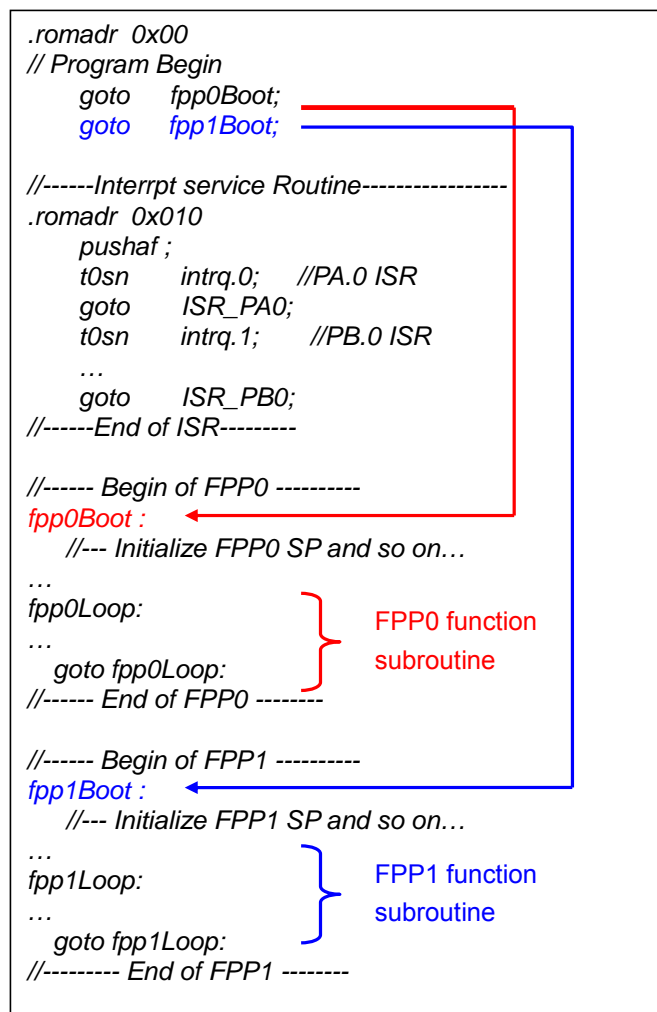


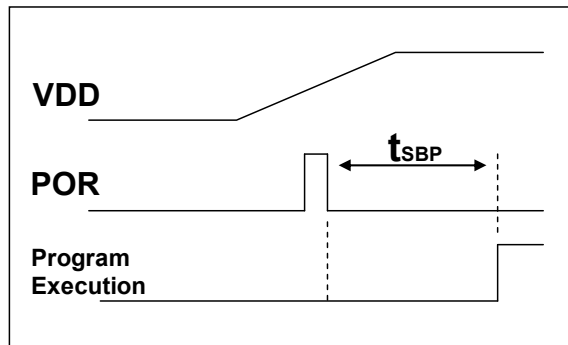
Fig. 4 : Program Structure

5-3-2. Program structure of single FPP mode

After power-up, the program starting address of FPP0 is 0x000, 0x010 is the entry address of interrupt service routine. For single FPP unit mode, the firmware structure is same as traditional MCU. After power-up, the program will be executed from address 0x000, then continuing the program sequence.

5-4. Boot Procedure

POR (Power-On-Reset) is used to reset PMC234/PMS234 when power up, however, the supply voltage may be not stable. To ensure the stability of supply voltage after power up, it will wait 1024 ILRC clock cycles before first instruction being executed, which is t_{SBP} and shown in the Fig. 5. After boot up procedure, the default system clock is ILRC; user should ensure that the power should be stable after boot up time.



Boot up from Power-On Reset

Fig. 5 : Power Up Sequence

Fig. 6 shows the typical program flow after boot up. Please notice that the FPP1 is disabled after reset and recommend NOT enabling FPP1 before system and FPP0 initialization.

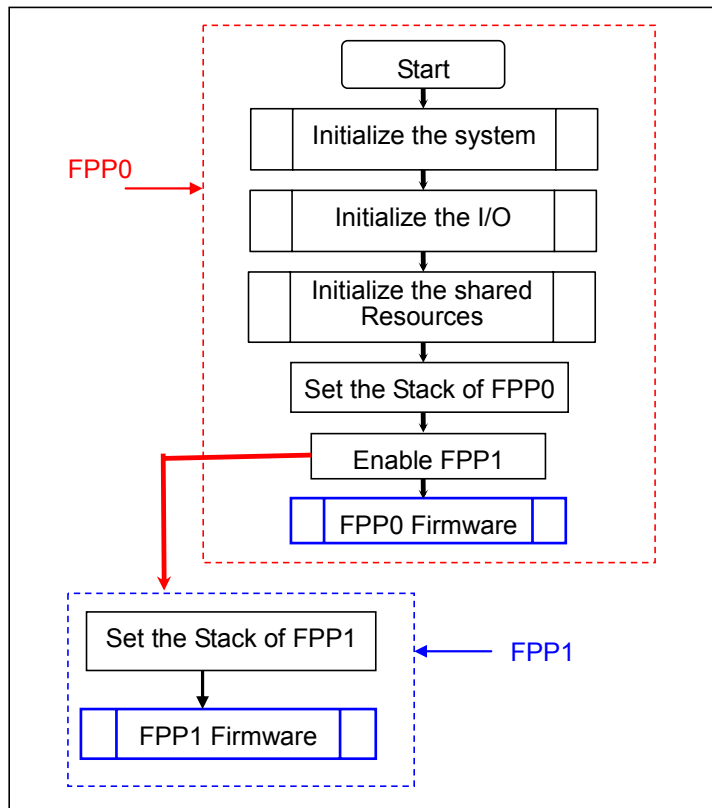


Fig. 6 : Boot Procedure

5-5. Data Memory -- SRAM

Fig. 7 shows the SRAM data memory organization of PMC234/PMS234, all the SRAM data memory could be accessed by FPP0 and FPP1 directly with 1T clock cycle, the data access can be byte or bit operation. Besides data storage, the SRAM data memory is also served as data pointer of indirect access method and the stack memory for all FPP units.

The stack memory for each processing unit should be independent from each other, and defined in the data memory. The stack pointer is defined in the stack pointer register of each processing unit; the depth of stack memory of each processing unit is defined by the user. The arrangement of stack memory fully flexible and can be dynamically adjusted by the user.

For indirect memory access mechanism, the data memory is used as the data pointer to address the data byte. All the data memory could be the data pointer; it's quite flexible and useful to do the indirect memory access. Since the data width is 8-bit, all the 208 bytes data memory of PMC234/PMS234 can be accessed by indirect access mechanism.

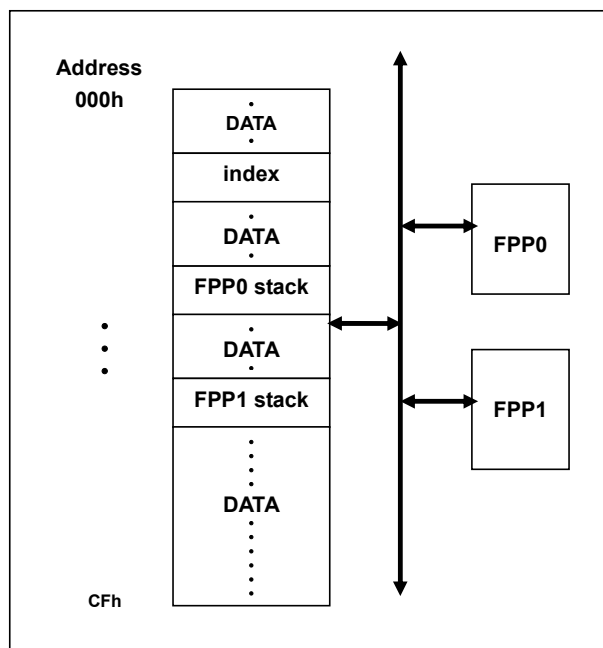


Fig. 7 : Data Memory Organization

5-6. Arithmetic and Logic Unit

Arithmetic and Logic Unit (ALU) is the computation element to operate integer arithmetic, logic, shift and other specialized operations. The operation data can be from instruction, accumulator or SRAM data memory. Computation result could be written into accumulator or SRAM. FPP0 and FPP1 will share ALU for its corresponding operation.

5-7. Oscillator and clock

There are three oscillator circuits provided by PMC234/PMS234: external crystal oscillator (EOSC), internal high RC oscillator (IHRC) and internal low RC oscillator (ILRC), and these three oscillators are enabled or disabled by registers `eoscr.7`, `clkmd.4` and `clkmd.2` independently. User can choose one of these three oscillators as system clock source and use `clkmd` register to target the desired frequency as system clock to meet different application.

Oscillator Module	Enable/Disable	Default after boot-up
EOSC	<code>eoscr.7</code>	Disabled
IHRC	<code>clkmd.4</code>	Enabled
ILRC	<code>clkmd.2</code>	Enabled

5-7-1. Internal High RC oscillator and Internal Low RC oscillator

After boot-up, the IHRC and ILRC oscillators are enabled. The frequency of IHRC can be calibrated to eliminate process variation by `ihrcr` register; normally it is calibrated to 16MHz. The frequency deviation can be within 1% normally after calibration under the calibrated voltage, however, it still drifts slightly with supply voltage and operating temperature, the total drift rate is within $\pm 6\%$ for $VDD=2.2V\sim 5.5V$ and $-40^{\circ}C\sim 85^{\circ}C$ operating conditions. Please refer to the measurement chart for IHRC frequency verse VDD and IHRC frequency verse temperature. The frequency of ILRC is around 24 KHz, however, its frequency will vary by process, supply voltage and temperature, please refer to DC specification and do not use for accurate timing application.

5-7-2. Chip calibration

The IHRC frequency and band-gap reference voltage may be different chip by chip due to manufacturing variation, PMC234/PMS234 provide both the IHRC frequency calibration and band-gap calibration to eliminate this variation, and this function can be selected when compiling user's program and the command will be inserted into user's program automatically. The calibration command is shown as below:

`.ADJUST_IC SYSCLK=IHRC/(p1), IHRC=(p2)MHz, VDD=(p3)V, Band-gap=(p4);`

Where, **p1**=2, 4, 8, 16, 32; In order to provide different system clock.

p2=14 ~ 18; In order to calibrate the chip to different frequency, 16MHz is the usually one.

p3=2.5 ~ 5.5; In order to calibrate the chip under different supply voltage.

p4= On or Off; Band-gap calibration is On or Off.

5-7-3. IHRC Frequency Calibration and System Clock

During compiling the user program, the options for IHRC calibration and system clock are shown as Table 4:

SYSCLK	CLKMD	IHRCR	Description
o Set IHRC / 2	= 34h (IHRC / 2)	Calibrated	IHRC calibrated to 16MHz, CLK=8MHz (IHRC/2)
o Set IHRC / 4	= 14h (IHRC / 4)	Calibrated	IHRC calibrated to 16MHz, CLK=4MHz (IHRC/4)
o Set IHRC / 8	= 3Ch (IHRC / 8)	Calibrated	IHRC calibrated to 16MHz, CLK=2MHz (IHRC/8)
o Set IHRC / 16	= 1Ch (IHRC / 16)	Calibrated	IHRC calibrated to 16MHz, CLK=1MHz (IHRC/16)
o Set IHRC / 32	= 7Ch (IHRC / 32)	Calibrated	IHRC calibrated to 16MHz, CLK=0.5MHz (IHRC/32)
o Set ILRC	= E4h (ILRC / 1)	Calibrated	IHRC calibrated to 16MHz, CLK=ILRC
o Disable	No change	No Change	IHRC not calibrated, CLK not changed, Band-gap OFF

Table 4 Options for IHRC Frequency Calibration

Usually, .ADJUST_IC will be the first command after boot up, in order to set the target operating frequency whenever starting the system. The program code for IHRC frequency calibration is executed only one time that occurs in writing the codes into OTP memory; after then, it will not be executed again. If the different option for IHRC calibration is chosen, the system status is also different after boot. The following shows the status of PMC234/PMS234 for different option:

- (1) .ADJUST_IC SYSCLK=IHRC/2, IHRC=16MHz, VDD=5V, Band-gap=On
 After boot up, CLKMD = 0x34:
 - ◆ IHRC frequency is calibrated to 16MHz@VDD=5V and IHRC module is enabled
 - ◆ System CLK = IHRC/2 = 8MHz
 - ◆ Watchdog timer is disabled, ILRC is enabled, PA5 is in input mode, BG=1.2V

- (2) .ADJUST_IC SYSCLK=IHRC/4, IHRC=16MHz, VDD=3.3V, Band-gap=On
 After boot up, CLKMD = 0x14:
 - ◆ IHRC frequency is calibrated to 16MHz@VDD=3.3V and IHRC module is enabled
 - ◆ System CLK = IHRC/4 = 4MHz
 - ◆ Watchdog timer is disabled, ILRC is enabled, PA5 is in input mode, BG=1.2V

- (3) .ADJUST_IC SYSCLK=IHRC/8, IHRC=16MHz, VDD=2.5V, Band-gap=On
 After boot up, CLKMD = 0x3C:
 - ◆ IHRC frequency is calibrated to 16MHz@VDD=2.5V and IHRC module is enabled
 - ◆ System CLK = IHRC/8 = 2MHz
 - ◆ Watchdog timer is disabled, ILRC is enabled, PA5 is in input mode, BG=1.2V

- (4) .ADJUST_IC SYSCLK=IHRC/16, IHRC=16MHz, VDD=2.5V, Band-gap=On
 After boot up, CLKMD = 0x1C:
 - ◆ IHRC frequency is calibrated to 16MHz@VDD=2.5V and IHRC module is enabled
 - ◆ System CLK = IHRC/16 = 1MHz
 - ◆ Watchdog timer is disabled, ILRC is enabled, PA5 is in input mode, BG=1.2V

- (5) .ADJUST_IC SYSCLK=IHRC/32, IHRC=16MHz, VDD=5V, Band-gap=Off
 After boot up, CLKMD = 0x7C:
 - ◆ IHRC frequency is calibrated to 16MHz@VDD=5V and IHRC module is enabled
 - ◆ System CLK = IHRC/32 = 500KHz
 - ◆ Watchdog timer is disabled, ILRC is enabled, PA5 is in input mode

- (6) .ADJUST_IC SYSCLK=ILRC, IHRC=16MHz, VDD=5V, Band-gap=Off
 After boot up, CLKMD = 0XE4:
 - ◆ IHRC frequency is calibrated to 16MHz@VDD=5V and IHRC module is disabled
 - ◆ System CLK = ILRC
 - ◆ Watchdog timer is enabled, ILRC is enabled, PA5 is input mode

- (7) .ADJUST_IC DISABLE
 After boot up, CLKMD is not changed (Do nothing):
 - ◆ IHRC is not calibrated and IHRC module is disabled, **Band-gap is not calibrated**
 - ◆ System CLK = ILRC
 - ◆ Watchdog timer is enabled, ILRC is enabled, PA5 is in input mode

5-7-4. External Crystal Oscillator

If crystal oscillator is used, a crystal or resonator is required between X1 and X2. Fig. 8 shows the hardware connection under this application; the range of operating frequency of crystal oscillator can be from 32 KHz to 4MHz, depending on the crystal placed on; higher frequency oscillator than 4MHz is NOT supported.

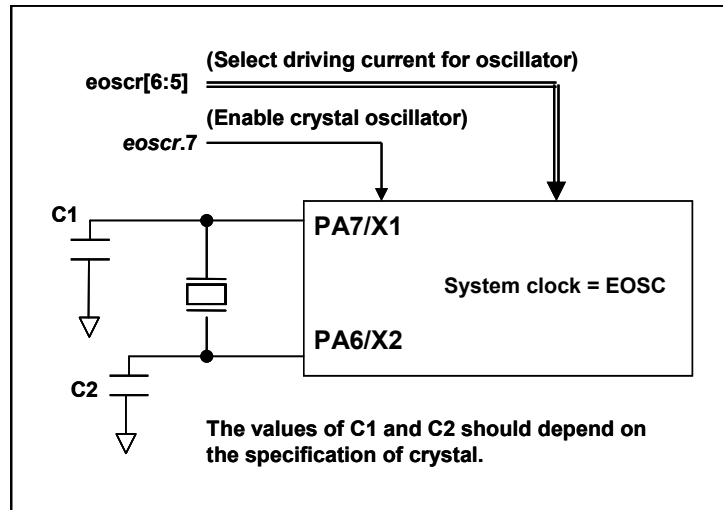


Fig. 8 : Connection of crystal oscillator

Besides crystal, external capacitor and options of PMC234/PMS234 should be fine tuned in *eoscr* (0x0b) register to have good sinusoidal waveform. The *eoscr.7* is used to enable crystal oscillator module, *eoscr.6* and *eoscr.5* are used to set the different driving current to meet the requirement of different frequency of crystal oscillator:

- ◆ *eoscr*[6:5]=01 : Low driving capability, for lower frequency, ex: 32KHz crystal oscillator
- ◆ *eoscr*[6:5]=10 : Middle driving capability, for middle frequency, ex: 1MHz crystal oscillator
- ◆ *eoscr*[6:5]=11 : High driving capability, for higher frequency, ex: 4MHz crystal oscillator

Table 5 shows the recommended values of C1 and C2 for different crystal oscillator; the measured start-up time under its corresponding conditions is also shown. Since the crystal or resonator had its own characteristic, the capacitors and start-up time may be slightly different for different type of crystal or resonator, please refer to its specification for proper values of C1 and C2.

Frequency	C1	C2	Measured Start-up time	Conditions
4MHz	4.7pF	4.7pF	6ms	(<i>eoscr</i> [6:5]=11, <i>misc.6</i> =0)
1MHz	10pF	10pF	11ms	(<i>eoscr</i> [6:5]=10, <i>misc.6</i> =0)
32KHz	22pF	22pF	450ms	(<i>eoscr</i> [6:5]=01, <i>misc.6</i> =0)

Table 5 : Recommend values of C1 and C2 for crystal and resonator oscillators

When using the crystal oscillator, user must pay attention to the stable time of oscillator after enabling it, the stable time of oscillator will depend on frequency ` crystal type ` external capacitor and supply voltage. Before switching the system to the crystal oscillator, user must make sure the oscillator is stable; the reference program is shown as below:

```

void    FPPA0 (void)
{
    .ADJUST_IC  SYSCLK=IHRC/16, IHRC=16MHz, VDD=5V, Band-gap=On
    // If Band-gap is not calibrated, it can use ". ADJUST_IC  DISABLE"

    ...
    $  EOSCR  Enable, 4Mhz; // EOSCR = 0b110_00000;
    $  T16M   EOSC, /1, BIT13; // T16 receive 2^14=16384 clocks
                                of crystal osc.,
                                // Intrq.T16 =>1, crystal osc. Is stable

    WORD  count  =  0;
    stt16count;
    Intrq.T16 =  0;
    wait1  Intrq.T16;           // count fm 0x0000 to 0x2000,
                                then setINTRQ.T16

    clkmd  =  0xA4;           // switch system clock to EOSC;

    ...

```

Please notice that the crystal oscillator should be fully turned off before entering the power-down mode, in order to avoid unexpected wakeup event. If the 32KHz crystal oscillator is used and extremely low operating current is required, *misc.6* can be set to reduce current after crystal oscillator is running normally.

5-7-5. System Clock and LVR level

The clock source of system clock comes from EOSC, IHRC and ILRC, the hardware diagram of system clock in the PMC234/PMS234 is shown as Fig. 9.

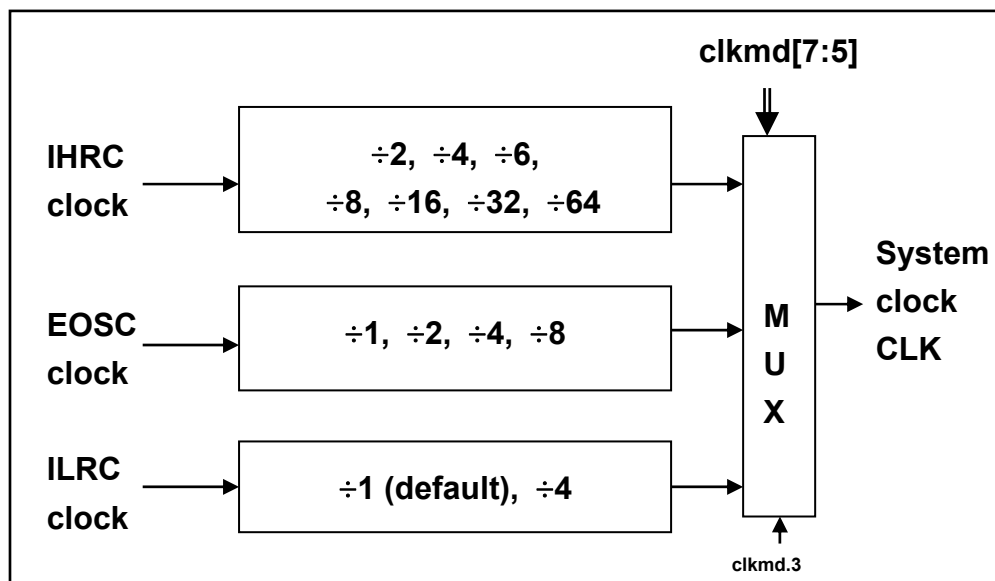


Fig. 9 : Options of System Clock

User can choose different operating system clock depends on its requirement; the selected operating system clock should be combined with supply voltage and LVR level to make system stable. The LVR level will be selected during compilation, the following operating frequency and LVR level is recommended:

- ◆ system clock = 8MHz with LVR=3.1V
- ◆ system clock = 4MHz with LVR=2.5V
- ◆ system clock = 2MHz with LVR=2.2V

5-7-6. System Clock Switching

After IHRC calibration, user may want to switch system clock to a new frequency or may switch system clock at any time to optimize the system performance and power consumption. Basically, the system clock of PMC234/PMS234 can be switched among IHRC, ILRC and EOSC by setting the **clkmd** register at any time; system clock will be the new one after writing to **clkmd** register immediately. Please notice that the original clock module can NOT be turned off at the same time as writing command to **clkmd** register. The examples are shown as below and more information about clock switching, please refer to the “Help -> Application Note -> CLKMD”.

Case 1: Switching system clock from ILRC to IHRC/2

```

...                               // system clock is ILRC
CLKMD      = 0x34 ; // switch to IHRC/2 , ILRC CAN NOT be disabled here
CLKMD.2    = 0 ; // ILRC CAN be disabled at this time
...

```

Case 2: Switching system clock from ILRC to EOSC

```

...                               // system clock is ILRC
CLKMD      = 0xA6 ; // switch to IHRC , ILRC CAN NOT be disabled here
CLKMD.2    = 0 ; // ILRC CAN be disabled at this time
...

```

Case 3: Switching system clock from IHRC/2 to ILRC

```

...                               // system clock is IHRC/2
CLKMD      = 0xF4 ; // switch to ILRC , IHRC CAN NOT be disabled here
CLKMD.4    = 0 ; // IHRC CAN be disabled at this time
...

```

Case 4: Switching system clock from IHRC/2 to EOSC

```

...                               // system clock is IHRC/2
CLKMD      = 0xB0 ; // switch to EOSC , IHRC CAN NOT be disabled here
CLKMD.4    = 0 ; // IHRC CAN be disabled at this time
...

```

Case 5: Switching system clock from IHRC/2 to IHRC/4

```

...                               // system clock is IHRC/2, ILRC is enabled here
CLKMD      = 0X14 ; // switch to IHRC/4
...

```

Case 6: System may hang if it is to switch clock and turn off original oscillator at the same time

```

...                               // system clock is ILRC
CLKMD      = 0x30 ; // CAN NOT switch clock from ILRC to IHRC/2 and
// turn off ILRC oscillator at the same time

```


5-8. 16-bit Timer (Timer16)

A 16-bit hardware timer (Timer16) is implemented in the PMC234/PMS234, the clock sources of Timer16 may come from system clock (CLK), clock of external crystal oscillator (EOSC), internal high RC oscillator (IHRC), internal low RC oscillator (ILRC) and PA0, a multiplex is used to select clock output for the clock source. Before sending clock to the counter16, a pre-scaling logic with divided-by-1, 4, 16, and 64 is used for wide range counting. The 16-bit counter performs up-counting operation only, the counter initial values can be stored from memory by *stt16* instruction and the counting values can be loaded to memory by *ldt16* instruction. A selector is used to select the interrupt condition of Timer16, whenever overflow occurs, the Timer16 interrupt can be triggered. The hardware diagram of Timer16 is shown as Fig. 10. The interrupt source of Timer16 comes from one of bit 8 to 15 of 16-bit counter, and the interrupt type can be rising edge trigger or falling edge trigger which is specified in the bit 4 of *integs* register (IO address 0x0C).

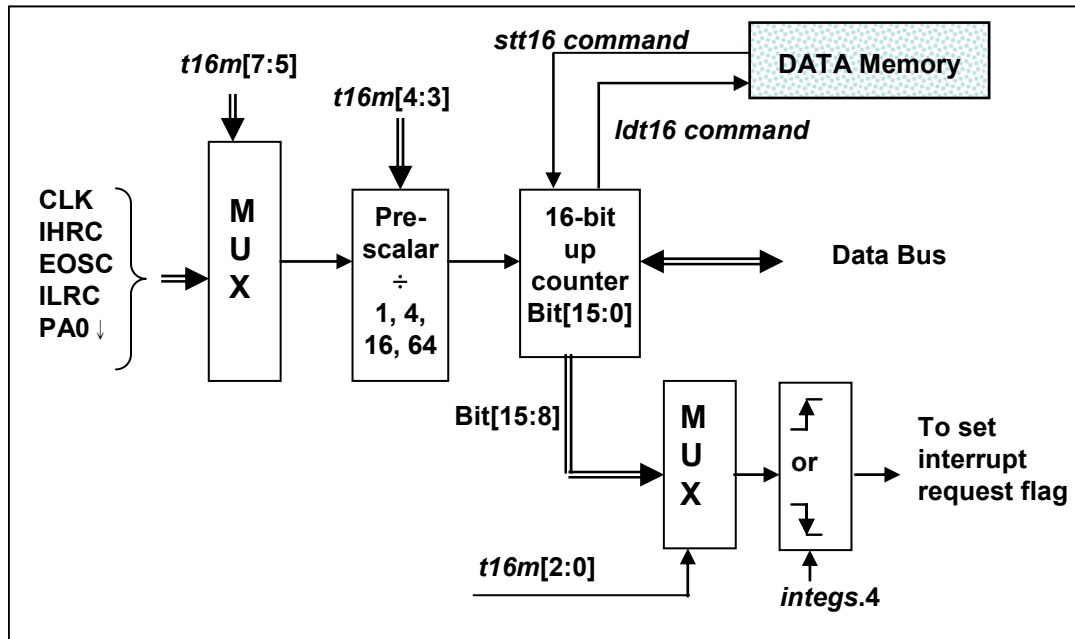


Fig. 10 : Hardware diagram of Timer16

When using the Timer16, the syntax for Timer16 has been defined in the .INC file. There are three parameters to define the Timer16; 1st parameter is used to define the clock source of Timer16, 2nd parameter is used to define the pre-scaler and the last one is to define the interrupt source. The detail description is shown as below:

```

T16M      IO_RW      0x06
$ 7~5:    STOP, SYSCLK, X, X, IHRC, EOSC, ILRC, PA0_F      // 1st par.
$ 4~3:    /1, /4, /16, /64                                  // 2nd par.
$ 2~0:    BIT8, BIT9, BIT10, BIT11, BIT12, BIT13, BIT14, BIT15 // 3rd par.

```

User can define the parameters of T16M based on system requirement, some examples are shown below and more examples can refer to “Help → Application Note → IC Introduction → Register Introduction → T16M” in IDE utility.

```

$ T16M   SYCLK, /64, BIT15;
// choose (SYCLK/64) as clock source, every 2^16 clock to set INTRQ.2=1
// if using System Clock = IHRC / 2 = 8 MHz
// SYCLK/64 = 8 MHz/64 = 125KHz, about every 512 mS to generate INTRQ.2=1

$ T16M   EOSC, /1, BIT13;
// choose (EOSC/1) as clock source, every 2^14 clocks to generate INTRQ.2=1
// if EOSC=32768 Hz, 32768 Hz/(2^14) = 2Hz, every 0.5S to generate INTRQ.2=1

$ T16M   PA0_F, /1, BIT8;
// choose PA0 as clock source, every 2^9 to generate INTRQ.2=1
// receiving every 512 times PA0 to generate INTRQ.2=1

$ T16M   STOP;
// stop Timer16 counting

```

If Timer16 is operated at free running, the frequency of interrupt can be described as below:

$$F_{\text{INTRQ_T16M}} = F_{\text{clock source}} \div P \div 2^{n+1}$$

Where, F is the frequency of selected clock source to Timer16;

P is the selection of t16m [4:3]; (1, 4, 16, 64)

N is the nth bit selected to request interrupt service, for example: n=10 if bit 10 is selected.

5-9. 8-bit Timer (Timer2) with PWM generation

An 8-bit hardware timer (Timer2) with PWM generation is implemented in the PMC234/PMS234, please refer to Fig. 11 shown its hardware diagram, the clock sources of Timer2 may come from system clock, internal high RC oscillator (IHRC), internal low RC oscillator (ILRC), PA0, PA3, PA4 and comparator output, bit [7:4] of register *tm2c* are used to select the clock of Timer2. Please notice that external crystal oscillator is NOT to be the clock of Timer2 because of possible clock glitch. If IHRC is selected for Timer2 clock source, the clock sent to Timer2 will keep running when using ICE in halt state. The output of Timer2 can be sent to pin PA1, PA2 or PA3, depending on bit [3-2] of *tm2c* register. A clock pre-scaling module is provided with divided-by-1, 4, 16, and 64 options, controlled by bit [6:5] of *tm2s* register; one scaling module with divided-by-1~31 is also provided and controlled by bit [4:0] of *tm2s* register. In conjunction of pre-scaling function and scaling function, the frequency of Timer2 clock (TM2_CLK) can be wide range and flexible. TM2_CLK can be selected as system clock in order to provide special frequency of system clock, please refer to *clkmd* register.

The Timer2 counter performs 8-bit up-counting operation only; the counter values can be set or read back by *tm2ct* register. The 8-bit counter will be clear to zero automatically when its values reach for upper bound register, the upper bound register is used to define the period of timer or duty of PWM. There are two operating modes for Timer2: period mode and PWM mode; period mode is used to generate periodical output waveform or interrupt event; PWM mode is used to generate PWM output waveform with optional 6-bit or 8-bit PWM resolution, Fig. 12 shows the timing diagram of Timer2 for both period mode and PWM mode.

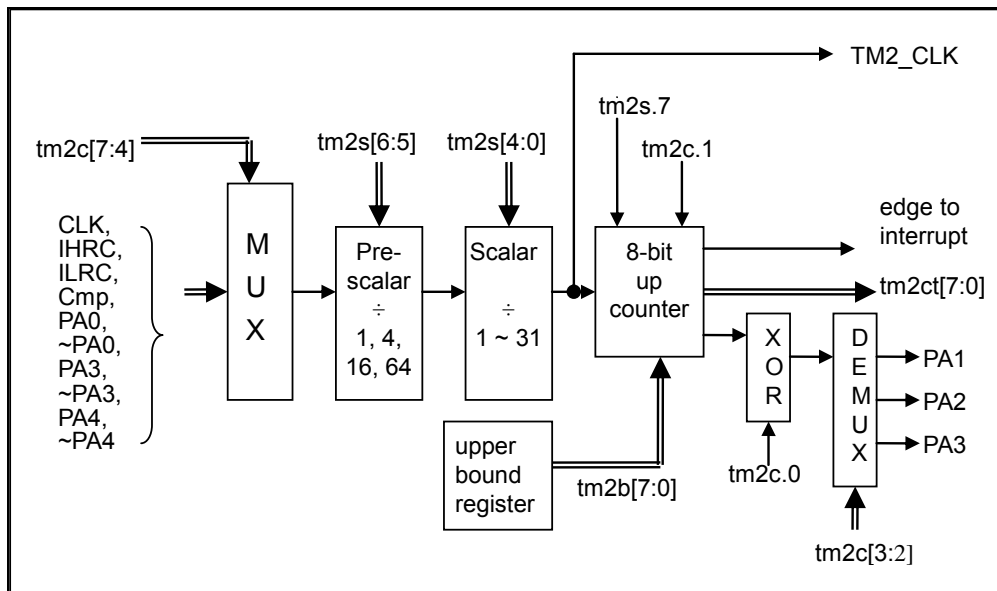


Fig. 11 : Timer2 hardware diagram

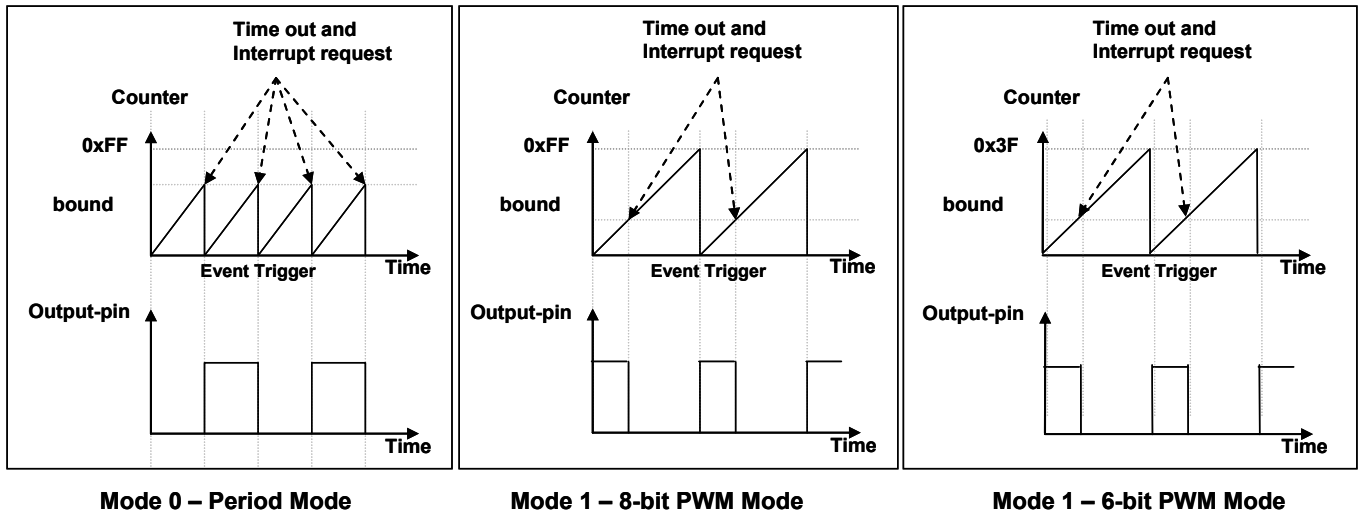


Fig. 12 : Timing diagram of Timer2 in period mode and PWM mode (tm2c.1=1)

5-9-1. Using the Timer2 to generate periodical waveform

If periodical mode is selected, the duty cycle of output is always 50%; its frequency can be summarized as below:

$$\text{Frequency of Output} = Y \div [2 \times (K+1) \times S1 \times (S2+1)]$$

Where, $Y = \text{tm2c}[7:4]$: frequency of selected clock source

$K = \text{tm2b}[7:0]$: bound register in decimal

$S1 = \text{tm2s}[6:5]$: pre-scalar (1, 4, 16, 64)

$S2 = \text{tm2s}[4:0]$: scalar register in decimal (1 ~ 31)

Example 1:

$\text{tm2c} = 0b0001_1100$, $Y=8\text{MHz}$

$\text{tm2b} = 0b0111_1111$, $K=127$

$\text{tm2s} = 0b0_00_00000$, $S1=1$, $S2=0$

→ frequency of output = $8\text{MHz} \div [2 \times (127+1) \times 1 \times (0+1)] = 31.25\text{KHz}$

Example 2:

$\text{tm2c} = 0b0001_1100$, $Y=8\text{MHz}$

$\text{tm2b} = 0b0111_1111$, $K=127$

$\text{tm2s}[7:0] = 0b0_11_11111$, $S1=64$, $S2 = 31$

→ frequency = $8\text{MHz} \div (2 \times (127+1) \times 64 \times (31+1)) = 15.25\text{Hz}$

Example 3:

tm2c = 0b0001_1100, Y=8MHz

tm2b = 0b0000_1111, K=15

tm2s = 0b0_00_00000, S1=1, S2=0

→ frequency = $8\text{MHz} \div (2 \times (15+1) \times 1 \times (0+1)) = 250\text{KHz}$

Example 4:

tm2c = 0b0001_1100, Y=8MHz

tm2b = 0b0000_0001, K=1

tm2s = 0b0_00_00000, S1=1, S2=0

→ frequency = $8\text{MHz} \div (2 \times (1+1) \times 1 \times (0+1)) = 2\text{MHz}$

The sample program for using the Timer2 to generate periodical waveform from PA2 is shown as below:

```
void FPPA0 (void)
{
    . ADJUST_IC SYSCLK=IHRC/2, IHRC=16MHz, VDD=5V
    ...
    tm2ct = 0x0;
    tm2b = 0x7f;
    tm2s = 0b0_00_00001; // 8-bit PWM, pre-scalar = 1, scalar = 2
    tm2c = 0b0001_01_0_0; // system clock, output=PA2, period mode
    while(1)
    {
        nop;
    }
}
```

5-9-2. Using the Timer2 to generate 8-bit PWM waveform

If 8-bit PWM mode is selected, it should set $tm2c[1]=1$ and $tm2s[7]=0$, the frequency and duty cycle of output waveform can be summarized as below:

$$\text{Frequency of Output} = Y \div [256 \times S1 \times (S2+1)]$$

$$\text{Duty of Output} = (K + 1) \div 256$$

Where, $Y = tm2c[7:4]$: frequency of selected clock source

$K = tm2b[7:0]$: bound register in decimal

$S1 = tm2s[6:5]$: pre-scalar (1, 4, 16, 64)

$S2 = tm2s[4:0]$: scalar register in decimal (1 ~ 31)

Example 1:

$tm2c = 0b0001_1110$, $Y=8\text{MHz}$

$tm2b = 0b0111_1111$, $K=127$

$tm2s = 0b0_00_00000$, $S1=1$, $S2=0$

→ frequency of output = $8\text{MHz} \div (256 \times 1 \times (0+1)) = 31.25\text{KHz}$

→ duty of output = $[(127+1) \div 256] \times 100\% = 50\%$

Example 2:

$tm2c = 0b0001_1110$, $Y=8\text{MHz}$

$tm2b = 0b0111_1111$, $K=127$

$tm2s = 0b0_11_11111$, $S1=64$, $S2=31$

→ frequency of output = $8\text{MHz} \div (256 \times 64 \times (31+1)) = 15.25\text{Hz}$

→ duty of output = $[(127+1) \div 256] \times 100\% = 50\%$

Example 3:

$tm2c = 0b0001_1110$, $Y=8\text{MHz}$

$tm2b = 0b1111_1111$, $K=255$

$tm2s = 0b0_00_00000$, $S1=1$, $S2=0$

→ frequency of output = $8\text{MHz} \div (256 \times 1 \times (0+1)) = 31.25\text{KHz}$

→ duty of output = $[(255+1) \div 256] \times 100\% = 100\%$

Example 4:

$tm2c = 0b0001_1110$, $Y=8\text{MHz}$

$tm2b = 0b0000_1001$, $K = 9$

$tm2s = 0b0_00_00000$, $S1=1$, $S2=0$

→ frequency of output = $8\text{MHz} \div (256 \times 1 \times (0+1)) = 31.25\text{KHz}$

→ duty of output = $[(9+1) \div 256] \times 100\% = 3.9\%$

The sample program for using the Timer2 to generate PWM waveform from PA2 is shown as below:

```

void  FPPA0 (void)
{
    ..ADJUST_IC SYSCLK=IHRC/2, IHRC=16MHz, VDD=5V
    wdreset;
    tm2ct = 0x0;
    tm2b = 0x7f;
    tm2s = 0b0_00_00001;    // 8-bit PWM, pre-scalar = 1, scalar = 2
    tm2c = 0b0001_01_1_0;  // system clock, output=PA2, PWM mode
    while(1)
    {
        nop;
    }
}

```

5-9-3. Using the Timer2 to generate 6-bit PWM waveform

If 6-bit PWM mode is selected, it should set *tm2c*[1]=1 and *tm2s*[7]=1, the frequency and duty cycle of output waveform can be summarized as below:

$$\text{Frequency of Output} = Y \div [64 \times S1 \times (S2+1)]$$

$$\text{Duty of Output} = [(K + 1) \div 64] \times 100\%$$

Where, *tm2c*[7:4] = Y : frequency of selected clock source

tm2b[7:0] = K : bound register in decimal

tm2s[6:5] = S1 : pre-scalar (1, 4, 16, 64)

tm2s[4:0] = S2 : scalar register in decimal (1 ~ 31)

Example 1:

tm2c = 0b0001_1110, Y=8MHz

tm2b = 0b0001_1111, K=31

tm2s = 0b1_00_00000, S1=1, S2=0

→ frequency of output = 8MHz ÷ (64 × 1 × (0+1)) = 125KHz

→ duty = [(31+1) ÷ 64] × 100% = 50%

Example 2:

tm2c = 0b0001_1110, Y=8MHz

tm2b = 0b0001_1111, K=31

tm2s = 0b1_11_11111, S1=64, S2=31

→ frequency of output = 8MHz ÷ (64 × 64 × (31+1)) = 61.03 Hz

→ duty of output = [(31+1) ÷ 64] × 100% = 50%

Example 3:

```

tm2c = 0b0001_1110, Y=8MHz
tm2b = 0b0011_1111, K=63
tm2s = 0b1_00_00000, S1=1, S2=0
→ frequency of output = 8MHz ÷ ( 64 × 1 × (0+1) ) = 125KHz
→ duty of output = [(63+1) ÷ 64] × 100% = 100%

```

Example 4:

```

tm2c = 0b0001_1110, Y=8MHz
tm2b = 0b0000_0000, K=0
tm2s = 0b1_00_00000, S1=1, S2=0
→ frequency = 8MHz ÷ ( 64 × 1 × (0+1) ) = 125KHz
→ duty = [(0+1) ÷ 64] × 100% = 1.5%

```

5-10. WatchDog Timer

The watchdog timer (WDT) is a counter with clock coming from ILRC and there are four different timeout periods of watchdog timer can be chosen by setting the *misc* register, it is:

- ◆ 256 ILRC clocks period if register misc[1:0]=11
- ◆ 2048 ILRC clocks period if register misc[1:0]=00 (default)
- ◆ 4096 ILRC clocks period if register misc[1:0]=01
- ◆ 16384 ILRC clocks period if register misc[1:0]=10

The frequency of ILRC may drift a lot due to the variation of manufacture, supply voltage and temperature; user should reserve guard band for save operation. WDT can be cleared by power-on-reset or by command *wdreset* at any time. When WDT is timeout, PMC234/PMS234 will be reset to restart the program execution. The relative timing diagram of watchdog timer is shown as Fig. 13.

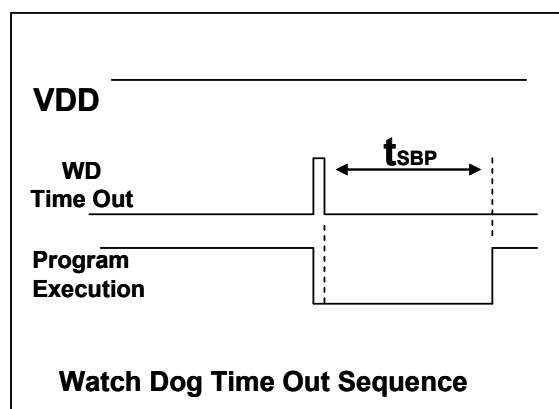


Fig. 13 : Sequence of Watch Dog Time Out

5-11. Interrupt

There are six interrupt lines for PMC234/PMS234:

- ◆ External interrupt PA0
- ◆ External interrupt PB0
- ◆ ADC interrupt
- ◆ Timer16 interrupt
- ◆ Timer2 interrupt
- ◆ Comparator interrupt,

Every interrupt request line has its own corresponding interrupt control bit to enable or disable it; the hardware diagram of interrupt function is shown as Fig. 14. All the interrupt request flags are set by hardware and cleared by writing *intrq* register. When the request flags are set, it can be rising edge, falling edge or both, depending on the setting of register *integs*. All the interrupt request lines are also controlled by *engint* instruction (enable global interrupt) to enable interrupt operation and *disgint* instruction (disable global interrupt) to disable it. Only FPP0 can accept the interrupt request, other FPP unit will not be interfered by interrupt.

The stack memory for interrupt is shared with data memory and its address is specified by stack register *sp*. Since the program counter is 16 bits width, the bit 0 of stack register *sp* should be kept 0. Moreover, user can use *pushaf* / *popaf* instructions to store or restore the values of *ACC* and *flag* register *to* / *from* stack memory. Since the stack memory is shared with data memory, user should manipulate the memory using carefully. By adjusting the memory location of stack point, the depth of stack pointer for every FPP unit could be fully specified by user to achieve maximum flexibility of system.

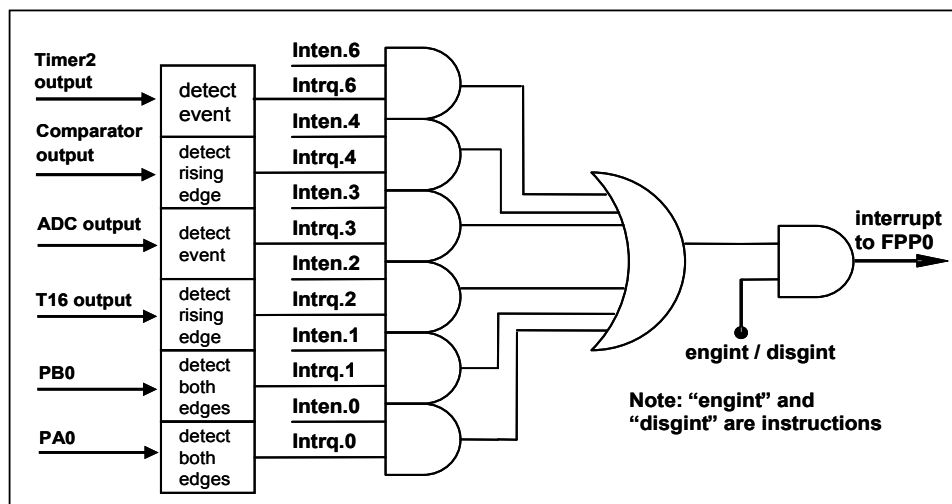


Fig. 14 : Hardware diagram of interrupt controller

Once the interrupt occurs, its operation will be:

- ◆ The program counter will be stored automatically to the stack memory specified by register *sp*.
- ◆ New *sp* will be updated to *sp+2*.
- ◆ Global interrupt will be disabled automatically.
- ◆ The next instruction will be fetched from address 0x10.

During the interrupt service routine, the interrupt source can be determined by reading the *intrq* register.

After finishing the interrupt service routine and issuing the *reti* instruction to return back, its operation will be:

- ◆ The program counter will be restored automatically from the stack memory specified by register *sp*.
- ◆ New *sp* will be updated to *sp-2*.
- ◆ Global interrupt will be enabled automatically.

The next instruction will be the original one before interrupt.

User must reserve enough stack memory for interrupt, two bytes stack memory for one level interrupt and four bytes for two levels interrupt. For interrupt operation, the following sample program shows how to handle the interrupt, noticing that it needs four bytes stack memory to handle interrupt and *pushaf*.

```

void      FPPA0  (void)
{
    ...
    $ INTEN PA0;           // INTEN =1; interrupt request when PA0 level changed
    INTRQ = 0;            // clear INTRQ
    ENGINT                // global interrupt enable
    ...
    DISGINT               // global interrupt disable
    ...
}

void      Interrupt (void) // interrupt service routine
{
    PUSHAF                // store ALU and FLAG register
    If (INTRQ.0)          // Here for PA0 interrupt service routine
    {
        INTRQ.0 = 0;
        ...
    }
    ...
    POPAF                 // restore ALU and FLAG register
}

```

5-12. Power-Save and Power-Down

There are three operational modes defined by hardware: ON mode, Power-Save mode and Power-Down modes. ON mode is the state of normal operation with all functions ON, Power-save mode (“**stopexe**”) is the state to reduce operating current and CPU keeps ready to continue, Power-Down mode (“**stopsys**”) is used to save power deeply. Therefore, Power-save mode is used in the system which needs low operating power with wake-up occasionally and Power-Down mode is used in the system which needs power down deeply with seldom wake-up. Fig. 15 shows the differences in oscillator modules between Power-Save mode (“**stopexe**”) and Power-Down mode (“**stopsys**”).

Differences in oscillator modules between STOPSYS and STOPEXE			
	IHRC	ILRC	EOSC
STOPSYS	Stop	Stop	Stop
STOPEXE	No Change	No Change	No Change

Fig. 15 : Differences in oscillator modules between STOPSYS and STOPEXE

5-12-1. Power-Save mode (“**stopexe**”)

Using “**stopexe**” instruction to enter the Power-Save mode, only system clock is disabled, remaining all the oscillator modules active. For CPU, it stops executing; however, for Timer16, counter keep counting if its clock source is not the system clock. The wake-up sources for “**stopexe**” can be IO-toggle or Timer16 counts to the set values when clock sources of Timer16 come from IHRC, ILRC or EOSC modules. Wake-up from input pins can be considered as a continuation of normal execution, **nop** command is recommended to follow the **stopexe** command, the detail information for Power-Save mode shows below:

- IHRC, ILRC and EOSC oscillator modules: No change, keep active if it was enabled
- System clock: Disable, therefore, CPU stops execution
- OTP memory is turned off
- Timer16: Stop counting if system clock is selected or the corresponding oscillator module is disabled; otherwise, it keeps counting.
- Wake-up sources: IO toggle or Timer16.

The watchdog timer must be disabled before issuing the “**stopexe**” command, the example is shown as below:

```

CLKMD.En_WatchDog = 0;      // disable watchdog timer
stopexe;
nop;
....                          // power saving
Wdreset;
CLKMD.En_WatchDog = 1;      // enable watchdog timer

```

Another example shows how to use Timer16 to wake-up from “**stopexe**”:

```

$ T16M  IHRC, /1, BIT8      // Timer16 setting
...
WORD   count   = 0;
STT16  count;
stopexe;
nop;
...

```

The initial counting value of Timer16 is zero and the system will be waken up after the Timer16 counts 256 IHRC clocks.

5-12-2. Power-Down mode (“*stopsys*”)

Power-Down mode is the state of deeply power-saving with turning off all the oscillator modules. By using the “*stopsys*” instruction, this chip will be put on Power-Down mode directly. The internal low frequency RC oscillator must be enabled before entering the Power-Down mode, means that bit 2 of register ***clkmd*** (0x03) must be set to high before issuing “*stopsys*” command in order to resume the system when wakeup. The following shows the internal status of PMC234/PMS234 in detail when “*stopsys*” command is issued:

- All the oscillator modules are turned off
- Enable internal low RC oscillator (set bit 2 of register ***clkmd***)
- OTP memory is turned off
- The contents of SRAM and registers remain unchanged
- Wake-up sources: ANY IO toggle.
- If PA or PB is input mode and set to analog input by ***padier*** or ***pbdir*** register, it can NOT be used to Wake-up the system.

Wake-up from input pins can be considered as a continuation of normal execution. To minimize power consumption, all the I/O pins should be carefully manipulated before entering power-down mode. The reference sample program for power down is shown as below:

```

CMKMD = 0xF4; // Change clock from IHRC to ILRC
CLKMD.4 = 0; // disable IHRC
...
while (1)
{
    STOPSYS; // enter power-down
    if (...) break; // if wakeup happen and check OK, then return to high speed,
                    // else stay in power-down mode again
}
CLKMD = 0x34; // Change clock from ILRC to IHRC/2

```

5-12-3. Wake-up

After entering the Power-Down or Power-Save modes, the PMC234/PMS234 can be resumed to normal operation by toggling IO pins, Timer16 interrupt is available for Power-Save mode ONLY. Fig. 16 shows the differences in wake-up sources between STOPSYS and STOPEXE.

Differences in wake-up sources between STOPSYS and STOPEXE		
	IO Toggle	T16 Interrupt
STOPSYS	Yes	No
STOPEXE	Yes	Yes

Fig. 16 : Differences in wake-up sources between Power-Save mode and Power-Down mode

When using the IO pins to wake-up the PMC234/PMS234, registers *padier* and *pbdiar* should be properly set to enable the wake-up function for every corresponding pin. The wake-up time for normal wake-up is about 1024 ILRC clocks counting from wake-up event; fast wake-up can be selected to reduce the wake-up time by *misc* register. For fast wake-up mechanism, the wake-up time is 128 system clocks from IO toggling if STOPEXE was issued, and 128 system clocks plus oscillator (IHRC or ILRC) stable time from IO toggling if STOPSYS was issued. The oscillator stable time is the time for IHRC or ILRC oscillator from power-on, depending on which oscillator is used as system clock source. Please notice that the fast wake-up will turn off automatically when EOSC is selected as system clock.

Suspend mode	Wake-up mode	System clock source	Wake-up time (t_{WUP}) from IO toggle
STOPEXE suspend	fast wake-up	IHRC or ILRC	$128 * T_{SYS}$, Where T_{SYS} is the time period of system clock
STOPSYS suspend	fast wake-up	IHRC	$128 T_{SYS} + T_{SIHRC}$; Where T_{SIHRC} is the stable time of IHRC from power-on.
STOPSYS suspend	fast wake-up	ILRC	$128 T_{SYS} + T_{SILRC}$; Where T_{SILRC} is the stable time of ILRC from power-on.
STOPSYS or STOPEXE suspend	fast wake-up	EOSC	$1024 * T_{ILRC}$, Where T_{ILRC} is the clock period of ILRC
STOPEXE suspend	normal wake-up	Any one	$1024 * T_{ILRC}$, Where T_{ILRC} is the clock period of ILRC
STOPSYS suspend	normal wake-up	Any one	$1024 * T_{ILRC}$, Where T_{ILRC} is the clock period of ILRC

Please notice that the clock source of watch-dog will be switched to system clock (for example: 4MHz) when fast wakeup is enabled. Therefore, for fast wake-up, recommending turning off the watchdog timer **before** enabling the fast wakeup. After wake-up, the watchdog timer is turned on **after** disabling the fast wakeup.

5-13. IO Pins

Other than PA5, all the pins can be independently set into two states output or input by configuring the data registers (*pa*, *pb*, *pc*, *pd*), control registers (*pac*, *pbc*, *pcc*, *pd*) and pull-high registers (*paph*, *pbph*, *pcph*, *pdph*). All these pins have Schmitt-trigger input buffer and output driver with CMOS level. When it is set to output low, the pull-up resistor is turned off automatically. If user wants to read the pin state, please notice that it should be set to input mode before reading the data port; if user reads the data port when it is set to output mode, the reading data comes from data register, NOT from IO pad. As an example, Table 6 shows the configuration table of bit 0 of port A. The hardware diagram of IO buffer is also shown as Fig. 17.

<i>pa.0</i>	<i>pac.0</i>	<i>paph.0</i>	Description
X	0	0	Input without pull-up resistor
X	0	1	Input with pull-up resistor
0	1	X	Output low without pull-up resistor
1	1	0	Output high without pull-up resistor
1	1	1	Output high with pull-up resistor

Table 6 : PA0 Configuration Table

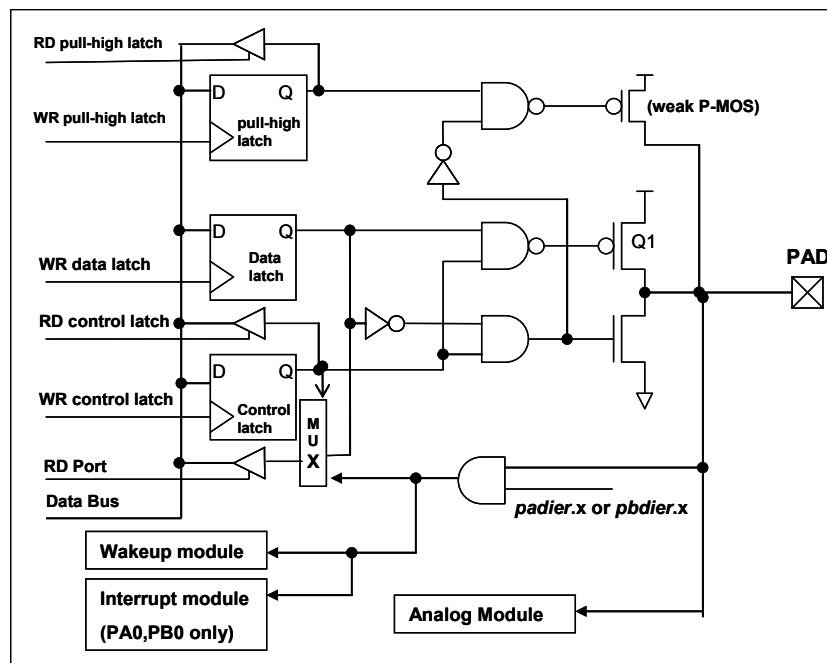


Fig. 17 : Hardware diagram of IO buffer

PA5 can be open-drain ONLY when setting to output mode (without Q1). The corresponding bits in registers *padier* / *pbdier* should be set to low to prevent leakage current for port A and port B pins which are selected to be analog function. When PMC234/PMS234 is put in power-down or power-save mode, every pin can be used to wake-up system by toggling its state. Therefore, those pins needed to wake-up system must be set to input mode and set the corresponding bits of registers *padier* and *pbdier* to high. The same reason, *padier.0* should be set high when PA0 is used as external interrupt pin and *pbdier.0* for PB0.

5-14. Reset and LVR

5-14-1. Reset

There are many causes to reset the PMC234/PMS234, once reset is asserted, most of all the registers in PMC234/PMS234 will be set to default values, When reset comes from WDT timeout, *gdi* register (IO address 0x7) keeps the same value, system should be restarted once abnormal cases happen, or by jumping program counter to address 0x0. The data memory is in uncertain state when reset comes from power-up and LVR; however, the content will be kept when reset comes from PRST# pin or WDT timeout.

5-14-2. LVR (low voltage reset)

By code option, there are 8 different levels of LVR for reset ~ 4.1V, 3.6V, 3.1V, 2.8V, 2.5V, 2.2V, 2.0V and 1.8V; usually, user selects LVR level to be in conjunction with operating frequency and supply voltage.

5-15. Comparator

5-15-1. Comparator Hardware Diagram

One comparator is built inside the PMC234/PMS234; Fig. 18 shows its hardware diagram. It can compare signals between two pins or with either internal reference voltage $V_{\text{internal R}}$ or internal band-gap reference voltage. The two signals to be compared, one will be the plus input of comparator and the other one is the minus input of comparator. For the minus input of comparator, it can be PC0/CIN3-, PC5/CIN+, Internal band-gap 1.20 volt, PA3/CIN1-, PA4/CIN2- or $V_{\text{internal R}}$ selected by bit [3:1] of *gpcc* register, and for the plus input of comparator, it can be PC5/CIN+ or $V_{\text{internal R}}$ selected by bit 0 of *gpcc* register. The comparator result can be enabled to output to PB0 directly, or sampled by rising edge of Time2 clock (TM2_CLK) which comes from Timer2 module. The output can be optional inverted the polarity by bit 4 of *gpcc* register, the comparator output can be used to request interrupt service or read out by *gpcc* register.

The comparator is disabled after power-on reset and can be enabled by setting *gpcc.7=1*, the comparator module can be put into power-down mode only when issuing **stopsys** command which will put PMC234/PMS234 into power-down mode.

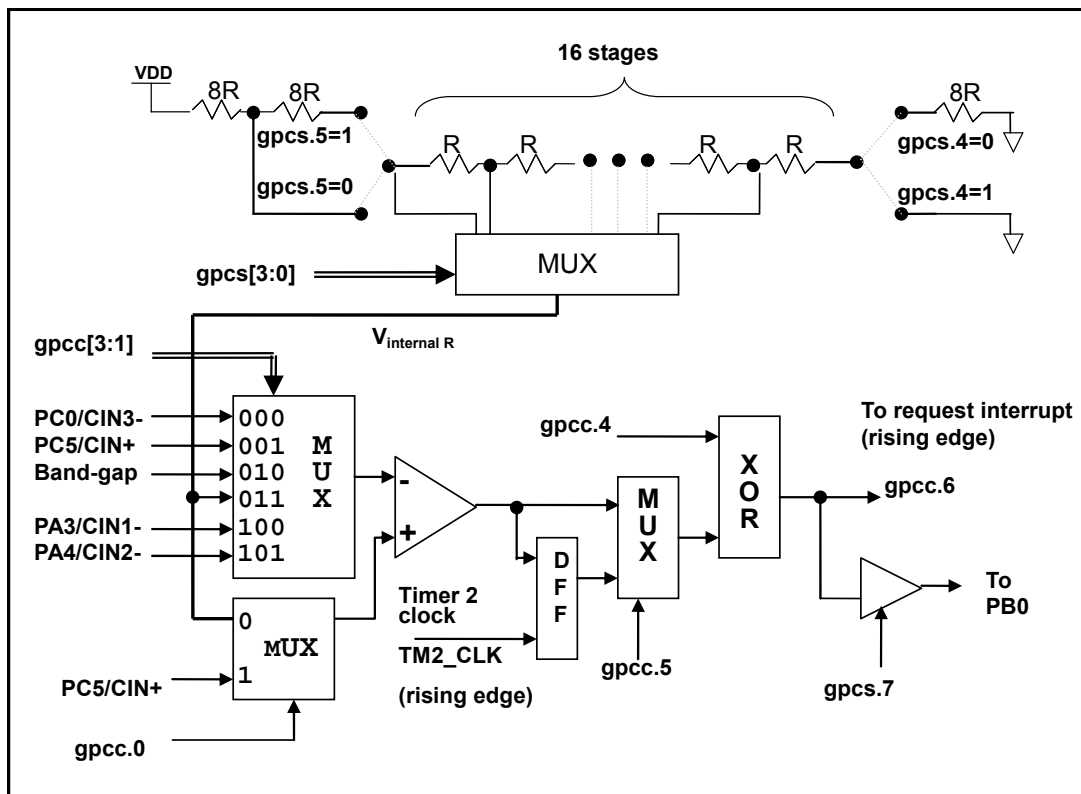


Fig. 18 : Hardware diagram of comparator

5-15-2. Analog Inputs

A simplified circuit for the analog inputs is shown in the Fig. 19. All the analog input pins for comparator are shared function with a digital input which had reverse biased ESD protection diodes to VDD and GND, therefore, the analog input signal must be between VDD and GND.

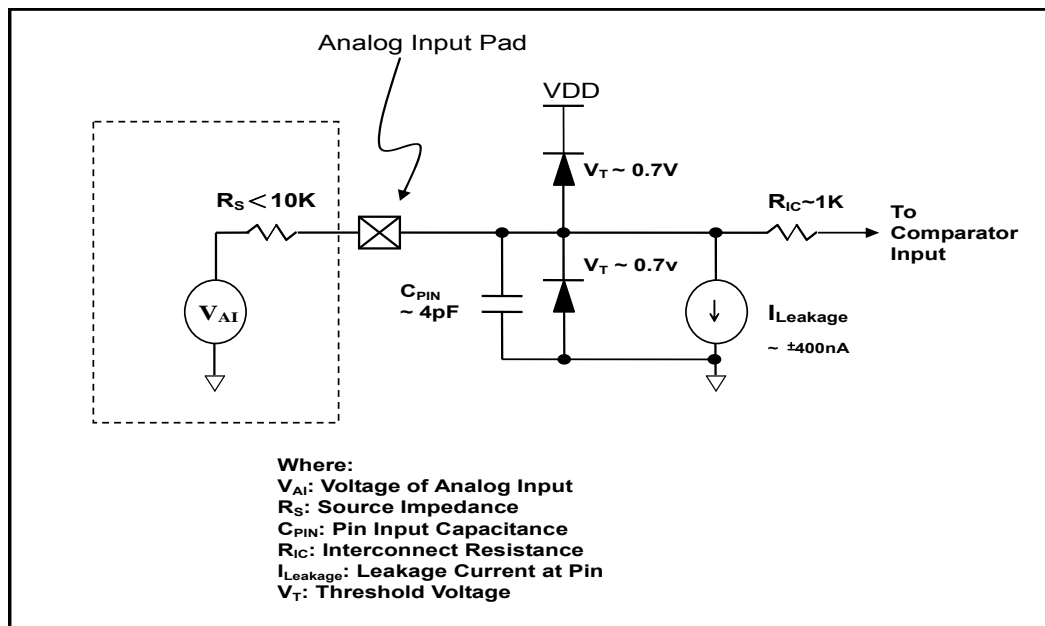


Fig. 19 Analog Input Model of Comparator

5-15-3. Internal reference voltage ($V_{internal R}$)

The internal reference voltage $V_{internal R}$ is built by series resistance to provide different level of reference voltage, bit 4 and bit 5 of **gpcs** register are used to select the maximum and minimum values of $V_{internal R}$ and bit [3:0] of **gpcs** register are used to select one of the voltage level which is divided-by-16 from the defined maximum level to minimum level. Fig. 20 to Fig. 23 shows four conditions to have different reference voltage $V_{internal R}$. By setting the **gpcs** register, the internal reference voltage $V_{internal R}$ can be ranged from $(1/32)*VDD$ to $(3/4)*VDD$.

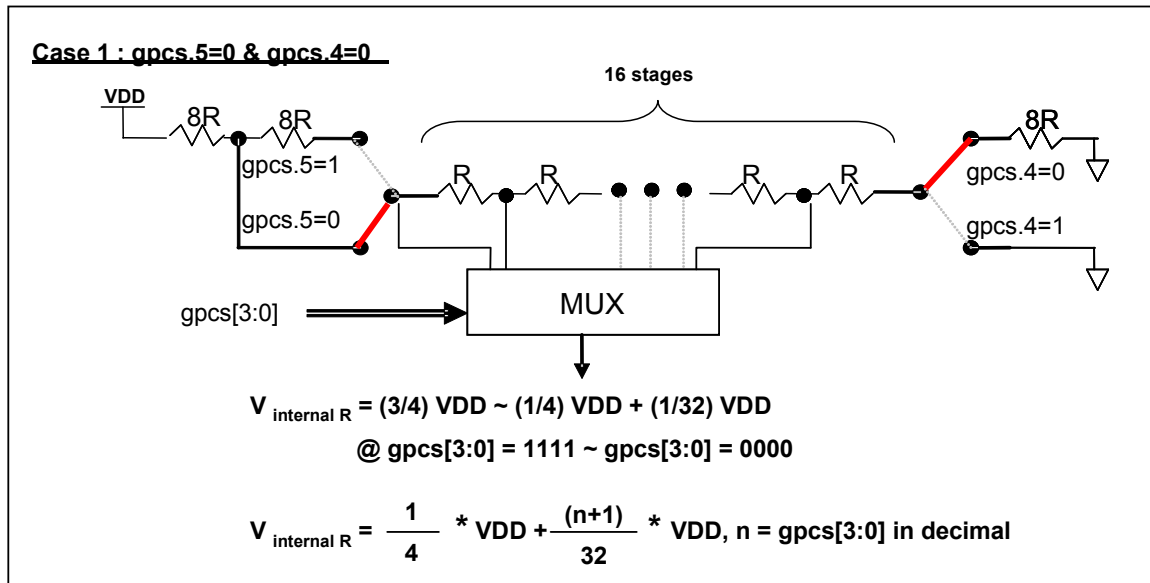


Fig. 20 : $V_{\text{internal R}}$ hardware connection if gpcs.5=0 and gpcs.4=0

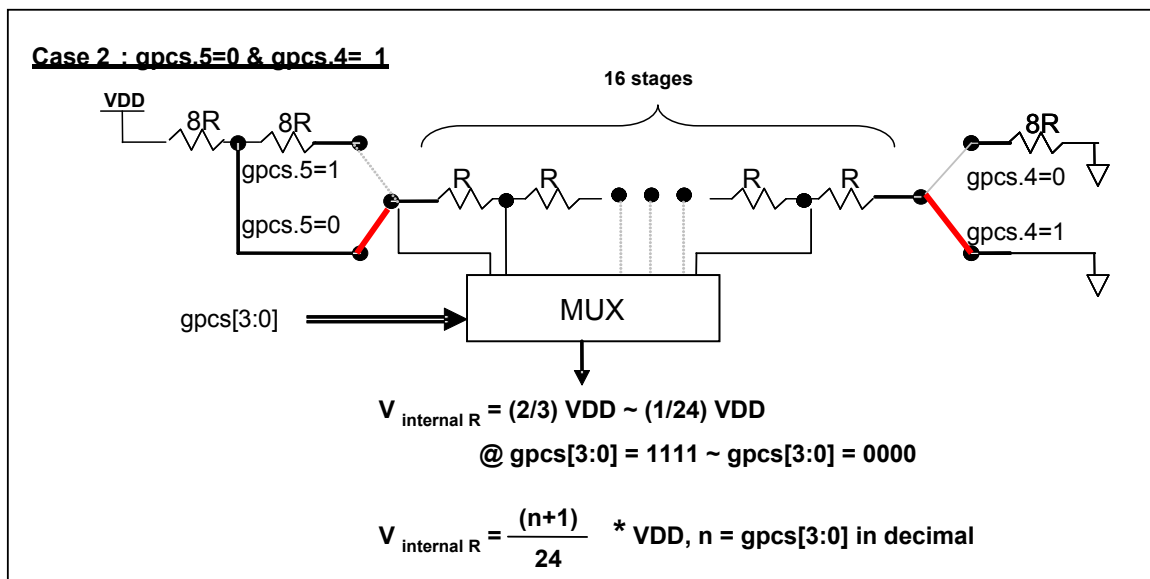


Fig. 21 : $V_{\text{internal R}}$ hardware connection if gpcs.5=0 and gpcs.4=1

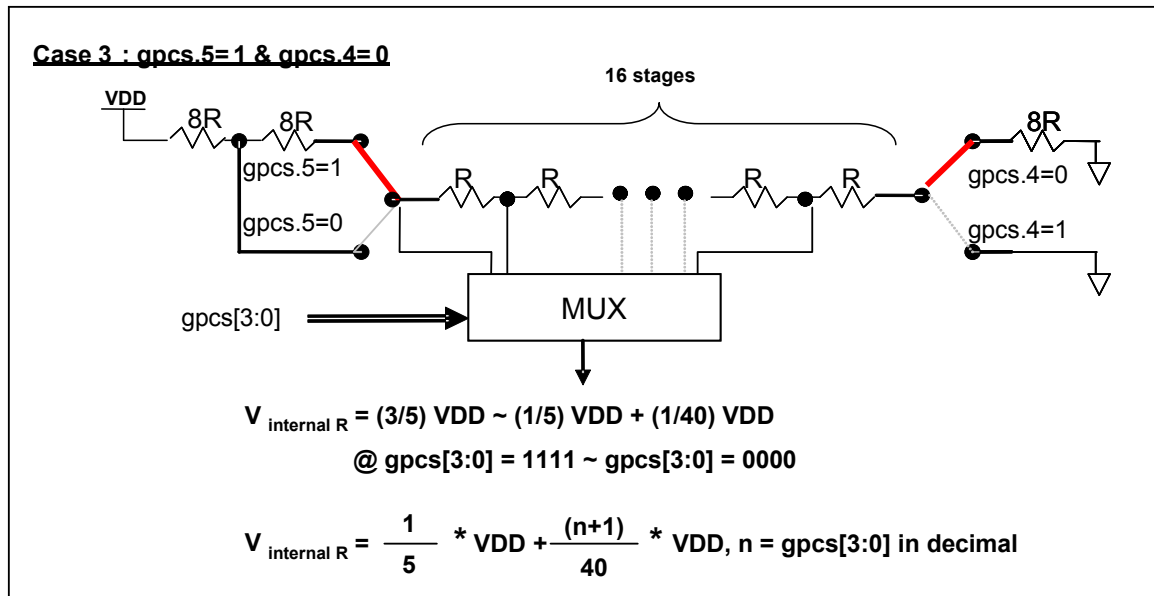


Fig. 22 : $V_{\text{internal R}}$ hardware connection if gpcs.5=1 and gpcs.4=0

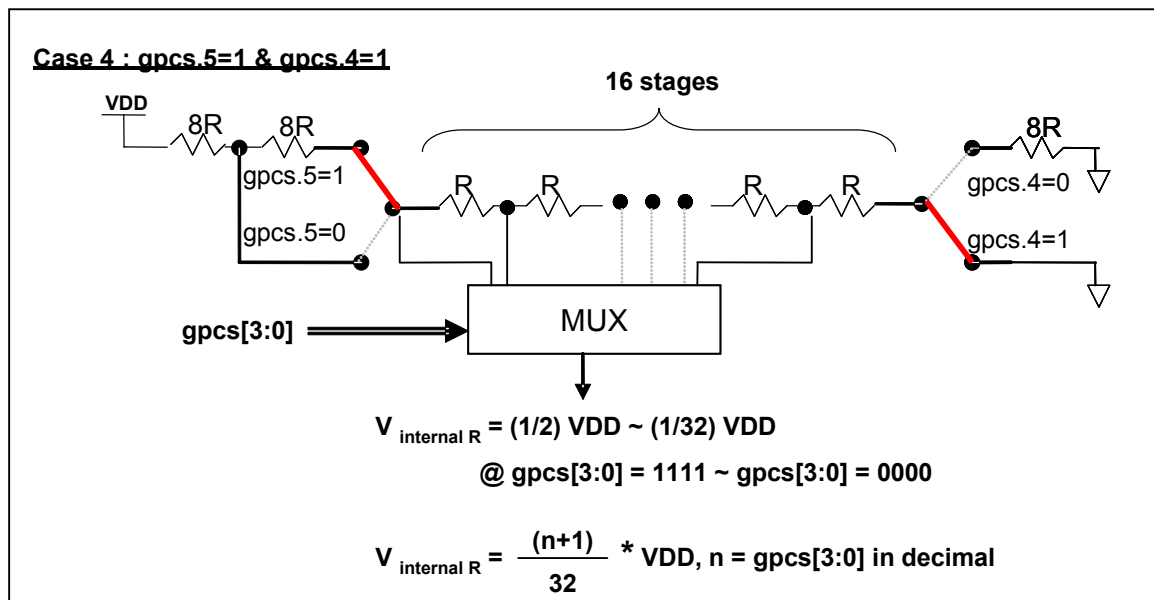


Fig. 23 : $V_{\text{internal R}}$ hardware connection if gpcs.5=1 and gpcs.4=1

5-15-4. Comparator Interrupt Operation

Interrupt request flag *intrq.4* will be set to request interrupt service whenever the rising edge of comparator output *gpcc.6* is detected, this interrupt request flag will be latched until cleared by software via *intrq* register. The interrupt service routine handling is similar to other interrupt request lines. The relative hardware timing diagram is shown as Fig. 24.

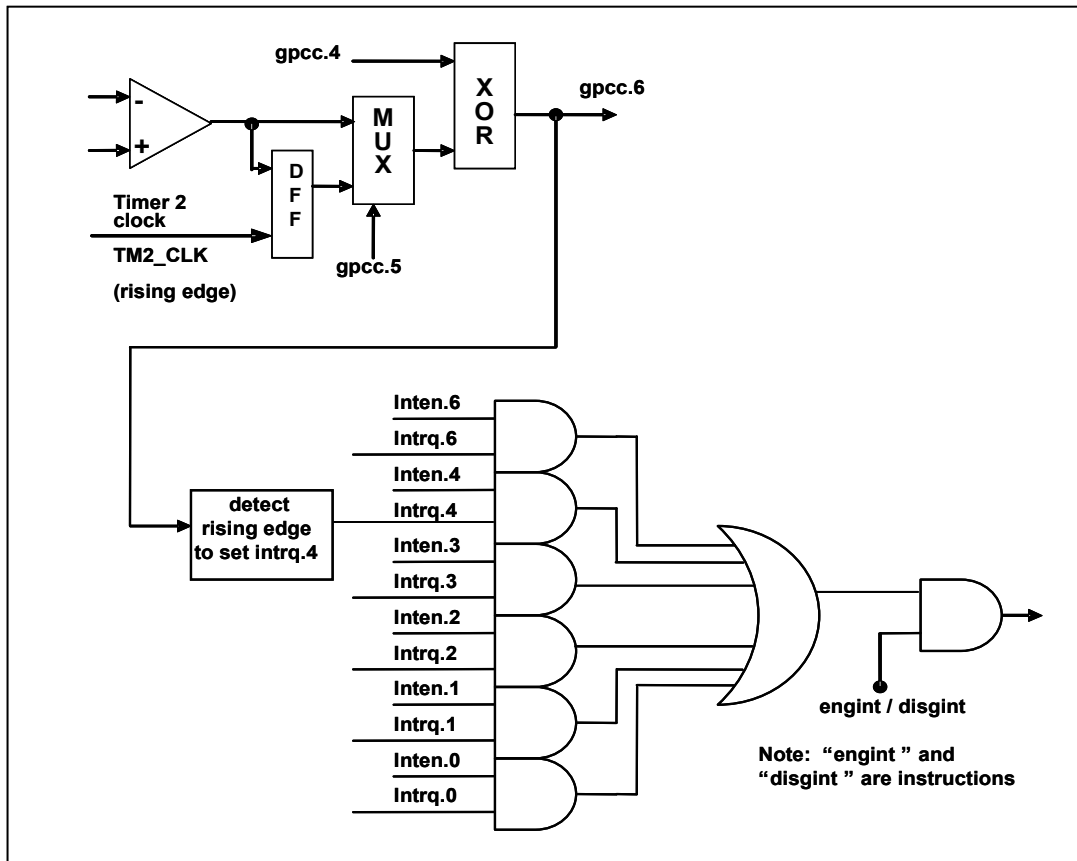


Fig. 24 : Hardware Diagram of Comparator Interrupt Operation

5-15-5. Synchronizing Comparator Output to Timer2

The comparator output can be synchronized with Timer2 by setting *gpcc.5*=1. When enabled, the comparator output is sampled by the rising edge of Timer2 clock source (TM2_CLK). If the pre-scaler function is used with Timer2, the comparator output is sampled after the pre-scaling and scaling functions, Please refer to Fig. 10 Timer2 hardware diagram and Fig. 15 Comparator hardware diagram, TM2_CLK is the clock source after pre-scaling and scaling functions, and will be sent to Timer2 counter for counting and comparator for sampling clock.

5-15-6. Comparator Response Time

The comparator output is undetermined for a period of time after changing an input source or a new reference voltage; this period is referred as the response time. Please refer to Fig. 4-31 and Fig. 4-32 the measurement of comparator responsive time. Typically, the measurement will be based on the compared voltage level near $(V_{DD}-1.5)/2$. If the compared voltage level is not within the range, the responsive time may be longer.

5-15-7. Using the comparator

Case 1:

Choosing PC0/CIN3- as minus input and $V_{internal R}$ with $(18/32)*V_{DD}$ voltage level as plus input, the comparator result will be output to PB0, $V_{internal R}$ is configured as Fig. 16 and `gpcs [3:0] = 4b'1001` (n=9) to have $V_{internal R} = (1/4)*V_{DD} + [(9+1)/32]*V_{DD} = (18/32)*V_{DD}$.

```
gpcs    = 0b1_0_00_1001;      // output to PB0,  $V_{internal R} = V_{DD}*(18/32)$ 
gpcc    = 0b1_0_0_0_000_0;    // enable comp, - input: PC0/CIN3-, + input:  $V_{internal R}$ 
```

Case 2:

Choosing $V_{internal R}$ as minus input with $(14/32)*V_{DD}$ voltage level and PC5/CIN+ as plus input, the comparator result will be inversed and then output to PB0. $V_{internal R}$ is configured as Fig. 19 and `gpcs [3:0] = 4b'1101` (n=13) to have $V_{internal R} = [(13+1)/32]*V_{DD} = (14/32)*V_{DD}$.

```
gpcs    = 0b1_1_1_1_1101;      //  $V_{internal R} = V_{DD}*(14/32)$ 
gpcc    = 0b1_0_0_1_011_1;    // Inverse output, - input:  $V_{internal R}$ , + input: PC5/CIN+
```

5-15-8. Using the comparator and band-gap 1.20V

The internal band-gap module can provide 1.20 volt, it can measure the external supply voltage level. The band-gap 1.20 volt is selected as minus input of comparator and $V_{\text{internal R}}$ is selected as plus input, the supply voltage of $V_{\text{internal R}}$ is VDD, the VDD voltage level can be detected by adjusting the voltage level of $V_{\text{internal R}}$ to compare with band-gap. If N (gpcs[3:0] in decimal) is the number to let $V_{\text{internal R}}$ closest to band-gap 1.20 volt, the supply voltage VDD can be calculated by using the following equations:

For using Case 1: $VDD = [32 / (N+9)] * 1.20 \text{ volt ;}$

For using Case 2: $VDD = [24 / (N+1)] * 1.20 \text{ volt ;}$

For using Case 3: $VDD = [40 / (N+9)] * 1.20 \text{ volt ;}$

For using Case 4: $VDD = [32 / (N+1)] * 1.20 \text{ volt ;}$

More informations and sample codes, please refer to IDE utility.

5-16. LCD Bias Voltage Generator

This function can be enabled by bit 4 of *misc* register. Those pins which are defined to output VDD/2 voltage are PA3、PA2、PA1、PA0 during input mode, being used as COM function for LCD application. If user wants to output VDD、VDD/2、GND three levels voltage, the corresponding pins must be set to output-high for VDD, enabling VDD/2 bias voltage with input mode for VDD/2, and output-low for GND correspondingly, Fig. 25 shows how to use this function.

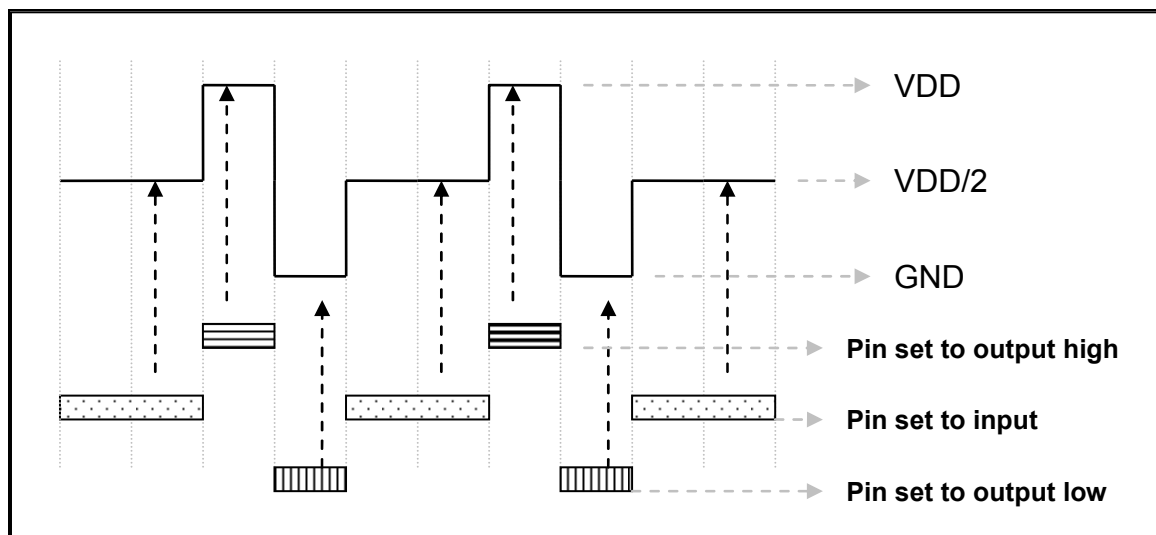


Fig. 25 : Using VDD/2 bias voltage generator

5-17. Analog-to-Digital Conversion (ADC) module

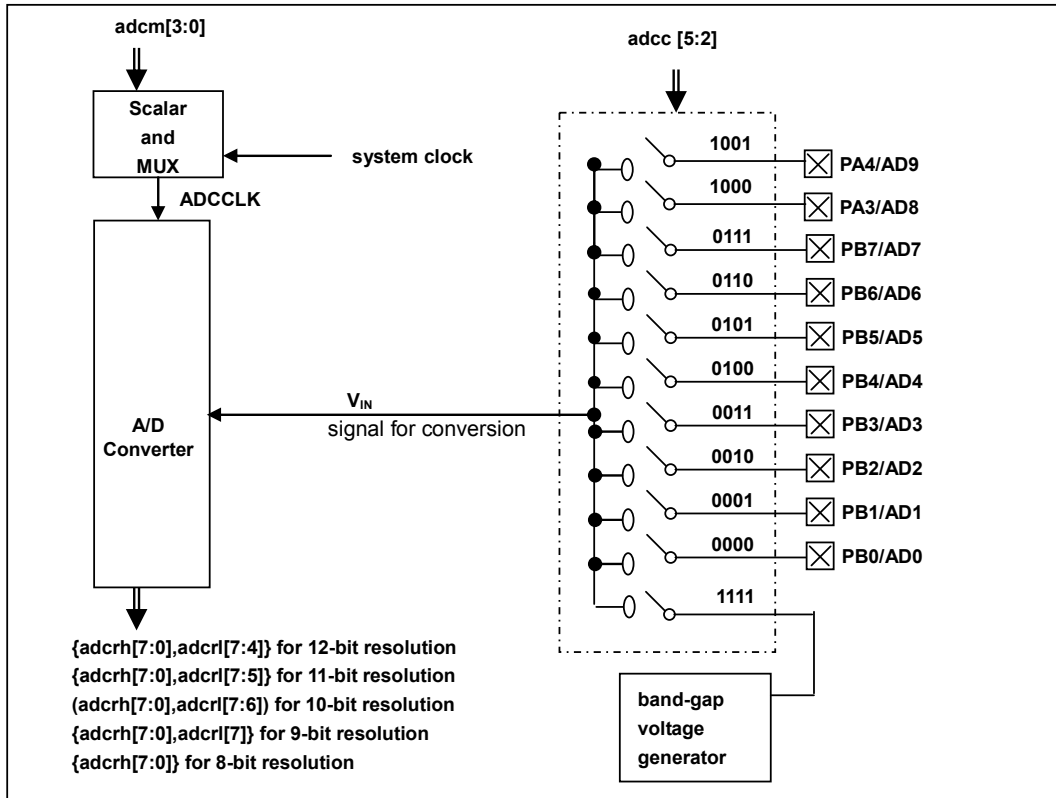


Fig. 26 : ADC Block Diagram

There are six registers when using the ADC module, which are:

- ◆ ADC Control Register (**adcc**)
- ◆ ADC Mode Register (**adcm**)
- ◆ ADC Result High/Low Register (**adcrh**, **adcl**)
- ◆ Port A/B Digital Input Enable Register (**padier**, **pbdi**)

The following steps are recommended to do the AD conversion procedure:

- (1) Configure the ADC module:
 - ◆ Select the ADC input channel by **adcc** register
 - ◆ Select the bit resolution of ADC by **adcm** register
 - ◆ Configure the AD conversion clock by **adcm** register
 - ◆ Configure the pin as analog input by **padier**, **pbdi** register
 - ◆ Enable the ADC module by **adcc** register

- (2) Configure interrupt for ADC: (if desired)
 - ◆ Clear the ADC interrupt request flag in bit 3 of intrq register
 - ◆ Enable the ADC interrupt request in bit 3 of inten register
 - ◆ Enable global interrupt by issuing **engint** command
- (3) Start AD conversion:
 - ◆ Set ADC process control bit in the **adcc** register to start the conversion (set1 **adcc.6**).
- (4) Wait for the completion flag of AD conversion, by either:
 - ◆ Waiting for the completion flag by using command “wait1 adcc.6”; or
 - ◆ Waiting for the ADC interrupt.
- (5) Read the ADC result registers:
 - ◆ Read **adcrh** and **adcrl** the result registers
- (6) For next conversion, goto step 1 or step 2 as required.

5-17-1. The input requirement for AD conversion

For the AD conversion to meet its specified accuracy, the charge holding capacitor (C_{HOLD}) must be allowed to fully charge to the voltage reference high level and discharge to the voltage reference low level. The analog input model is shown as Fig. 22, the signal driving source impedance (R_s) and the internal sampling switch impedance (R_{ss}) will affect the required time to charge the capacitor C_{HOLD} directly. The internal sampling switch impedance may vary with ADC supply voltage; the signal driving source impedance will affect accuracy of analog input signal. User must ensure the measured signal is stable before sampling; therefore, the maximum signal driving source impedance is highly dependent on the frequency of signal to be measured.

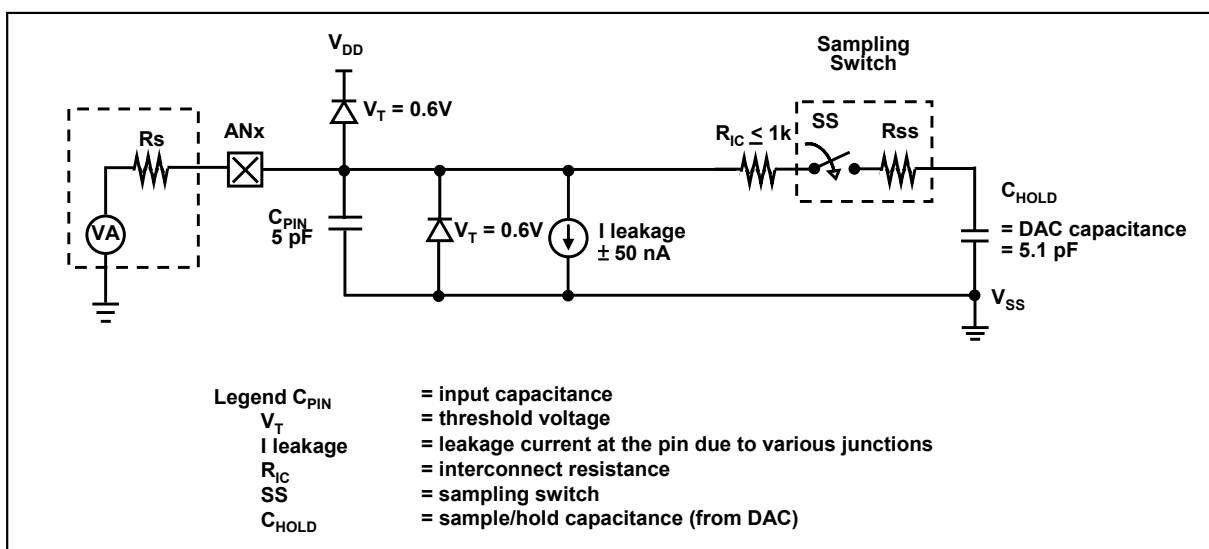


Fig. 22 : Analog Input Model

Before starting the AD conversion, the minimum signal acquisition time should be met for the selected analog input signal. The signal acquisition time (T_{ACQ}) of ADC in PMC234/PMS234 series is fixed to one clock period of ADCLK, the selection of ADCLK must be met the minimum signal acquisition time.

5-17-2. Select the ADC bit resolution

The ADC resolution is 12-bit, please configure **adcm** register bit [7:5] to be “100” before starting AD conversion via **\$ADCM** instruction. Higher resolution can detect small signal variation; however, it will take more time to convert the analog signal to digital signal. The selection can be done via **adcm** register. The ADC bit resolution should be configured before starting the AD conversion.

5-17-3. ADC clock selection

The clock of ADC module (ADCLK) can be selected by **adcm** register; there are 8 possible options for ADCLK from sysclk/1 to sysclk/128. Due to the signal acquisition time T_{ACQ} is one clock period of ADCLK, the ADCLK must meet that requirement. The recommended ADC clock is to operate at 2us.

5-17-4. AD conversion

The process of AD conversion starts from setting START/DONE bit (bit 6 of **adcc**) to high, the START/DONE flag for read will be cleared automatically, then converting analog signal bit by bit and finally setting START/DONE high to indicate the completion of AD conversion. If ADCLK is selected, T_{ADCLK} is the period of ADCLK and the AD conversion time can be calculated as follows:

- ◆ 12-bit resolution: AD conversion time = 17 T_{ADCLK}

5-17-5. Configure the analog pins

The 10 analog input signals for ADC shared with Port A[3], Port A[4], and Port B[7:0]. To avoid leakage current at the digital circuit, those pins defined for analog input should disable the digital input function (set the corresponding bit of **padier or pbdier** register to be 0).

The measurement signals of ADC belong to small signal; it should avoid the measured signal to be interfered during the measurement period, the selected pin should (1) be set to input mode (2) turn off weak pull-high resistor (3) set the corresponding pin to analog input by port A/B digital input disable register (**padier / pbdier**).

5-17-6. Using the ADC

The following example shows how to use ADC with PB0~PB3.

First, defining the selected pins:

```
PBC    = 0B_XXXX_0000;    // PB0 ~ PB3 as Input
PBPH  = 0B_XXXX_0000;    // PB0 ~ PB3 without pull-high
PBDIER = 0B_XXXX_0000;    // PB0 ~ PB3digital input is disabled
```

Next, setting **ADCC** register, example as below:

```
$ ADCC  Enable, PB3;      // set PB3 as ADC input
$ ADCC  Enable, PB0;      // set PB0 as ADC input
```

Next, setting **ADCM** register, example as below:

```
$ ADCM  12BIT, /32;      // recommend /32 @System Clock=16MHz
$ ADCM  12BIT, /16;     // recommend /16 @System Clock=8MHz
$ ADCM  12BIT, /8;      // recommend /8 @System Clock=4MHz
$ ADCM  8BIT, /8;
```

Then, start the ADC conversion:

```
AD_START = 1;           // start ADC conversion
WAIT1    AD_DONE ;      // wait ADC conversion result
```

Finally, it can read ADC result when **AD_DONE** is high:

```
WORD    Data;           // two bytes result: ADCRH and ADCRL
Data    = (ADCRH << 8) | ADCRL;
```

The ADC can be disabled by using the following method:

```
$ ADCC  Disable;
```

or

```
ADCC    = 0;
```

6. IO Registers

6-1. ACC Status Flag Register (*flag*), IO address = 0x00

Bit	Reset	R/W	Description
7–4	-	-	Reserved. These four bits are “1” when reading.
3	0	R/W	OV (Overflow Flag). This bit is set whenever the sign operation is overflow.
2	0	R/W	AC (Auxiliary Carry Flag). There are two conditions to set this bit, the first one is carry out of low nibble in addition operation and the other one is borrow from the high nibble into low nibble in subtraction operation.
1	0	R/W	C (Carry Flag). There are two conditions to set this bit, the first one is carry out in addition operation, and the other one is borrow in subtraction operation. Carry is also affected by shift with carry instruction.
0	0	R/W	Z (Zero Flag). This bit will be set when the result of arithmetic or logic operation is zero; Otherwise, it is cleared.

6-2. FPP unit Enable Register (*fppen*), IO address = 0x01

Bit	Reset	R/W	Description
7–2	-	-	Reserved. Please keep 0 for future compatibility.
1	0	R/W	FPP1 enable. This bit is used to enable FPP1. 0 / 1: disable / enable
0	1	R/W	FPP0 enable. This bit is used to enable FPP0. 0 / 1: disable / enable

6-3. Stack Pointer Register (*sp*), IO address = 0x02

Bit	Reset	R/W	Description
7–0	-	R/W	Stack Pointer Register. Read out the current stack pointer, or write to change the stack pointer.

6-4. Clock Mode Register (*clkmd*), IO address = 0x03

Bit	Reset	R/W	Description		
7-5	111	R/W	System clock selection:		
			Type 0, clkmd[3]=0	Type 1, clkmd[3]=1	
			<table style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 50%; vertical-align: top;"> 000: IHRC/4 001: IHRC/2 010: reserved 011: EOSC/4 100: EOSC/2 101: EOSC 110: ILRC/4 111: ILRC (default) </td> <td style="width: 50%; vertical-align: top;"> 000: IHRC/16 001: IHRC/8 010: reserved 011: IHRC/32 100: IHRC/64 101: EOSC/8 11x: reserved. </td> </tr> </table>	000: IHRC/4 001: IHRC/2 010: reserved 011: EOSC/4 100: EOSC/2 101: EOSC 110: ILRC/4 111: ILRC (default)	000: IHRC/16 001: IHRC/8 010: reserved 011: IHRC/32 100: IHRC/64 101: EOSC/8 11x: reserved.
000: IHRC/4 001: IHRC/2 010: reserved 011: EOSC/4 100: EOSC/2 101: EOSC 110: ILRC/4 111: ILRC (default)	000: IHRC/16 001: IHRC/8 010: reserved 011: IHRC/32 100: IHRC/64 101: EOSC/8 11x: reserved.				
4	1	R/W	Internal High RC Enable. 0 / 1: disable / enable		
3	0	RW	Clock Type Select. This bit is used to select the clock type in bit [7:5]. 0 / 1: Type 0 / Type 1.		
2	1	R/W	Internal Low RC Enable. 0 / 1: disable / enable If ILRC is disabled, watchdog timer is also disabled.		
1	1	R/W	Watch Dog Enable. 0 / 1: disable / enable		
0	0	R/W	Pin PA5/RESET# function. 0 / 1: PA5 / RESET#.		

6-5. Interrupt Enable Register (*inten*), IO address = 0x04

Bit	Reset	R/W	Description
7	-	-	Reserved.
6	0	R/W	Enable interrupt from Timer2. 0 / 1: disable / enable.
5	-	-	Reserved.
4	0	R/W	Enable interrupt from comparator. 0 / 1: disable / enable.
3	0	R/W	Enable interrupt from ADC. 0 / 1: disable / enable.
2	0	R/W	Enable interrupt from Timer16 overflow. 0 / 1: disable / enable.
1	0	R/W	Enable interrupt from PB0. 0 / 1: disable / enable.
0	0	R/W	Enable interrupt from PA0.0 / 1: disable / enable.

6-6. Interrupt Request Register (*intrq*), IO address = 0x05

Bit	Reset	R/W	Description
7	-	-	Reserved.
6	-	R/W	Interrupt Request from Timer2, this bit is set by hardware and cleared by software. 0 / 1: No request / Request
5	-	-	Reserved.
4	-	R/W	Interrupt Request from comparator output, this bit is set by hardware (rising edge) and cleared by software. 0 / 1: No request / Request
3	-	R/W	Interrupt Request from ADC, this bit is set by hardware and cleared by software. 0 / 1: No request / Request
2	-	R/W	Interrupt Request from Timer16, this bit is set by hardware and cleared by software. 0 / 1: No request / Request
1	-	R/W	Interrupt Request from pin PB0, this bit is set by hardware and cleared by software. 0 / 1: No request / Request
0	-	R/W	Interrupt Request from pin PA0, this bit is set by hardware and cleared by software. 0 / 1: No Request / request

6-7. Timer16 mode Register (*t16m*), IO address = 0x06

Bit	Reset	R/W	Description
7–5	000	R/W	Timer Clock source selection. 000: disable 001: system clock (CLK) 010: reserved 011: reserved 100: IHRC 101: EOSC 110: ILRC 111: PA0
4–3	00	R/W	Timer16 clock pre-divider. 00: /1 01: /4 10: /16 11: /64
2–0	000	R/W	Interrupt source selection. Interrupt event happens when selected bit goes high. 0 : bit 8 of Timer16 1 : bit 9 of Timer16 2 : bit 10 of Timer16 3 : bit 11 of Timer16 4 : bit 12 of Timer16 5 : bit 13 of Timer16 6 : bit 14 of Timer16 7 : bit 15 of Timer16

6-8. General Data register for IO (*gdio*), IO address = 0x07

Bit	Reset	R/W	Description
7–0	00	R/W	General data for IO. This port is the general data buffer in IO space and cleared when POR or LVR, and <u>it will KEEP the old values when reset from watch dog time out</u> . It can perform the IO operation, like <i>wait0 gdio.x</i> , <i>wait1 gdio.x</i> and <i>tog gdio.x</i> to replace of operations which instructions are supported in memory space (ex: <i>wait1 mem</i> ; <i>wait0 mem</i> ; <i>tog mem</i>).

6-9. External Oscillator setting Register (*eoscr*), IO address = 0x0a

Bit	Reset	R/W	Description
7	0	WO	Enable external crystal oscillator. 0 / 1 : Disable / Enable
6–5	00	WO	External crystal oscillator selection. 00 : Reserved 01 : Low driving capability, for lower frequency, ex: 32KHz crystal oscillator 10 : Middle driving capability, for middle frequency, ex: 1MHz crystal oscillator 11 : High driving capability, for higher frequency, ex: 4MHz crystal oscillator
4–1	-	-	Reserved. Please keep 0 for future compatibility.
0	0	WO	Power-down the Band-gap and LVR hardware modules. 0 / 1: normal / power-down.

6-10. Internal High RC oscillator control Register (*ihrcr*), IO address = 0x0b

Bit	Reset	R/W	Description
7–0	00	WO	Bit [7:0] for frequency calibration of IHRC. 0x00 for lowest frequency and 0xff for highest frequency.

6-11. Interrupt Edge Select Register (*integs*), IO address = 0x0c

Bit	Reset	R/W	Description
7–5	-	-	Reserved.
4	0	WO	Timer16 edge selection. 0 : rising edge of the selected bit to trigger interrupt 1 : falling edge of the selected bit to trigger interrupt
3–2	00	WO	PB0 edge selection. 00 : both rising edge and falling edge of the selected bit to trigger interrupt 01 : rising edge of the selected bit to trigger interrupt 10 : falling edge of the selected bit to trigger interrupt 11 : reserved.
1–0	00	WO	PA0 edge selection. 00 : both rising edge and falling edge of the selected bit to trigger interrupt 01 : rising edge of the selected bit to trigger interrupt 10 : falling edge of the selected bit to trigger interrupt 11 : reserved.

6-12. Port A Digital Input Enable Register (*padier*), IO address = 0x0d

Bit	Reset	R/W	Description
7	1	WO	Enable PA7 digital input and wake-up event. 1 / 0 : enable / disable. This bit should be set to low to prevent leakage current when external crystal oscillator is used. If this bit is set to low, PA7 can NOT be used to wake-up the system. Note: For ICE emulation, the function is disabled when this bit is "1" and "0" is enabled.
6	1	WO	Enable PA6 digital input and wake-up event. 1 / 0 : enable / disable. This bit should be set to low to prevent leakage current when external crystal oscillator is used. If this bit is set to low, PA6 can NOT be used to wake-up the system. Note: For ICE emulation, the function is disabled when this bit is "1" and "0" is enabled.
5	1	WO	Enable PA5 wake-up event. 1 / 0 : enable / disable. This bit can be set to low to disable wake-up from PA5 toggling. Note: For ICE emulation, wakeup is disabled when this bit is "1" and "0" is enabled.
4	1	WO	Enable PA4 digital input and wake-up event. 1 / 0 : enable / disable. This bit should be set to low when PA4 is assigned as AD input to prevent leakage current. If this bit is set to low, PA4 can NOT be used to wake-up the system. Note: For ICE emulation, the function is disabled when this bit is "1" and "0" is enabled.
3	1	WO	Enable PA3 digital input and wake-up event. 1 / 0 : enable / disable. This bit should be set to low when PA3 is assigned as AD input to prevent leakage current. If this bit is set to low, PA3 can NOT be used to wake-up the system. Note: For ICE emulation, the function is disabled when this bit is "1" and "0" is enabled.
2	1	WO	Enable PA2 wake-up event. 1 / 0 : enable / disable. This bit can be set to low to disable wake-up from PA2 toggling. Note: For ICE emulation, wakeup is disabled when this bit is "1" and "0" is enabled.
1	1	WO	Enable PA1 wake-up event. 1 / 0 : enable / disable. This bit can be set to low to disable wake-up from PA1 toggling. Note: For ICE emulation, wakeup is disabled when this bit is "1" and "0" is enabled.
0	1	WO	Enable PA0 wake-up event and interrupt request. 1 / 0 : enable / disable. This bit can be set to low to disable wake-up from PA0 toggling and interrupt request from this pin. Note: For ICE emulation, the function is disabled when this bit is "1" and "0" is enabled.

Note: Due to the controlling polarity of this register is different between ICE and real chip. In order to unify the program for both ICE emulation and real chip to be the same one, please use the following command to write this register:

```
"$ PADIER    0xhh" ;
```

For example:

```
$ PADIER    0xF0;
```

It is used to enable the digital input and wakeup function of bit [7:4] of port A for both ICE and real chip, IDE will handle the difference between ICE and real chip automatically.

6-13. Port B Digital Input Enable Register (*pbdier*), IO address = 0x0e

Bit	Reset	R/W	Description
7–0	0xff	WO	Enable PB7~PB0 digital input to prevent leakage when the pin is assigned for AD input. When disable is selected, the wakeup function from this pin is also disabled. 0 / 1 : disable / enable Note: For ICE emulation, the function is disabled when this bit is “1” and “0” is enabled.

Note: Due to the controlling polarity of this register is different between ICE and real chip. In order to unify the program for both ICE emulation and real chip to be the same one, please use the following command to write this register:

```
$ PBDIER    0xhh" ;
```

For example:

```
$ PBDIER    0xF0;
```

It is used to enable the digital input and wakeup function of bit [7:4] of port B for both ICE and real chip, IDE will handle the difference between ICE and real chip automatically.

6-14. Port A Data Register (*pa*), IO address = 0x10

Bit	Reset	R/W	Description
7–0	-	R/W	Data register for Port A.

6-15. Port A Control Register (*pac*), IO address = 0x11

Bit	Reset	R/W	Description
7–0	0x00	R/W	Port A control registers. This register is used to define input mode or output mode for each corresponding pin of port A. 0 / 1: input / output <u>Please note that PA5 can be INPUT or OUTPUT LOW ONLY, the output state will be tri-state when PA5 is programmed into output mode with data 1.</u>

6-16. Port A Pull-High Register (*paph*), IO address = 0x12

Bit	Reset	R/W	Description
7–0	0x00	R/W	Port A pull-high register. This register is used to enable the internal pull-high device on each corresponding pin of port A and this pull high function is active only for input mode. 0 / 1 : disable / enable Please note that PA5 does NOT have pull-up resistor.

6-17. Port B Data Register (*pb*), IO address = 0x14

Bit	Reset	R/W	Description
7–0	-	R/W	Data register for Port B.

6-18. Port B Control Register (*pbcr*), IO address = 0x15

Bit	Reset	R/W	Description
7–0	0x00	R/W	Port B control register. This register is used to define input mode or output mode for each corresponding pin of port B. 0 / 1: input / output

6-19. Port B Pull-High Register (*pbph*), IO address = 0x16

Bit	Reset	R/W	Description
7–0	0x00	R/W	Port B pull-high register. This register is used to enable the internal pull-high device on each corresponding pin of port B. 0 / 1 : disable / enable

6-20. Port C Data Register (*pc*), IO address = 0x17

Bit	Reset	R/W	Description
7–0	-	R/W	Bit [7:0] of data register for Port C.

6-21. Port C Control Register (*pcc*), IO address = 0x18

Bit	Reset	R/W	Description
7–0	0x00	R/W	Bit [7:0] of port C control register. This register is used to define input mode or output mode for each corresponding pin. 0 / 1: input / output

6-22. Port C Pull-High Register (*pcph*), IO address = 0x19

Bit	Reset	R/W	Description
7–0	0x00	R/W	Bit [7:0] of Port C pull-high register. This register is used to enable the internal pull-high device on each corresponding pin. 0 / 1 : disable / enable

6-23. Port D Data Register (*pd*), IO address = 0x1a

Bit	Reset	R/W	Description
7–2	-	-	Reserved
1–0	-	R/W	Bit [1:0] of data register for Port D.

6-24. Port D Control Register (*pdcc*), IO address = 0x1b

Bit	Reset	R/W	Description
7–2	-	-	Reserved
1–0	00	R/W	Bit [1:0] of port D control register. This register is used to define input mode or output mode for each corresponding pin. 0 / 1: input / output

6-25. Port D Pull-High Register (*pdph*), IO address = 0x1c

Bit	Reset	R/W	Description
7–2	-	-	Reserved
1–0	00	R/W	Bit [1:0] of Port D pull-high register. This register is used to enable the internal pull-high device on each corresponding pin. 0 / 1 : disable / enable

6-26. ADC Control Register (*adcc*), IO address = 0x20

Bit	Reset	R/W	Description
7	0	R/W	Enable ADC function. 0/1: Disable/Enable.
6	-	R/W	ADC process control bit. Write “1” to start AD conversion, and the flag is cleared automatically when starting the AD conversion ; Read “1” to indicate the completion of AD conversion and “0” is in progressing.
5–2	0000	R/W	Channel selector. These four bits are used to select input signal for AD conversion. 0000: PB0/AD0, 0001: PB1/AD1, 0010: PB2/AD2, 0011: PB3/AD3, 0100: PB4/AD4, 0101: PB5/AD5, 0110: PB6/AD6, 0111: PB7/AD7 1000: PA3/AD8 1001: PA4/AD9 1111: band-gap 1.20 volt reference voltage Others: reserved
1–0	-	-	Reserved. (keep 0 for future compatibility)

6-27. ADC Mode Register (*adcm*), IO address = 0x21

Bit	Reset	R/W	Description
7–5	000	WO	Bit Resolution. 100: 12-bit, AD 12-bit result [11:0] = { <i>adcrh</i> [7:0], <i>adcrl</i> [7:4] }. Others: reserved,
4	-	-	Reserved (keep 0 for future compatibility)
3–1	000	WO	ADC clock source selection. 000: sysclk/1, 001: sysclk/2, 010: sysclk/4, 011: sysclk/8, 100: sysclk/16, 101: sysclk/32, 110: sysclk/64, 111: sysclk/128,
0	-	-	Reserved

6-28. ADC Result High Register (*adcrh*), IO address = 0x22

Bit	Reset	R/W	Description
7–0	-	RO	These eight read-only bits will be the bit [11:4] of AD conversion result. The bit 7 of this register is the MSB of ADC result for any resolution.

6-29. ADC Result Low Register (*adcrl*), IO address = 0x23

Bit	Reset	R/W	Description
7–4	-	RO	These four bits will be the bit [3:0] of AD conversion result.
3–0	-	-	Reserved

6-30. Miscellaneous Register (misc), IO address = 0x3b

Bit	Reset	R/W	Description
7	0	-	Reserved
6	0	WO	Enable extremely low current for 32KHz crystal oscillator AFTER oscillation. 0: Normal. 1: Low driving current for 32KHz crystal oscillator.
5	0	WO	Enable fast Wake-up. Fast wake-up is NOT supported when EOSC is enabled. 0: Normal wake-up. The wake-up time is 1024 ILRC clocks 1: Fast wake-up. (for The wake-up time is 128 CLKs (system clock) + oscillator stable time. If wake-up from STOPEXE suspend, there is no oscillator stable time; If wake-up from STOPSYS suspend, it will be IHRC or ILRC stable time from power-on. Please notice that the clock source of watch-dog will be switched to system clock (for example: 4MHz) when fast wakeup is enabled. Therefore, for fast wake-up, recommending turning off the watchdog timer before enabling the fast wakeup. After wake-up, turning on the watchdog timer after disabling the fast wakeup.
4	0	WO	Enable to generate half VDD on PA0/PA1/PA2/PA3 pins. 0 / 1 : Disable / Enable
3	0	WO	Recover time from LVR. 0: Normal. The system will take about 1024 ILRC clocks to boot up from LVR. 1: Fast. The system will take about 64 ILRC clocks to boot up from LVR.
2	0	WO	Disable LVR function. 0 / 1 : Enable / Disable
1-0	00	WO	Watch dog time out period 00: 2048 ILRC clock period 01: 4096 ILRC clock period 10: 16384 ILRC clock period 11: 256 ILRC clock period

6-31. Timer2 Control Register (tm2c), IO address = 0x3c

Bit	Reset	R/W	Description
7-4	0000	R/W	Timer2 clock selection. 0000 : disable 0001 : system clock 0010 : internal high RC oscillator (IHRC) 0011 : reserved 0100 : ILRC 0101 : comparator output 011x : reserved 1000 : PA0 (rising edge) 1001 : ~PA0 (falling edge) 1010 : PA3 (rising edge) 1011 : ~PA3 (falling edge) 1100 : PA4 (rising edge) 1101 : ~PA4 (falling edge) <u>Notice: In ICE mode and IHRC is selected for Timer2 clock, the clock sent to Timer2 does NOT be stopped. Timer2 will keep counting when ICE is in halt state.</u>
3-2	00	R/W	Timer2 output selection. 00 : disable 01 : PA2 10 : PA3 11 : PA1
1	0	R/W	Timer2 mode selection. 0 / 1 : period mode / PWM mode
0	0	R/W	Enable to inverse the polarity of Timer2 output. 0 / 1: disable / enable.

6-32. Timer2 Counter Register (tm2ct), IO address = 0x3d

Bit	Reset	R/W	Description
7-0	0x00	R/W	Bit [7:0] of Timer2 counter register.

6-33. Timer2 Scalar Register (tm2s), IO address = 0x37

Bit	Reset	R/W	Description
7	0	WO	PWM resolution selection. 0 : 8-bit 1 : 6-bit
6-5	00	WO	Timer2 clock pre-scalar. 00 : ÷ 1 01 : ÷ 4 10 : ÷ 16 11 : ÷ 64
4-0	00000	WO	Timer2 clock scalar.

6-34. Timer2 Bound Register (tm2b), IO address = 0x09

Bit	Reset	R/W	Description
7-0	0x00	WO	Timer2 bound register.

6-35. Comparator Control Register (gpcc), IO address = 0x3e

Bit	Reset	R/W	Description
7	0	R/W	Enable comparator. 0 / 1 : disable / enable When this bit is set to enable, please also set the corresponding analog input pins to be digital disable to prevent IO leakage.
6	-	RO	Comparator result of comparator. 0: plus input < minus input 1: plus input > minus input
5	0	R/W	Select whether the comparator result output will be sampled by TM2_CLK? 0: result output NOT sampled by TM2_CLK 1: result output sampled by TM2_CLK
4	0	R/W	Inverse the polarity of result output of comparator. 0: polarity is NOT inversed. 1: polarity is inversed.
3-1	000	R/W	Selection the minus input (-) of comparator. 000 : PC0/CIN3- 001 : PC5/CIN+ 010 : Internal 1.20 volt band-gap reference voltage 011 : $V_{internal R}$ 100 : PA3/CIN1- 101 : PA4/CIN2-
0	0	R/W	Selection the plus input (+) of comparator. 0 : $V_{internal R}$ 1 : PC5/CIN+

6-36. Comparator Selection Register (gpccs), IO address = 0x22

Bit	Reset	R/W	Description
7	0	WO	Comparator output enable (to PB0). 0 / 1 : disable / enable
6	-	-	Reserved.
5	0	WO	Selection of high range of comparator.
4	0	WO	Selection of low range of comparator.
3-0	0000	WO	Selection the voltage level of comparator. 0000 (lowest) ~ 1111 (highest)

7. Instructions

Symbol	Description
ACC	Accumulator (ACC is the shorthand of accumulator)
a	Accumulator (a is symbol of accumulator in program)
sp	Stack pointer
flag	ACC status flag register
l	Immediate data
&	Logical AND
	Logical OR
←	Movement
^	Exclusive logic OR
+	Add
–	Subtraction
~	NOT (logical complement, 1's complement)
$\overline{\text{T}}$	NEG (2's complement)
OV	Overflow (The operational result is out of range in signed 2's complement number system)
Z	Zero (If the result of ALU operation is zero, this bit is set to 1)
C	Carry (The operational result is to have carry out for addition or to borrow carry for subtraction in unsigned number system)
AC	Auxiliary Carry (If there is a carry out from low nibble after the result of ALU operation, this bit is set to 1)
pc0	Program counter for FPP0
pc1	Program counter for FPP1

7-1. Data Transfer Instructions

<i>mov</i> a, l	Move immediate data into ACC. Example: <i>mov a, 0x0f;</i> Result: <i>a ← 0fh;</i> Affected flags: 『N』 Z 『N』 C 『N』 AC 『N』 OV
<i>mov</i> M, a	Move data from ACC into memory Example: <i>mov MEM, a;</i> Result: <i>MEM ← a</i> Affected flags: 『N』 Z 『N』 C 『N』 AC 『N』 OV
<i>mov</i> a, M	Move data from memory into ACC Example: <i>mov a, MEM;</i> Result: <i>a ← MEM;</i> Flag Z is set when MEM is zero. Affected flags: 『Y』 Z 『N』 C 『N』 AC 『N』 OV

<i>mov</i> a, IO	<p>Move data from IO into ACC</p> <p>Example: <i>mov a, pa;</i></p> <p>Result: $a \leftarrow pa$; Flag Z is set when pa is zero.</p> <p>Affected flags: 『Y』Z 『N』C 『N』AC 『N』OV</p>
<i>mov</i> IO, a	<p>Move data from ACC into IO</p> <p>Example: <i>mov pb, a;</i></p> <p>Result: $pb \leftarrow a$</p> <p>Affected flags: 『N』Z 『N』C 『N』AC 『N』OV</p>
<i>nmov</i> M, a	<p>Take the negative logic (2's complement) of ACC to put on memory</p> <p>Example: <i>mov MEM, a;</i></p> <p>Result: $MEM \leftarrow \overline{a}$</p> <p>Affected flags: 『N』Z 『N』C 『N』AC 『N』OV</p> <p>Application Example:</p> <hr style="border-top: 1px dashed black;"/> <pre style="margin-left: 40px;"> mov a, 0xf5 ; // ACC is 0xf5 nmov ram9, a; // ram9 is 0x0b, ACC is 0xf5 </pre> <hr style="border-top: 1px dashed black;"/>
<i>nmov</i> a, M	<p>Take the negative logic (2's complement) of memory to put on ACC</p> <p>Example: <i>mov a, MEM;</i></p> <p>Result: $a \leftarrow \overline{MEM}$; Flag Z is set when \overline{MEM} is zero.</p> <p>Affected flags: 『Y』Z 『N』C 『N』AC 『N』OV</p> <p>Application Example:</p> <hr style="border-top: 1px dashed black;"/> <pre style="margin-left: 40px;"> mov a, 0xf5 ; mov ram9, a ; // ram9 is 0xf5 nmov a, ram9 ; // ram9 is 0xf5, ACC is 0x0b </pre> <hr style="border-top: 1px dashed black;"/>
<i>ldtabh</i> index	<p>Load high byte data in OTP program memory to ACC by using index as OTP address. It needs 2T to execute this instruction.</p> <p>Example: <i>ldtabh index;</i></p> <p>Result: $a \leftarrow \{\text{bit 15~8 of OTP [index]}\}$;</p> <p>Affected flags: 『N』Z 『N』C 『N』AC 『N』OV</p> <p>Application Example:</p> <hr style="border-top: 1px dashed black;"/> <pre style="margin-left: 40px;"> word ROMptr ; // declare a pointer of ROM in RAM ... mov a, la@TableA ; // assign pointer to ROM TableA (LSB) mov lb@ROMptr, a ; // save pointer to RAM (LSB) mov a, ha@TableA ; // assign pointer to ROM TableA (MSB) mov hb@ROMptr, a ; // save pointer to RAM (MSB) ... ldtabh ROMptr ; // load TableA MSB to ACC (ACC=0X02) ... TableA : dc 0x0234, 0x0042, 0x0024, 0x0018 ; </pre> <hr style="border-top: 1px dashed black;"/>

<i>ldtabl</i> index	<p>Load low byte data in OTP to ACC by using index as OTP address. It needs 2T to execute this instruction.</p> <p>Example: <i>ldtabl index;</i></p> <p>Result: $a \leftarrow \{\text{bit7}\sim\text{0 of OTP} [\text{index}]\};$</p> <p>Affected flags: 『N』 Z 『N』 C 『N』 AC 『N』 OV</p> <p>Application Example:</p> <hr style="border-top: 1px dashed black;"/> <pre style="margin: 0;"> word ROMptr ; // declare a pointer of ROM in RAM ... mov a, la@TableA ; // assign pointer to ROM TableA (LSB) mov lb@ROMptr, a ; // save pointer to RAM (LSB) mov a, ha@TableA ; // assign pointer to ROM TableA (MSB) mov hb@ROMptr, a ; // save pointer to RAM (MSB) ... ldtabl ROMptr ; // load TableA LSB to ACC (ACC=0x34) ... TableA : dc 0x0234, 0x0042, 0x0024, 0x0018 ; </pre> <hr style="border-top: 1px dashed black;"/>
<i>ldt16</i> word	<p>Move 16-bit counting values in Timer16 to memory in word.</p> <p>Example: <i>ldt16 word;</i></p> <p>Result: $\text{word} \leftarrow \text{16-bit timer}$</p> <p>Affected flags: 『N』 Z 『N』 C 『N』 AC 『N』 OV</p> <p>Application Example:</p> <hr style="border-top: 1px dashed black;"/> <pre style="margin: 0;"> word T16val ; // declare a RAM word ... clear lb@ T16val ; // clear T16val (LSB) clear hb@ T16val ; // clear T16val (MSB) stt16 T16val ; // initial T16 with 0 ... set1 t16m.5 ; // enable Timer16 ... set0 t16m.5 ; // disable Timer16 ldt16 T16val ; // save the T16 counting value to T16val ... </pre> <hr style="border-top: 1px dashed black;"/>

<i>stt16</i> word	<p>Store 16-bit data from memory in word to Timer16.</p> <p>Example: <i>stt16 word;</i></p> <p>Result: 16-bit timer ←word</p> <p>Affected flags: 『N』 Z 『N』 C 『N』 AC 『N』 OV</p> <p>Application Example:</p> <hr style="border-top: 1px dashed black;"/> <pre style="margin-left: 40px;"> word T16val ; // declare a RAM word ... mov a, 0x34 ; mov lb@T16val, a ; // move 0x34 to T16val (LSB) mov a, 0x12 ; mov hb@T16val, a ; // move 0x12 to T16val (MSB) stt16 T16val ; // initial T16 with 0x1234 ... </pre> <hr style="border-top: 1px dashed black;"/>
<i>xch</i> M	<p>Exchange data between ACC and memory</p> <p>Example: <i>xch MEM;</i></p> <p>Result: MEM ← a, a ← MEM</p> <p>Affected flags: 『N』 Z 『N』 C 『N』 AC 『N』 OV</p>
<i>idxm</i> a, index	<p>Move data from specified memory to ACC by indirect method. It needs 2T to execute this instruction.</p> <p>Example: <i>idxm a, index;</i></p> <p>Result: a ← [index], where index is declared by word.</p> <p>Affected flags: 『N』 Z 『N』 C 『N』 AC 『N』 OV</p> <p>Application Example:</p> <hr style="border-top: 1px dashed black;"/> <pre style="margin-left: 40px;"> word RAMIndex ; // declare a RAM pointer ... mov a, 0x5B ; // assign pointer to an address (LSB) mov lb@RAMIndex, a ; // save pointer to RAM (LSB) mov a, 0x00 ; // assign 0x00 to an addr.(MSB), be 0 mov hb@RAMIndex, a ; // save pointer to RAM (MSB) ... idxm a, RAMIndex ; // move data in address 0x5B to ACC </pre> <hr style="border-top: 1px dashed black;"/>
<i>ldxm</i> index, a	<p>Move data from ACC to specified memory by indirect method. It needs 2T to execute this instruction.</p> <p>Example: <i>ldxm index, a;</i></p> <p>Result: [index] ← a; where index is declared by word.</p> <p>Affected flags: 『N』 Z 『N』 C 『N』 AC 『N』 OV</p>

Application Example:

```

word    RAMIndex ;           // declare a RAM pointer
...
mov     a, 0x5B ;           // assign pointer to an address (LSB)
mov     lb@RAMIndex, a ;    // save pointer to RAM (LSB)
mov     a, 0x00 ;           // assign 0x00 to an addr.(MSB), be 0
mov     hb@RAMIndex, a ;    // save pointer to RAM (MSB)
...
mov     a, 0xA5 ;
idxm    RAMIndex, a ;     // mov 0xA5 to memory in address 0x5B

```

pushaf

Move the ACC and flag register to memory that address specified in the stack pointer.

Example: **pushaf;**

Result: [sp] ← {flag, ACC};
sp ← sp + 2 ;

Affected flags: 『N』 Z 『N』 C 『N』 AC 『N』 OV

Application Example:

```

.romadr 0x10 ;           // ISR entry address
pushaf ;                // put ACC and flag into stack memory
...
...                    // ISR program
...                    // ISR program
popaf ;                 // restore ACC and flag from stack memory
reti ;

```

popaf

Restore ACC and flag from the memory which address is specified in the stack pointer.

Example: **popaf;**

Result: sp ← sp - 2 ;
{Flag, ACC} ← [sp] ;

Affected flags: 『Y』 Z 『Y』 C 『Y』 AC 『Y』 OV

7-2. Arithmetic Operation Instructions

<i>add</i> a, I	<p>Add immediate data with ACC, then put result into ACC</p> <p>Example: add a, 0x0f ;</p> <p>Result: a ← a + 0fh</p> <p>Affected flags: 『Y』 Z 『Y』 C 『Y』 AC 『Y』 OV</p>
<i>add</i> a, M	<p>Add data in memory with ACC, then put result into ACC</p> <p>Example: add a, MEM ;</p> <p>Result: a ← a + MEM</p> <p>Affected flags: 『Y』 Z 『Y』 C 『Y』 AC 『Y』 OV</p>

<i>add</i> M, a	<p>Add data in memory with ACC, then put result into memory</p> <p>Example: <i>add MEM, a;</i></p> <p>Result: $MEM \leftarrow a + MEM$</p> <p>Affected flags: 『Y』 Z 『Y』 C 『Y』 AC 『Y』 OV</p>
<i>addc</i> a, M	<p>Add data in memory with ACC and carry bit, then put result into ACC</p> <p>Example: <i>addc a, MEM;</i></p> <p>Result: $a \leftarrow a + MEM + C$</p> <p>Affected flags: 『Y』 Z 『Y』 C 『Y』 AC 『Y』 OV</p>
<i>addc</i> M, a	<p>Add data in memory with ACC and carry bit, then put result into memory</p> <p>Example: <i>addc MEM, a;</i></p> <p>Result: $MEM \leftarrow a + MEM + C$</p> <p>Affected flags: 『Y』 Z 『Y』 C 『Y』 AC 『Y』 OV</p>
<i>addc</i> a	<p>Add carry with ACC, then put result into ACC</p> <p>Example: <i>addc a;</i></p> <p>Result: $a \leftarrow a + C$</p> <p>Affected flags: 『Y』 Z 『Y』 C 『Y』 AC 『Y』 OV</p>
<i>addc</i> M	<p>Add carry with memory, then put result into memory</p> <p>Example: <i>addc MEM;</i></p> <p>Result: $MEM \leftarrow MEM + C$</p> <p>Affected flags: 『Y』 Z 『Y』 C 『Y』 AC 『Y』 OV</p>
<i>nadd</i> a, M	<p>Add negative logic (2's complement) of ACC with memory</p> <p>Example: <i>nadd a, MEM;</i></p> <p>Result: $a \leftarrow \overline{a} + MEM$</p> <p>Affected flags: 『Y』 Z 『Y』 C 『Y』 AC 『Y』 OV</p>
<i>nadd</i> M, a	<p>Add negative logic (2's complement) of memory with ACC</p> <p>Example: <i>nadd MEM, a;</i></p> <p>Result: $MEM \leftarrow \overline{MEM} + a$</p> <p>Affected flags: 『Y』 Z 『Y』 C 『Y』 AC 『Y』 OV</p>
<i>sub</i> a, I	<p>Subtraction immediate data from ACC, then put result into ACC.</p> <p>Example: <i>sub a, 0x0f;</i></p> <p>Result: $a \leftarrow a - 0fh$ ($a + [2's \text{ complement of } 0fh]$)</p> <p>Affected flags: 『Y』 Z 『Y』 C 『Y』 AC 『Y』 OV</p>
<i>sub</i> a, M	<p>Subtraction data in memory from ACC, then put result into ACC</p> <p>Example: <i>sub a, MEM;</i></p> <p>Result: $a \leftarrow a - MEM$ ($a + [2's \text{ complement of } M]$)</p> <p>Affected flags: 『Y』 Z 『Y』 C 『Y』 AC 『Y』 OV</p>

<i>sub</i> M, a	<p>Subtraction data in ACC from memory, then put result into memory</p> <p>Example: <i>sub</i> MEM, a;</p> <p>Result: $MEM \leftarrow MEM - a$ ($MEM + [2\text{'s complement of } a]$)</p> <p>Affected flags: 『Y』Z 『Y』C 『Y』AC 『Y』OV</p>
<i>subc</i> a, M	<p>Subtraction data in memory and carry from ACC, then put result into ACC</p> <p>Example: <i>subc</i> a, MEM;</p> <p>Result: $a \leftarrow a - MEM - C$</p> <p>Affected flags: 『Y』Z 『Y』C 『Y』AC 『Y』OV</p>
<i>subc</i> M, a	<p>Subtraction ACC and carry bit from memory, then put result into memory</p> <p>Example: <i>subc</i> MEM, a ;</p> <p>Result: $MEM \leftarrow MEM - a - C$</p> <p>Affected flags: 『Y』Z 『Y』C 『Y』AC 『Y』OV</p>
<i>subc</i> a	<p>Subtraction carry from ACC, then put result into ACC</p> <p>Example: <i>subc</i> a;</p> <p>Result: $a \leftarrow a - C$</p> <p>Affected flags: 『Y』Z 『Y』C 『Y』AC 『Y』OV</p>
<i>subc</i> M	<p>Subtraction carry from the content of memory, then put result into memory</p> <p>Example: <i>subc</i> MEM;</p> <p>Result: $MEM \leftarrow MEM - C$</p> <p>Affected flags: 『Y』Z 『Y』C 『Y』AC 『Y』OV</p>
<i>inc</i> M	<p>Increment the content of memory</p> <p>Example: <i>inc</i> MEM ;</p> <p>Result: $MEM \leftarrow MEM + 1$</p> <p>Affected flags: 『Y』Z 『Y』C 『Y』AC 『Y』OV</p>
<i>dec</i> M	<p>Decrement the content of memory</p> <p>Example: <i>dec</i> MEM;</p> <p>Result: $MEM \leftarrow MEM - 1$</p> <p>Affected flags: 『Y』Z 『Y』C 『Y』AC 『Y』OV</p>
<i>clear</i> M	<p>Clear the content of memory</p> <p>Example: <i>clear</i> MEM ;</p> <p>Result: $MEM \leftarrow 0$</p> <p>Affected flags: 『N』Z 『N』C 『N』AC 『N』OV</p>

7-3. Shift Operation Instructions

<i>sr a</i>	Shift right of ACC Example: <i>sr a;</i> Result: $a(0, b7, b6, b5, b4, b3, b2, b1) \leftarrow a(b7, b6, b5, b4, b3, b2, b1, b0), C \leftarrow a(b0)$ Affected flags: 『N』 Z 『Y』 C 『N』 AC 『N』 OV
<i>src a</i>	Shift right of ACC with carry Example: <i>src a;</i> Result: $a(c, b7, b6, b5, b4, b3, b2, b1) \leftarrow a(b7, b6, b5, b4, b3, b2, b1, b0), C \leftarrow a(b0)$ Affected flags: 『N』 Z 『Y』 C 『N』 AC 『N』 OV
<i>sr M</i>	Shift right the content of memory Example: <i>sr MEM;</i> Result: $MEM(0, b7, b6, b5, b4, b3, b2, b1) \leftarrow MEM(b7, b6, b5, b4, b3, b2, b1, b0), C \leftarrow MEM(b0)$ Affected flags: 『N』 Z 『Y』 C 『N』 AC 『N』 OV
<i>src M</i>	Shift right of memory with carry Example: <i>src MEM;</i> Result: $MEM(c, b7, b6, b5, b4, b3, b2, b1) \leftarrow MEM(b7, b6, b5, b4, b3, b2, b1, b0), C \leftarrow MEM(b0)$ Affected flags: 『N』 Z 『Y』 C 『N』 AC 『N』 OV
<i>sl a</i>	Shift left of ACC Example: <i>sl a;</i> Result: $a(b6, b5, b4, b3, b2, b1, b0, 0) \leftarrow a(b7, b6, b5, b4, b3, b2, b1, b0), C \leftarrow a(b7)$ Affected flags: 『N』 Z 『Y』 C 『N』 AC 『N』 OV
<i>slc a</i>	Shift left of ACC with carry Example: <i>slc a;</i> Result: $a(b6, b5, b4, b3, b2, b1, b0, c) \leftarrow a(b7, b6, b5, b4, b3, b2, b1, b0), C \leftarrow a(b7)$ Affected flags: 『N』 Z 『Y』 C 『N』 AC 『N』 OV
<i>sl M</i>	Shift left of memory Example: <i>sl MEM;</i> Result: $MEM(b6, b5, b4, b3, b2, b1, b0, 0) \leftarrow MEM(b7, b6, b5, b4, b3, b2, b1, b0), C \leftarrow MEM(b7)$ Affected flags: 『N』 Z 『Y』 C 『N』 AC 『N』 OV
<i>slc M</i>	Shift left of memory with carry Example: <i>slc MEM;</i> Result: $MEM(b6, b5, b4, b3, b2, b1, b0, C) \leftarrow MEM(b7, b6, b5, b4, b3, b2, b1, b0), C \leftarrow MEM(b7)$ Affected flags: 『N』 Z 『Y』 C 『N』 AC 『N』 OV
<i>swap a</i>	Swap the high nibble and low nibble of ACC Example: <i>swap a;</i> Result: $a(b3, b2, b1, b0, b7, b6, b5, b4) \leftarrow a(b7, b6, b5, b4, b3, b2, b1, b0)$ Affected flags: 『N』 Z 『N』 C 『N』 AC 『N』 OV
<i>swap M</i>	Swap the high nibble and low nibble of memory Example: <i>swap MEM;</i> Result: $MEM(b3, b2, b1, b0, b7, b6, b5, b4) \leftarrow MEM(b7, b6, b5, b4, b3, b2, b1, b0)$ Affected flags: 『N』 Z 『N』 C 『N』 AC 『N』 OV

7-4. Logic Operation Instructions

<i>and</i> a, I	Perform logic AND on ACC and immediate data, then put result into ACC Example: <i>and a, 0x0f;</i> Result: $a \leftarrow a \& 0fh$ Affected flags: 『Y』 Z 『N』 C 『N』 AC 『N』 OV
<i>and</i> a, M	Perform logic AND on ACC and memory, then put result into ACC Example: <i>and a, RAM10;</i> Result: $a \leftarrow a \& RAM10$ Affected flags: 『Y』 Z 『N』 C 『N』 AC 『N』 OV
<i>and</i> M, a	Perform logic AND on ACC and memory, then put result into memory Example: <i>and MEM, a;</i> Result: $MEM \leftarrow a \& MEM$ Affected flags: 『Y』 Z 『N』 C 『N』 AC 『N』 OV
<i>or</i> a, I	Perform logic OR on ACC and immediate data, then put result into ACC Example: <i>or a, 0x0f;</i> Result: $a \leftarrow a 0fh$ Affected flags: 『Y』 Z 『N』 C 『N』 AC 『N』 OV
<i>or</i> a, M	Perform logic OR on ACC and memory, then put result into ACC Example: <i>or a, MEM;</i> Result: $a \leftarrow a MEM$ Affected flags: 『Y』 Z 『N』 C 『N』 AC 『N』 OV
<i>or</i> M, a	Perform logic OR on ACC and memory, then put result into memory Example: <i>or MEM, a;</i> Result: $MEM \leftarrow a MEM$ Affected flags: 『Y』 Z 『N』 C 『N』 AC 『N』 OV
<i>xor</i> a, I	Perform logic XOR on ACC and immediate data, then put result into ACC Example: <i>xor a, 0x0f;</i> Result: $a \leftarrow a \wedge 0fh$ Affected flags: 『Y』 Z 『N』 C 『N』 AC 『N』 OV
<i>xor</i> a, IO	Perform logic XOR on ACC and IO register, then put result into ACC Example: <i>xor a, pa;</i> Result: $a \leftarrow a \wedge pa$; // pa is the data register of port A Affected flags: 『Y』 Z 『N』 C 『N』 AC 『N』 OV
<i>xor</i> IO, a	Perform logic XOR on ACC and IO register, then put result into IO register Example: <i>xor pa, a;</i> Result: $pa \leftarrow a \wedge pa$; // pa is the data register of port A Affected flags: 『N』 Z 『N』 C 『N』 AC 『N』 OV

<i>xor</i> a, M	<p>Perform logic XOR on ACC and memory, then put result into ACC</p> <p>Example: <i>xor a, MEM;</i></p> <p>Result: $a \leftarrow a \wedge \text{RAM10}$</p> <p>Affected flags: 『Y』 Z 『N』 C 『N』 AC 『N』 OV</p>
<i>xor</i> M, a	<p>Perform logic XOR on ACC and memory, then put result into memory</p> <p>Example: <i>xor MEM, a;</i></p> <p>Result: $\text{MEM} \leftarrow a \wedge \text{MEM}$</p> <p>Affected flags: 『Y』 Z 『N』 C 『N』 AC 『N』 OV</p>
<i>not</i> a	<p>Perform 1's complement (logical complement) of ACC</p> <p>Example: <i>not a;</i></p> <p>Result: $a \leftarrow \sim a$</p> <p>Affected flags: 『Y』 Z 『N』 C 『N』 AC 『N』 OV</p> <p>Application Example:</p> <hr style="border-top: 1px dashed black;"/> <pre style="margin-left: 40px;"> mov a, 0x38; // ACC=0X38 not a; // ACC=0XC7 </pre> <hr style="border-top: 1px dashed black;"/>
<i>not</i> M	<p>Perform 1's complement (logical complement) of memory</p> <p>Example: <i>not MEM;</i></p> <p>Result: $\text{MEM} \leftarrow \sim \text{MEM}$</p> <p>Affected flags: 『Y』 Z 『N』 C 『N』 AC 『N』 OV</p> <p>Application Example:</p> <hr style="border-top: 1px dashed black;"/> <pre style="margin-left: 40px;"> mov a, 0x38; mov mem, a; // mem = 0x38 not mem; // mem = 0xC7 </pre> <hr style="border-top: 1px dashed black;"/>
<i>neg</i> a	<p>Perform 2's complement of ACC</p> <p>Example: <i>neg a;</i></p> <p>Result: $a \leftarrow \overline{\text{a}}$</p> <p>Affected flags: 『Y』 Z 『N』 C 『N』 AC 『N』 OV</p> <p>Application Example:</p> <hr style="border-top: 1px dashed black;"/> <pre style="margin-left: 40px;"> mov a, 0x38; // ACC=0X38 neg a; // ACC=0XC8 </pre> <hr style="border-top: 1px dashed black;"/>
<i>neg</i> M	<p>Perform 2's complement of memory</p> <p>Example: <i>neg MEM;</i></p> <p>Result: $\text{MEM} \leftarrow \overline{\text{MEM}}$</p> <p>Affected flags: 『Y』 Z 『N』 C 『N』 AC 『N』 OV</p>

	<p>Application Example:</p> <hr style="border-top: 1px dashed black;"/> <pre style="margin: 0;"> mov a, 0x38 ; mov mem, a ; // mem = 0x38 neg mem ; // mem = 0xC8 </pre> <hr style="border-top: 1px dashed black;"/>
<i>comp</i> a, I	<p>Compare ACC with immediate data Example: comp a, 0x55; Result: Flag will be changed by regarding as (a - 0x55) Affected flags: 『Y』 Z 『Y』 C 『Y』 AC 『Y』 OV Application Example:</p> <hr style="border-top: 1px dashed black;"/> <pre style="margin: 0;"> mov a, 0x38 ; comp a, 0x38 ; // Z flag is set comp a, 0x42 ; // C flag is set comp a, 0x24 ; // C, Z flags are clear comp a, 0x6a ; // C, AC flags are set </pre> <hr style="border-top: 1px dashed black;"/>
<i>comp</i> a, M	<p>Compare ACC with the content of memory Example: comp a, MEM; Result: Flag will be changed by regarding as (a - MEM) Affected flags: 『Y』 Z 『Y』 C 『Y』 AC 『Y』 OV Application Example:</p> <hr style="border-top: 1px dashed black;"/> <pre style="margin: 0;"> mov a, 0x38 ; mov mem, a ; comp a, mem ; // Z flag is set mov a, 0x42 ; mov mem, a ; mov a, 0x38 ; comp a, mem ; // C flag is set </pre> <hr style="border-top: 1px dashed black;"/>
<i>comp</i> M, a	<p>Compare ACC with the content of memory Example: comp MEM, a; Result: Flag will be changed by regarding as (MEM - a) Affected flags: 『Y』 Z 『Y』 C 『Y』 AC 『Y』 OV</p>

7-5. Bit Operation Instructions

<i>set0</i> IO.n	<p>Set bit n of IO port to low</p> <p>Example: <i>set0 pa.5;</i></p> <p>Result: set bit 5 of port A to low</p> <p>Affected flags: 『N』 Z 『N』 C 『N』 AC 『N』 OV</p>
<i>set1</i> IO.n	<p>Set bit n of IO port to high</p> <p>Example: <i>set1 pb.5;</i></p> <p>Result: set bit 5 of port B to high</p> <p>Affected flags: 『N』 Z 『N』 C 『N』 AC 『N』 OV</p>
<i>tog</i> IO.n	<p>Toggle bit state of bit n of IO port</p> <p>Example: <i>tog pa.5;</i></p> <p>Result: toggle bit 5 of port A</p> <p>Affected flags: 『N』 Z 『N』 C 『N』 AC 『N』 OV</p>
<i>set0</i> M.n	<p>Set bit n of memory to low</p> <p>Example: <i>set0 MEM.5;</i></p> <p>Result: set bit 5 of MEM to low</p> <p>Affected flags: 『N』 Z 『N』 C 『N』 AC 『N』 OV</p>
<i>set1</i> M.n	<p>Set bit n of memory to high</p> <p>Example: <i>set1 MEM.5;</i></p> <p>Result: set bit 5 of MEM to high</p> <p>Affected flags: 『N』 Z 『N』 C 『N』 AC 『N』 OV</p>
<i>swapc</i> IO.n	<p>Swap the nth bit of IO port with carry bit</p> <p>Example: <i>swapc IO.0;</i></p> <p>Result: $C \leftarrow IO.0, IO.0 \leftarrow C$</p> <p>When IO.0 is a port to output pin, carry C will be sent to IO.0;</p> <p>When IO.0 is a port from input pin, IO.0 will be sent to carry C;</p> <p>Affected flags: 『N』 Z 『Y』 C 『N』 AC 『N』 OV</p> <p>Application Example1 (serial output) :</p> <p>-----</p> <pre> ... set1 pac.0 ; // set PA.0 as output ... set0 flag.1 ; // C=0 swapc pa.0 ; // move C to PA.0 (bit operation), PA.0=0 set1 flag.1 ; // C=1 swapc pa.0 ; // move C to PA.0 (bit operation), PA.0=1 ... </pre>

Application Example2 (serial input) :

```

...
set0    pac.0 ;    // set PA.0 as input
...
swapc   pa.0 ;    // read PA.0 to C (bit operation)
src     a ;        // shift C to bit 7 of ACC
swapc   pa.0 ;    // read PA.0 to C (bit operation)
src     a ;        // shift new C to bit 7, old C
...

```

7-6. Conditional Operation Instructions

<i>ceqsn</i> a, I	<p>Compare ACC with immediate data and skip next instruction if both are equal. Flag will be changed like as ($a \leftarrow a - I$) Example: ceqsn a, 0x55; inc MEM; goto error; Result: If a=0x55, then “goto error”; otherwise, “inc MEM”. Affected flags: 『Y』 Z 『Y』 C 『Y』 AC 『Y』 OV</p>
<i>ceqsn</i> a, M	<p>Compare ACC with memory and skip next instruction if both are equal. Flag will be changed like as ($a \leftarrow a - M$) Example: ceqsn a, MEM; Result: If a=MEM, skip next instruction Affected flags: 『Y』 Z 『Y』 C 『Y』 AC 『Y』 OV</p>
<i>ceqsn</i> M, a	<p>Compare ACC with memory and skip next instruction if both are equal. Example: ceqsn MEM, a; Result: If a=MEM, skip next instruction Affected flags: 『Y』 Z 『Y』 C 『Y』 AC 『Y』 OV</p>
<i>cneqsn</i> a, M	<p>Compare ACC with memory and skip next instruction if both are not equal. Flag will be changed like as ($a \leftarrow a - M$) Example: cneqsn a, MEM; Result: If a≠MEM, skip next instruction Affected flags: 『Y』 Z 『Y』 C 『Y』 AC 『Y』 OV</p>

<i>cneqsn a, l</i>	<p>Compare ACC with immediate data and skip next instruction if both are no equal. Flag will be changed like as ($a \leftarrow a - l$) Example: <i>cneqsn a,0x55 ;</i> <i>inc MEM ;</i> <i>goto error ;</i> Result: If $a \neq 0x55$, then “goto error”; Otherwise, “inc MEM”. Affected flags: 『Y』 Z 『Y』 C 『Y』 AC 『Y』 OV</p>
<i>t0sn IO.n</i>	<p>Check IO bit and skip next instruction if it's low Example: <i>t0sn pa.5;</i> Result: If bit 5 of port A is low, skip next instruction Affected flags: 『N』 Z 『N』 C 『N』 AC 『N』 OV</p>
<i>t1sn IO.n</i>	<p>Check IO bit and skip next instruction if it's high Example: <i>t1sn pa.5 ;</i> Result: If bit 5 of port A is high, skip next instruction Affected flags: 『N』 Z 『N』 C 『N』 AC 『N』 OV</p>
<i>t0sn M.n</i>	<p>Check memory bit and skip next instruction if it's low Example: <i>t0sn MEM.5 ;</i> Result: If bit 5 of MEM is low, then skip next instruction Affected flags: 『N』 Z 『N』 C 『N』 AC 『N』 OV</p>
<i>t1sn M.n</i>	<p>Check memory bit and skip next instruction if it's high EX: <i>t1sn MEM.5 ;</i> Result: If bit 5 of MEM is high, then skip next instruction Affected flags: 『N』 Z 『N』 C 『N』 AC 『N』 OV</p>
<i>izsn a</i>	<p>Increment ACC and skip next instruction if ACC is zero Example: <i>izsn a;</i> Result: $a \leftarrow a + 1$, skip next instruction if $a = 0$ Affected flags: 『Y』 Z 『Y』 C 『Y』 AC 『Y』 OV</p>
<i>dzsn a</i>	<p>Decrement ACC and skip next instruction if ACC is zero Example: <i>dzsn a;</i> Result: $A \leftarrow A - 1$, skip next instruction if $a = 0$ Affected flags: 『Y』 Z 『Y』 C 『Y』 AC 『Y』 OV</p>
<i>izsn M</i>	<p>Increment memory and skip next instruction if memory is zero Example: <i>izsn MEM;</i> Result: $MEM \leftarrow MEM + 1$, skip next instruction if $MEM = 0$ Affected flags: 『Y』 Z 『Y』 C 『Y』 AC 『Y』 OV</p>
<i>dzsn M</i>	<p>Decrement memory and skip next instruction if memory is zero Example: <i>dzsn MEM;</i> Result: $MEM \leftarrow MEM - 1$, skip next instruction if $MEM = 0$ Affected flags: 『Y』 Z 『Y』 C 『Y』 AC 『Y』 OV</p>

<i>wait0</i> IO.n	<p>Wait here until bit n of IO port is low.</p> <p>Example: <i>wait0 pa.5;</i></p> <p>Result: Wait bit 5 of port A low to execute next instruction;</p> <p>Affected flags: 『N』 Z 『N』 C 『N』 AC 『N』 OV</p>
<i>wait1</i> IO.n	<p>Wait here until bit n of IO port is low.</p> <p>Example: <i>wait1 pa.5;</i></p> <p>Result: Wait bit 5 of port A high to execute next instruction;</p> <p>Affected flags: 『N』 Z 『N』 C 『N』 AC 『N』 OV</p>

7-7. System control Instructions

<i>call</i> label	<p>Function call, address can be full range address space</p> <p>Example: <i>call function1;</i></p> <p>Result: [sp] ← pc + 1 pc ← function1 sp ← sp + 2</p> <p>Affected flags: 『N』 Z 『N』 C 『N』 AC 『N』 OV</p>
<i>goto</i> label	<p>Go to specific address which can be full range address space</p> <p>Example: <i>goto error;</i></p> <p>Result: Go to error and execute program.</p> <p>Affected flags: 『N』 Z 『N』 C 『N』 AC 『N』 OV</p>
<i>delay</i> a	<p>Delay the (N + 1) cycles which N is specified by the content of ACC, the timing is based on the executing FPP unit. After the <i>delay</i> instruction is executed, the ACC will be zero.</p> <p>Example: <i>delay a;</i></p> <p>Result: Delay 16 cycles here if ACC=0fh</p> <p>Affected flags: 『N』 Z 『N』 C 『N』 AC 『N』 OV</p> <p>Note: Because ACC is the temporarily buffer for counting, please make sure that it will not be interrupted when executing this instruction. Otherwise, it may be not the expected delay time.</p>
<i>delay</i> I	<p>Delay the (N + 1) cycles which N is specified by the immediate data, the timing is based on the executing FPP unit. After the <i>delay</i> instruction is executed, the ACC will be zero.</p> <p>Example: <i>delay 0x05;</i></p> <p>Result: Delay 6 cycles here</p> <p>Affected flags: 『N』 Z 『N』 C 『N』 AC 『N』 OV</p> <p>Note: Because ACC is the temporarily buffer for counting, please make sure that it will not be interrupted when executing this instruction. Otherwise, it may be not the expected delay time.</p>

<i>delay</i> M	<p>Delay the (N + 1) cycles which N is specified by the content of memory, the timing is based on the executing FPP unit. After the <i>delay</i> instruction is executed, the ACC will be zero.</p> <p>Example: <i>delay M;</i></p> <p>Result: Delay 256 cycles here if M=ffh</p> <p>Affected flags: 『N』 Z 『N』 C 『N』 AC 『N』 OV</p> <p>Note: Because ACC is the temporarily buffer for counting, please make sure that it will not be interrupted when executing this instruction. Otherwise, it may be not the expected delay time.</p>
<i>ret</i> I	<p>Place immediate data to ACC, then return</p> <p>Example: <i>ret 0x55;</i></p> <p>Result: A ← 55h</p> <p><i>ret ;</i></p> <p>Affected flags: 『N』 Z 『N』 C 『N』 AC 『N』 OV</p>
<i>ret</i>	<p>Return to program which had function call</p> <p>Example: <i>ret;</i></p> <p>Result: sp ← sp - 2 pc ← [sp]</p> <p>Affected flags: 『N』 Z 『N』 C 『N』 AC 『N』 OV</p>
<i>reti</i>	<p>Return to program that is interrupt service routine. After this command is executed, global interrupt is enabled automatically.</p> <p>Example: <i>reti;</i></p> <p>Affected flags: 『N』 Z 『N』 C 『N』 AC 『N』 OV</p>
<i>nop</i>	<p>No operation</p> <p>Example: <i>nop;</i></p> <p>Result: nothing changed</p> <p>Affected flags: 『N』 Z 『N』 C 『N』 AC 『N』 OV</p>
<i>pcadd</i> a	<p>Next program counter is current program counter plus ACC.</p> <p>Example: <i>pcadd a;</i></p> <p>Result: pc ← pc + a</p> <p>Affected flags: 『N』 Z 『N』 C 『N』 AC 『N』 OV</p> <p>Application Example:</p> <pre> ----- ... mov a, 0x02 ; pcadd a ; // PC <- PC+2 goto err1 ; goto correct ; // jump here goto err2 ; goto err3 ; ... correct: // jump here ----- </pre>

<i>engint</i>	<p>Enable global interrupt enable</p> <p>Example: engint;</p> <p>Result: Interrupt request can be sent to FPP0</p> <p>Affected flags: 『N』 Z 『N』 C 『N』 AC 『N』 OV</p>
<i>disgint</i>	<p>Disable global interrupt enable</p> <p>Example: disgint ;</p> <p>Result: Interrupt request is blocked from FPP0</p> <p>Affected flags: 『N』 Z 『N』 C 『N』 AC 『N』 OV</p>
<i>stopsys</i>	<p>System halt.</p> <p>Example: stopsys;</p> <p>Result: Stop the system clocks and halt the system</p> <p>Affected flags: 『N』 Z 『N』 C 『N』 AC 『N』 OV</p>
<i>stopexe</i>	<p>CPU halt. The oscillator modes is still active to output clock, however, system clock is disabled to save power.</p> <p>Example: stopexe;</p> <p>Result: Stop the system clocks and keep oscillator modes active.</p> <p>Affected flags: 『N』 Z 『N』 C 『N』 AC 『N』 OV</p>
<i>reset</i>	<p>Reset the whole chip, its operation will be same as hardware reset.</p> <p>Example: reset;</p> <p>Result: Reset the whole chip.</p> <p>Affected flags: 『N』 Z 『N』 C 『N』 AC 『N』 OV</p>
<i>wdreset</i>	<p>Reset Watchdog timer.</p> <p>Example: wdreset ;</p> <p>Result: Reset Watchdog timer.</p> <p>Affected flags: 『N』 Z 『N』 C 『N』 AC 『N』 OV</p>

7-8. Summary of Instructions Execution Cycle

If two FPP units are selected:

2T	<i>ldtabh, ldtabl, idxm</i>
1T	Others

If single FPP unit is selected:

2T	<i>goto, call, ldtabh, ldtabl, idxm</i>
1T+S	<i>ceqsn, cneqsn, t0sn, t1sn, dzsn, izsn</i>
1T	Others

Where: 1T + S means that it is 2T if skip condition happen, otherwise, it is 1T and continues executing down.

7-9. Summary of affected flags by Instructions

Instruction	Z	C	AC	OV	Instruction	Z	C	AC	OV	Instruction	Z	C	AC	OV
<i>mov a, l</i>	-	-	-	-	<i>mov M, a</i>	-	-	-	-	<i>mov a, M</i>	Y	-	-	-
<i>mov a, IO</i>	Y	-	-	-	<i>mov IO, a</i>	-	-	-	-	<i>nmov M, a</i>	-	-	-	-
<i>nmov a, M</i>	Y	-	-	-	<i>ldtabh index</i>	-	-	-	-	<i>ldtabl</i>	-	-	-	-
<i>ldt16 word</i>	-	-	-	-	<i>stt16 word</i>	-	-	-	-	<i>xch M</i>	-	-	-	-
<i>idxm a, index</i>	-	-	-	-	<i>idxm index,</i>	-	-	-	-	<i>pushaf</i>	-	-	-	-
<i>popaf</i>	Y	Y	Y	Y	<i>add a, l</i>	Y	Y	Y	Y	<i>add a, M</i>	Y	Y	Y	Y
<i>add M, a</i>	Y	Y	Y	Y	<i>addc a, M</i>	Y	Y	Y	Y	<i>addc M, a</i>	Y	Y	Y	Y
<i>addc a</i>	Y	Y	Y	Y	<i>addc M</i>	Y	Y	Y	Y	<i>nadd a, M</i>	Y	Y	Y	Y
<i>nadd M, a</i>	Y	Y	Y	Y	<i>sub a, l</i>	Y	Y	Y	Y	<i>sub a, M</i>	Y	Y	Y	Y
<i>sub M, a</i>	Y	Y	Y	Y	<i>subc a, M</i>	Y	Y	Y	Y	<i>subc M, a</i>	Y	Y	Y	Y
<i>subc a</i>	Y	Y	Y	Y	<i>subc M</i>	Y	Y	Y	Y	<i>inc M</i>	Y	Y	Y	Y
<i>dec M</i>	Y	Y	Y	Y	<i>clear M</i>	-	-	-	-	<i>sr a</i>	-	Y	-	-
<i>src a</i>	-	Y	-	-	<i>sr M</i>	-	Y	-	-	<i>src M</i>	-	Y	-	-
<i>sl a</i>	-	Y	-	-	<i>slc a</i>	-	Y	-	-	<i>sl M</i>	-	Y	-	-
<i>slc M</i>	-	Y	-	-	<i>swap a</i>	-	-	-	-	<i>swap M</i>	-	-	-	-
<i>and a, l</i>	Y	-	-	-	<i>and a, M</i>	Y	-	-	-	<i>and M, a</i>	Y	-	-	-
<i>or a, l</i>	Y	-	-	-	<i>or a, M</i>	Y	-	-	-	<i>or M, a</i>	Y	-	-	-
<i>xor a, l</i>	Y	-	-	-	<i>xor a, IO</i>	Y	-	-	-	<i>xor IO, a</i>	-	-	-	-
<i>xor a, M</i>	Y	-	-	-	<i>xor M, a</i>	Y	-	-	-	<i>not a</i>	Y	-	-	-
<i>not M</i>	Y	-	-	-	<i>neg a</i>	Y	-	-	-	<i>neg M</i>	Y	-	-	-
<i>comp a, l</i>	Y	Y	Y	Y	<i>comp a,</i>	Y	Y	Y	Y	<i>comp M,</i>	Y	Y	Y	Y
<i>set0 IO.n</i>	-	-	-	-	<i>set1 IO.n</i>	-	-	-	-	<i>tog IO.n</i>	-	-	-	-
<i>set0 M.n</i>	-	-	-	-	<i>set1 M.n</i>	-	-	-	-	<i>swapc</i>	-	Y	-	-
<i>ceqsn a, l</i>	Y	Y	Y	Y	<i>ceqsn a, M</i>	Y	Y	Y	Y	<i>ceqsn M, a</i>	Y	Y	Y	Y
<i>cneqsn a, M</i>	Y	Y	Y	Y	<i>cneqsn a, l</i>	Y	Y	Y	Y	<i>t0sn IO.n</i>	-	-	-	-
<i>t1sn IO.n</i>	-	-	-	-	<i>t0sn M.n</i>	-	-	-	-	<i>t1sn M.n</i>	-	-	-	-
<i>izsn a</i>	Y	Y	Y	Y	<i>dzsn a</i>	Y	Y	Y	Y	<i>izsn M</i>	Y	Y	Y	Y
<i>dzsn M</i>	Y	Y	Y	Y	<i>wait0 IO.n</i>	-	-	-	-	<i>wait1 IO.n</i>	-	-	-	-
<i>call label</i>	-	-	-	-	<i>goto label</i>	-	-	-	-	<i>delay a</i>	-	-	-	-
<i>delay l</i>	-	-	-	-	<i>delay M</i>	-	-	-	-	<i>ret l</i>	-	-	-	-
<i>ret</i>	-	-	-	-	<i>reti</i>	-	-	-	-	<i>nop</i>	-	-	-	-
<i>pcadd a</i>	-	-	-	-	<i>engint</i>	-	-	-	-	<i>disgint</i>	-	-	-	-
<i>stopsys</i>	-	-	-	-	<i>stopexe</i>	-	-	-	-	<i>reset</i>	-	-	-	-
<i>wdreset</i>	-	-	-	-										

8. Special Notes

This chapter is to remind user who use PMC234/PMS234 series IC in order to avoid frequent errors upon operation.

8-1. Using IC

8-1-1. IO pin usage and setting

(1) Description of PBDIER

There are no PCDIER and PDDIER registers in PMC234/PMS234, therefore, enabling the digital input of PC/PD pins should be configured through PB pin and share PBDIER register accordingly. For example:

PC[0:3] and PB.0 are configured by PBDIER.0

PC[4:7] and PB.1 are configured by PBDIER.1

PD[0:1] and PB.2 are configured by PBDIER.2

When PBDIER.0 = 1, certain IOs of PB.0 and PC[0:3] that are configured as input will enable the digital input and wake-up function.

(2) IO pin as analog input

- ◆ Configure IO pin as input
- ◆ Set PADIER and PBDIER registers to configure corresponding IO as analog input
- ◆ Set PAPH and PBPH registers to disable corresponding IO pull-up resistor
- ◆ The functions of PADIER and PBDIER registers of PMC234/PMS234 series IC is contrary to ICE functions

Please use following program in order to keep ICE emulation consisting with PMC234/PMS234 series IC procedure.

```
$ PADIER 0xF0;
```

```
$ PBDIER 0x0F;
```

(3) PA5 as output pin

PA5 can only be Open-Drain output pin. Output high requires adding pull-up resistor.

(4) PA5 as PRST# input pin

- ◆ No internal pull-up resistor for PA5
- ◆ Configure PA5 as input
- ◆ Set CLKMD.0=1 to enable PA5 as PRST# input pin

(5) PA5 as input pin to connect with a push button or a switch by a long wire

- ◆ Needs to put a >10Ω resistor in between PA5 and the long wire
- ◆ Avoid using PA5 as input

(6) PA7 and PA6 as external crystal oscillator

- ◆ Configure PA7 and PA6 as input
- ◆ Disable PA7 and PA6 internal pull-up resistor

- ◆ Configure PADIER register to set PA6 and PA7 as analog input
- ◆ EOSCR register bit [6:5] selects corresponding crystal oscillator frequency :
 - ◇ 01 : for lower frequency, ex : 32kHz
 - ◇ 10 : for middle frequency, ex : 455kHz、 1MHz
 - ◇ 11 : for higher frequency, ex : 4MHz
- ◆ Program EOSCR.7 =1 to enable crystal oscillator
- ◆ Ensure EOSC working well before switching from IHRC or ILRC to EOSC, refer to [8.1.3.\(2\)](#)

8-1-2. Interrupt

(1) When using the interrupt function, the procedure should be:

Step1: Set INTEN register, enable the interrupt control bit

Step2: Clear INTRQ register

Step3: In the main program, using ENGINT to enable CPU interrupt function

Step4: Wait for interrupt. When interrupt occurs, enter to Interrupt Service Routine

Step5: After the Interrupt Service Routine being executed, return to the main program

* Use DISGINT in the main program to disable all interrupts

* When interrupt service routine starts, use PUSHAF instruction to save ALU and FLAG register. POPAF instruction is to restore ALU and FLAG register before RETI as below:

```
void Interrupt (void) // Once the interrupt occurs, jump to interrupt service
routine
{
    // enter DISGINT status automatically, no more interrupt is
    accepted
    PUSHAF;
    ...
    POPAF;
} // RETI will be added automatically. After RETI being executed, ENGINT
status will be restored
```

(2) INTEN and INTRQ have no initial values. Please set required value before enabling interrupt function

(3) FPPA1 will not be affected by interrupt at all

8-1-3. System clock switching

(1) S ystem clock can be switched by CLKMD register. Please notice that, **NEVER switch the system clock and turn off the original clock source at the same time**. For example: When switching from clock A to clock B, please switch to clock B first; and after that turn off the clock A oscillator through CLKMD.

- ◆ Case 1 : Switch system clock from ILRC to IHRC/2

```
CLKMD = 0x36; // switch to IHRC, ILRC can not be disabled here
```

```
CLKMD.2 = 0; // ILRC can be disabled at this time
```

- ◆ Case 2 : Switch system clock from ILRC to EOSC

```
CLKMD = 0xA6; // switch to EOSC, ILRC can not be disabled here
```

```
CLKMD.2 = 0; // ILRC can be disabled at this time
```

- ◆ **ERROR.** Switch ILRC to IHRC and turn off ILRC simultaneously

```
CLKMD = 0x50; // MCU will hang
```

- (2) Please ensure the EOSC oscillation has established before switching from ILRC or IHRC to EOSC. MCU will not check its status. Please wait for a while after enabling EOSC. System clock can be switched to EOSC afterwards. Otherwise, MCU will hang. The example for switching system clock from ILRC to 4MHz EOSC after boot up is as below:

```
.ADJUST_IC DISABLE
```

```
CLKMD.1 = 0; // turn off WDT for executing delay instruction.
```

```
$ EOSCR Enable, 4MHz; // 4MHz EOSC start to oscillate.
```

```
delay 255 // Delay for EOSC establishment
```

```
CLKMD = 0xA4; // ILRC -> EOSC;
```

```
CLKMD.2 = 0; // turn off ILRC only if necessary
```

The delay duration should be adjusted in accordance with the characteristic of the crystal and PCB.

To measure the oscillator signal by the oscilloscope, please select (x10) on the probe and measure through PA6(X2) pin to avoid the interference on the oscillator.

8-1-4. Power down mode, wakeup and watchdog

- (1) Watchdog will be inactive once ILRC is disabled
- (2) Please turn off watchdog before executing STOPSYS or STOPEXE instruction, otherwise IC will be reset due to watchdog timeout. It is the same as in ICE emulation.
- (3) The clock source of Watchdog is ILRC if the fast wakeup is disabled; otherwise, *the clock source of Watchdog will be the system clock and the reset time becomes much shorter.* It is recommended to disable Watchdog and enable fast wakeup before entering STOPSYS mode. When the system is waken up from power down mode, please firstly disable fast wakeup function, and then enable Watchdog. It is to avoid system to be reset after being waken up.
- (4) If enable Watchdog during programming and also wants the fast wakeup, the example as below:

```
CLKMD.En_WatchDog = 0; // disable watchdog timer
```

```
$ MISC Fast_Wake_Up;
```

```
stopexe;
```

```
nop;
```

```
$ MISC WT_xx; // Reset Watchdog time to normal wake-up
```

```
Wdreset;
```

```
CLKMD.En_WatchDog = 1; // enable watchdog timer
```

8-1-5. TIMER time out

- (1) When select T16M counter BIT8 as 1 to generate interrupt, the first interrupt will occur when the counter reaches to 0x100 (BIT8 from 0 to 1) and the second interrupt will occur when the counter reaches 0x300 (BIT8 from 0 to 1). Therefore, selecting BIT8 as 1 to generate interrupt means that the interrupt occurs every 512 counts. Please notice that if T16M counter is restarted, the next interrupt will occur once Bit8 turns from 0 to 1.
- (2) When Timer2 is set to be 6bit mode and enabled, interrupt will occur not every time but one of every four times of Timer2 timeout.

8-1-6. Using ADC

- (1) Configure corresponding IO as input through PxDIER register
- (2) The recommended highest ADC conversion frequency is 500kHz and maximum output impedance of analog signal source is 10K Ω .
- (3) Never restart next conversion before completion of last ADC conversion; otherwise, wrong value would get.
- (4) Please pay attention on sequence of operation if the program fits for below conditions,
 1. Use the FPP (ex. FPPA0) for handling power-save mode to disable ADC.
 2. Use the FPP (ex. FPPA1) for handling ADC conversion to enable ADC and wait for completion of ADC conversion with **WAIT1 ADC_Done** instruction.
 3. Execute above **【1】** & **【2】** simultaneously

In case the above sequence is not properly arranged, there may be a chance that FPPA1 can not pass through **WAIT1 ADC_Done** instruction because the ADC may be disabled by FPPA0 before **WAIT1 ADC_Done** instruction being executed.

Recommendation:

Define a Flag which represents the operation of ADC. Every time FPPA1 set the flag when enable the ADC and reset it when the completion of ADC conversion. FPPA0 checks this flag and decides to enter power-save and disable ADC if it is reset.

8-1-7. LVR

- (1) VDD must reach or above 2.2V for successful power-on process; otherwise IC will be inactive.
- (2) The setting of LVR (1.8V, 2.0V, 2.2V etc) will be valid just after successful power-on process.

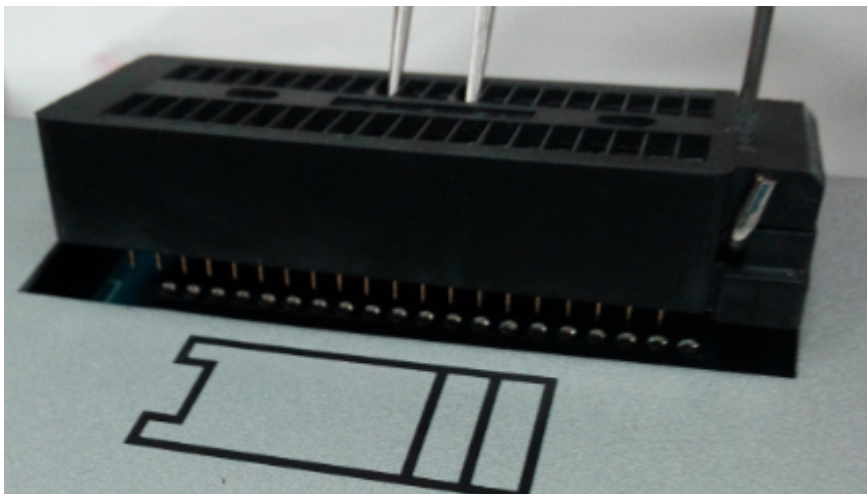
8-1-8. Differences in command timing between single / double FPPA mode

Differences on PMC234/PMS234 series instruction cycle

Instruction	Condition	1 FPPA	2 FPPA
goto, call		2T	1T
ceqsn, cneqsn, t0sn, t1sn, dzsn, izsn	Condition is fulfilled	2T	1T
	Condition is not fulfilled	1T	1T
ldtabh, ldtabl, idxm		2T	2T
Others		1T	1T

8-1-9. Programming 28pin package

- (1) Connect to the Jumper CN35 on writer;
- (2) Unplug the Textool;
- (3) Rotate the Textool by 180° and re-plug in with shifting 1 pin upward as below:



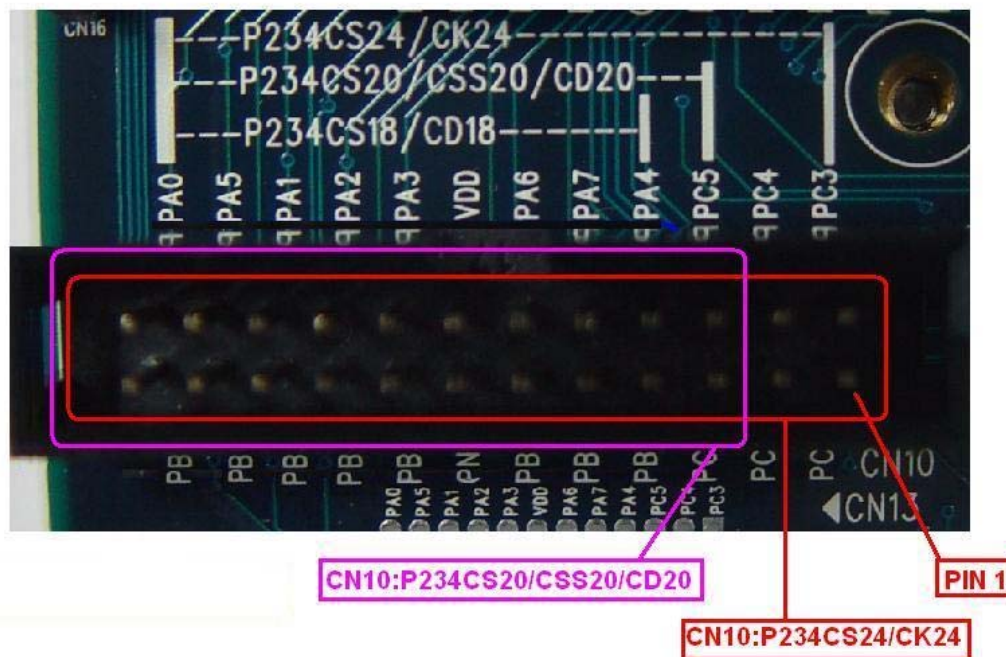
- (4) Put the IC at the top (as 24pin) and start programming

8-2. Using ICE

8-2-1. Emulating PMC234/PMS234 series IC on ICE PDK3S-I-001/002/003

PMC234/PMS234 series I/O pin is defined as compatible with P234C series, but it has 4 more I/O pins than P234C. When using ICE PDK3S-I-001/002 to emulate PMC234/PMS234 series IC, please according to different number of pin used, use cable labeled CN10 or with jumper wire to connect to CN10 connector on ICE PDK3S-I-001/002.

- (1) Emulating PMC234/PMS234(SOP20/DIP20) : Use cable labeled CN10:P234CS20/CSS20/CD20 to connect CN10 connector on ICE PDK3S-I-001/002. Connection is shown as below.
- (2) Emulating PMC234/PMS234(SOP24/DIP24) : Use cable labeled CN10:P234CS24/CK24 to connect CN10 connector on ICE PDK3S-I-001/002. Connection is also shown as below.



- (3) Emulating PMC234/PMS234(SOP28/DIP28) : Use cable labeled CN10:P234CS24/CK24 to connect CN10 connector on ICE PDK3S-I-001/002 as (1) and (2). But the extra 4 I/O pins PC6, PC7, PD0 and PD1 required jumper wire on CN11.

8-2-2. Important Notice for ICE operation

PDK3S-I-001/002/003 **does not support** emulating below PMC234/PMS234 series IC functions from (a) to (g). Thus, user needs to take PMC234/PMS234 Real Chip for test. Please notice : In order to avoid the problems on difference between ICE and Real Chip test procedure, those functions will be temporarily removed from datasheet before ICE is set to support them. PMC234/PMS234 Real Chip is defaulted with these functions and performing ordinarily. User is recommended to consider these exceptions and careful handle whenever doing the test.

- (a) PMC234/PMS234 series IC is able to set LVD minimum to 1.8V, total 8 stages 4.0V, 3.5V, 3.0V, 2.75V, 2.5V, 2.2V, 2.0V, 1.8V.
- (b) PMC234/PMS234 series IC LVD function can be disabled by register (misc.2).
- (c) PMC234/PMS234 series IC is able to support LVD reset and fast-recover by register (misc.3).
- (d) PMC234/PMS234 series IC is able to be set as single core operating model.
- (e) PMC234/PMS234 series IC supports VDD less than 4V, 3V, 2V voltage level detection and store detecting result to internal register (rstst).
- (f) PMC234/PMS234 series IC supports reset-source detection.
- (g) Watch-dog series IC provides watch-dog time out function which is assigned by register misc[1:0]