PMC232/PMS232 Series

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

Data Sheet

Version 1.04 – Dec. 18, 2018

6F-6, No.1, Sec. 3, Gongdao 5th Rd., Hsinchu City 30069, Taiwan, R.O.C.

TEL: 886-3-572-8688 www.padauk.com.tw

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 content

Revision History:

Revision	Date	Description
0.01	2015/08/01	1st version
0.02	2015/10/30	1. Add 1-4. Package Information: PMC232-Y24A 2. Add 3. PMC232-Y24A Pin Assignment and Description
0.03	2017/03/27	1. Add 8-1. Warning 2. Add 8-2-8. IHRC Description
1.04	2018/12/18	1. Updated company address & Tel No. 2. Amend 1-1, 1-2, 1-3 3. Add AVDD and AGND in Chapter 3 4. Amend 4-1 AC/DC Device Characteristics 5. Amend 4-3 to 4-12 6. Add 4-13 Typical power down current (I_{PD}) and power save current (I_{PS}) 7. Amend 5-7 Oscillator and clock 8. Amend 5-7-1, 5-7-3, 5-7-4, 5-7-5 9. Amend 5-9 8-bit Timer (Timer2) with PWM generation 10. Amend 5-9-1, 5-9-2 11. Amend 5-11 Interrupt 12. Amend 5-12-1, 5-12-2, 5-12-3 13. Amend Table 7: Differences in wake-up sources between Power-Save mode and Power-Down mode 14. Amend Fig.18: Analog Input Model 15. Amend 6-7, 6-12, 6-13 16. Amend Section 7-8 Summary of Instructions Execution Cycle and delete 9-2-9 17. Add Chapter 8 Special Notes 18. Updated the link in Section 9-1 19. Amend 9-2-1, 9-2-5 20. Add 9-2-9 Program writing 21. Amend 9-3 Using ICE

Major Differences between P232C and PMC232/PMS232

There are many differences between P232C and PMC232/PMS232. The table below only shows the major differences between them.

Item	Function	P232C	PMC232/PMS232
1	IO capability	12mA@5.0V	10mA@5.0V
2	SRAM	200 bytes	160 bytes
3	Band-gap	+/- 200mV(@1.20V)	+/- 60mV(@1.20V) after calibration
4	LVR	4 levels LVR setting	8 levels LVR setting
5	Single FPPA mode	N/A	Yes
6	LCD Half VDD bias voltage	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	***padidr*** , ***pbdidr, pcdidr***	***padier*** , ***pbdier***
10	IHRC option command	.ADJUST_OTP_IHRCR	.ADJUST_IIC
11	WDT timeout	512 ILRC clock cycles	4 selectable periods
12	Hardware Comparator	Yes	N/A

Procedure for converting code from P232C to PMC232/PMS232

Please follow the below steps for converting code from P232C to PMC232/PMS232:

1. Go through the PMC232/PMS232 datasheet and user guide;
2. Modify the source code engineering file ".pre"; change ".chip P232CXXX" to ".chip PMC232" or ".chip PMS232"
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

- PMC232 series:
  - High EFT series
  - Operating temperature range: -40°C ~ 85°C
- PMS232 series:
  - General purpose series
  - Not supposed to use in AC RC step-down powered or high EFT requirement applications.
    PADAUK assumes no liability if such kind of applications can not pass the safety regulation tests.
  - Operating temperature range: -20°C ~ 70°C

1-2. System Functions

- 2Kx16 bits OTP program memory for both FPP units
- 160 Bytes data RAM for both FPP units
- One hardware 16-bit timer
- Support fast wake-up
- 18 IO pins with 10mA driving / sink capability
- Every IO pin can be configured to enable wake-up function
- Band-gap circuit to provide 1.20V reference voltage
- Up to 10-channel 12-bit resolution ADC with 1-channel for internal band-gap reference voltage
- One hardware 8-bit timer with PWM generator
- Provide software configurable LCD driver IO with optional VDD/2 LCD bias voltage
- Provide maximum 4x13 dots LCD display
- Operating voltage range: 2.2V ~ 5.5V
- Clock sources: internal high RC oscillator, internal low RC oscillator and external crystal oscillator
- Eight levels of LVR reset ~ 4.1V, 3.6V, 3.1V, 2.8V, 2.5V, 2.2V, 2.0V, 1.8V
- Two external interrupt pins

1-3. High Performance RISC CPU Array

- Operating modes: Two processing units FPPA™ mode or Traditional one processing unit mode
- 100 powerful instructions
- Most instructions are 1T execution cycle
- Programmable stack pointer to provide adjustable stack level
- Direct and indirect addressing modes for data access. Data memories are available for use as an index pointer of Indirect addressing mode
- IO space and memory space are independent
- Separated IO and memory space

1-4. Package Information

◆ **PMC232 series**

PMC232 - S14: SOP14 (150mil);
PMC232 - S16A: SOP16 Type A (150mil);
PMC232 - S16B: SOP16 Type B (150mil);
PMC232 - Y24A: SSOP24 (150mil);
PMC232 - S18: SOP18 (300mil);
PMC232 - S20: SOP20 (300mil);
PMC232 - D14: DIP14 (300mil);
PMC232 - D18: DIP18 (300mil);
PMC232 - D20: DIP20 (300mil);

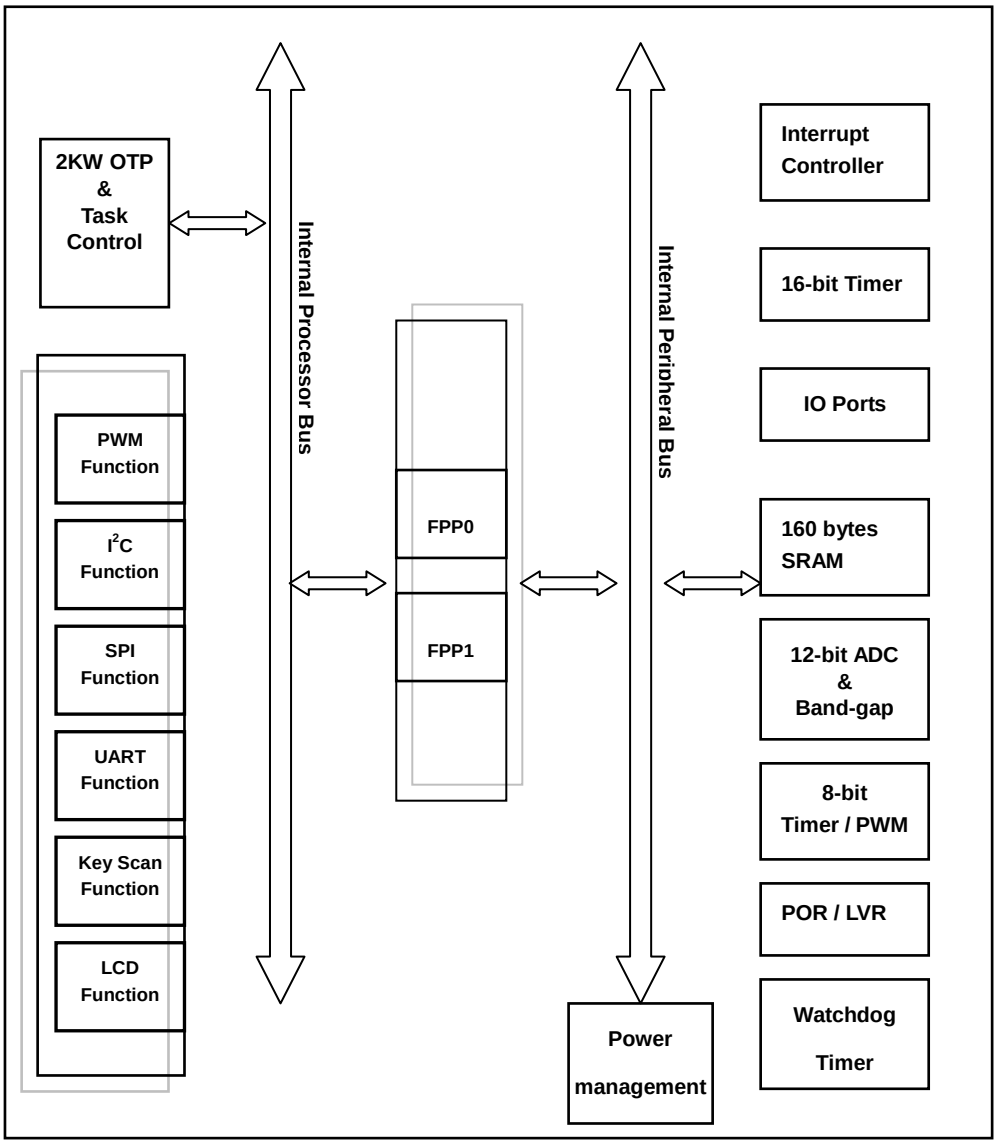
◆ **PMS232 series**

PMS232 - S14: SOP14 (150mil);
PMS232 - S16A: SOP16 Type A (150mil) ;
PMS232 - S16B: SOP16 Type B (150mil) ;
PMS232 - S18: SOP18 (300mil) ;
PMS232 - S20: SOP20 (300mil) ;
PMS232 - D14: DIP14 (300mil);
PMS232 - D18: DIP18 (300mil);
PMS232 - D20: DIP20 (300mil);

2. General Description and Block Diagram

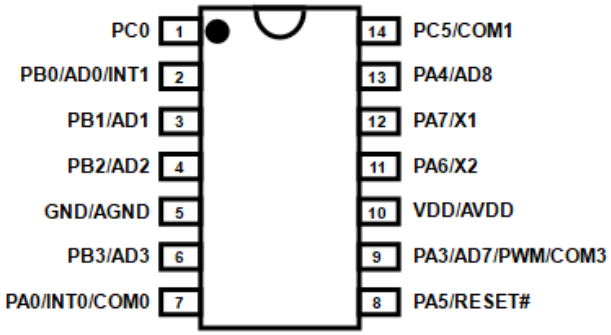
The PMC232/PMS232 family is an ADC-Type of PADAUK's parallel processing, fully static, OTP-based CMOS 2x8 bit processor array that can execute two peripheral functions in parallel. It employs RISC architecture based on patent pending FPPA™ (Field Programmable Processor Array) technology and all the instructions are executed in one cycle except that some instructions are two cycles that handle indirect memory access.

2Kx16 bits OTP program memory and 160 bytes data SRAM are inside for two FPP units using, one up to 10 channels 12-bit ADC is built inside the chip with one channel for internal band-gap reference voltage. PMC232/PMS232 also provides two hardware timers: one is 16-bit timer and the other one is 8-bit with PWM generation.

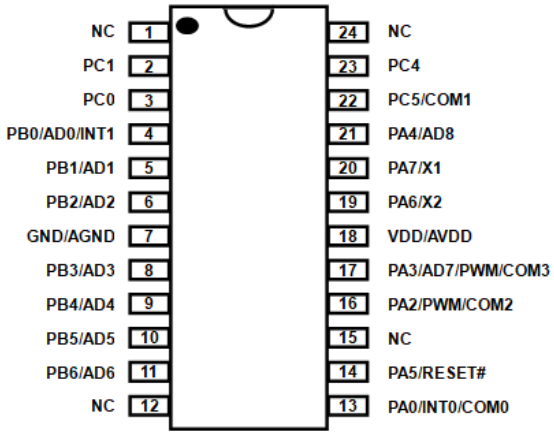


3. Pin Assignment and Description

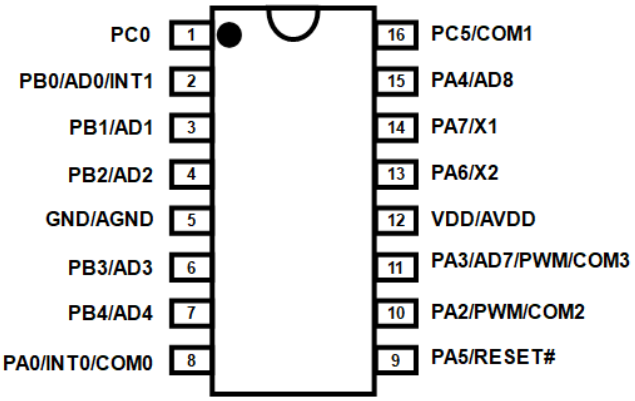
◆PMC232 series



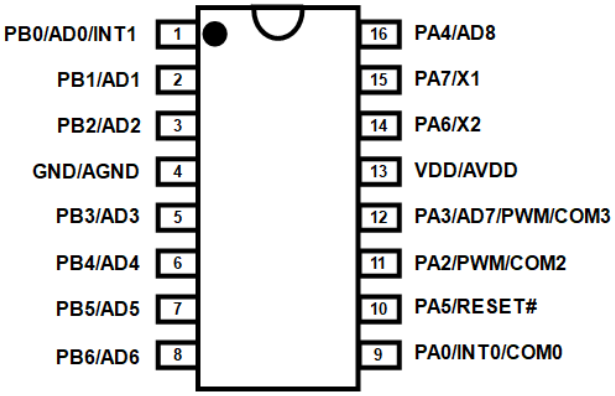
PMC232-S14 (SOP14-150mil)
PMC232-D14 (DIP14-300mil)



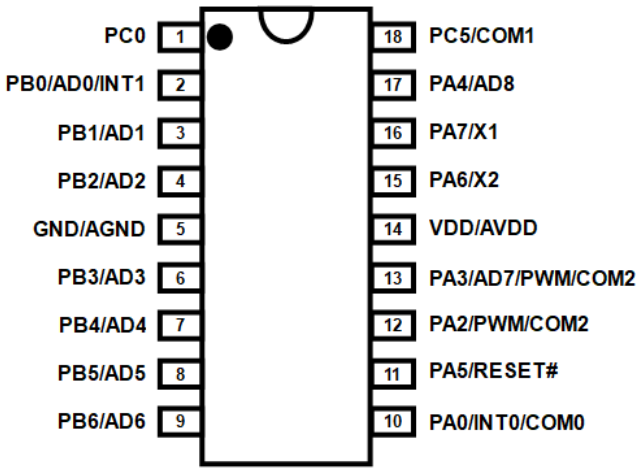
PMC232-Y24A (SSOP24-150mil)



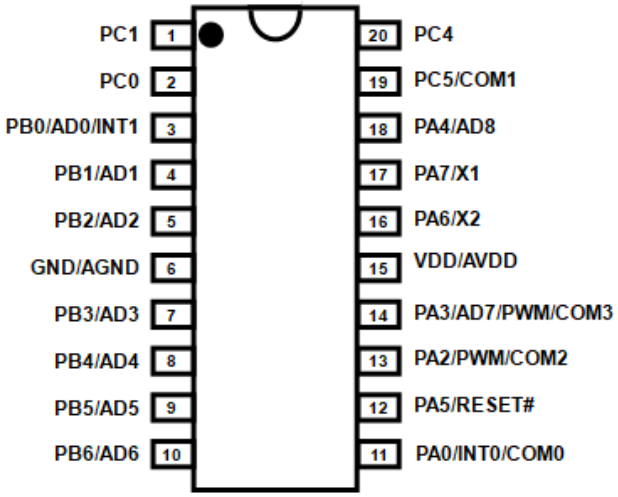
PMC232-S16A (SOP16A-150mil)



PMC232-S16B (SOP16B-150mil)

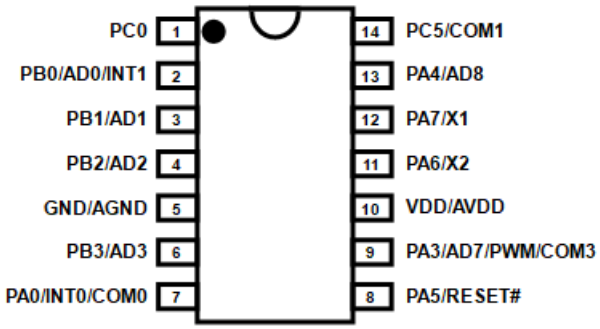


PMC232-S18 (SOP18-300mil)
PMC232-D18 (DIP18-300mil)

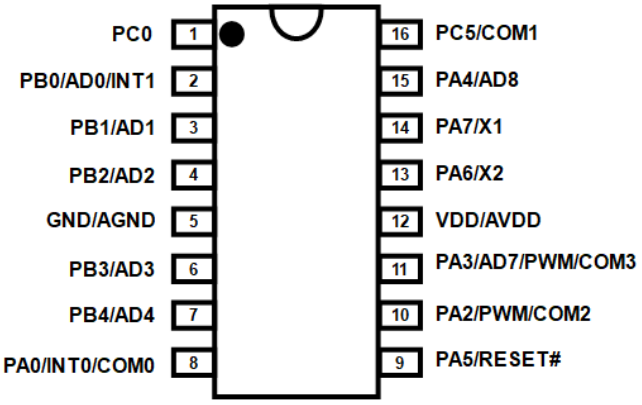


PMC232-S20 (SOP20-300mil)
PMC232-D20 (DIP20-300mil)

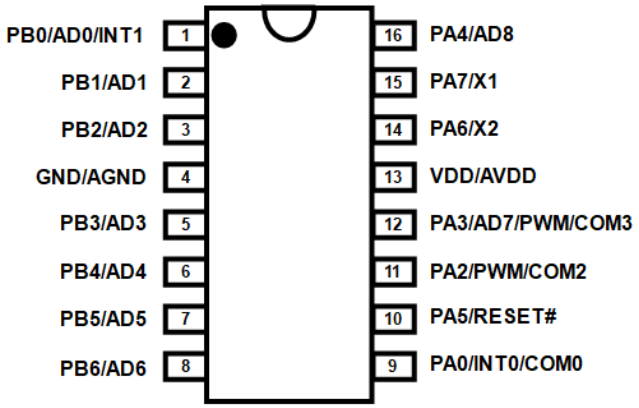
◆PMS232 series



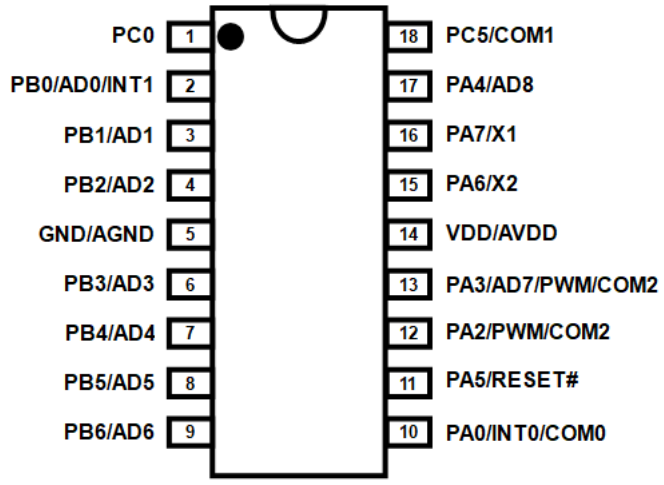
PMS232-S14 (SOP14-150mil)
PMS232-D14 (DIP14-300mil)



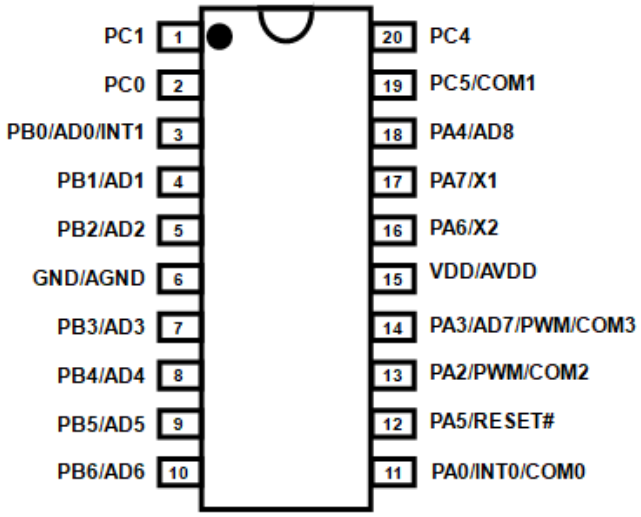
PMS232-S16A (SOP16A-150mil)



PMS232-S16B (SOP16B-150mil)



PMS232-S18 (SOP18-300mil)
PMS232-D18 (DIP18-300mil)



PMS232-S20 (SOP20-300mil)
PMS232-D20 (DIP20-300mil)

Pin Description

Pin Name	Pin Type & Buffer Type	Description
PA7 / X1	IO ST / CMOS	The functions of this pin can be: (1) Bit 7 of port A. It can be configured as digital input or two-state output, with pull-up resistor . (2) X1 when crystal oscillator is used. 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".
PA6 / X2	IO ST / CMOS	The functions of this pin can be: (1) Bit 6 of port A. It can be configured as digital input or two-state output, with pull-up resistor . (2) X2 when crystal oscillator is used. 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".
PA5 / RESET#	IO (OC) ST / CMOS	The functions of this pin can be: (1) Bit 5 of port A. It can be configured as digital input or open-drain output pin. Please notice that there is no pull-up resistor in this pin. (2) Hardware reset. 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". Please put 33Ω resistor in series to have high noise immunity when this pin is in input mode.
PA4 / AD8	IO ST / CMOS / Analog	The functions of this pin can be: (1) Bit 4 of port A. It can be configured as digital inputor two-state output, with pull-up resistor by software independently (2) Channel 8 of ADC analog input 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.
PA3 / AD7 / PWM / COM3	IO ST / CMOS / Analog	The functions of this pin can be: (1) Bit 3 of port A. It can be configured as digital inputor two-state output, with pull-up resistor independently by software (2) Channel 7 of ADC analog input (3) PWM output from Timer2 (4) COM3 for LCD. It can provide VDD/2 LCD bias voltage. 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.

Pin Name	Pin Type & Buffer Type	Description
PA2 / PWM / COM2	IO ST / CMOS	The functions of this pin can be: (1) Bit 2 of port A. It can be configured as digital input or 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. The bit 2 of ***padier*** register can be set to "0" to disable wake-up from power-down by toggling this pin.
PA0 / INT0 / COM0	IO ST / CMOS	The functions of this pin can be: (1) Bit 0 of port A. It can be configured as digital input or 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. Both rising edge and falling edge are accepted to request interrupt service and configurable by register setting. (3) COM0 for LCD. This pin is also the COM0 from LCD bias voltage generator. The bit 0 of ***padier*** register can be set to "0" to disable wake-up from power-down by toggling this pin.
PB6 / AD6 PB5 / AD5 PB4 / AD4 PB3 / AD3 PB2 / AD2 PB1 / AD1	IO ST / CMOS / Analog	The functions of these pins can be: (1) Bit 6~1 of port B. These six pins can each be configured as analog input, digital input, and two-state output mode with pull-up resistor independently by software. (2) Channel 6~1 of ADC analog input. When any of these six pins acts as analog inputs, please use register ***pbdier*** to disable the digital input to prevent current leakage. If the control bit in ***pbdier*** register is set to "0" to disable digital input, wake-up from power-down by toggling the corresponding pin is also disabled. When using PB2 as ADC input, please add one 0.1uF capacitance on it.
PB0 / AD0 / INT1	IO ST / CMOS / Analog	The functions of this pin can be: (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. (2) Channel 0 of ADC analog input. When this pin acts as analog input, please use bit 0 of register ***pbdier*** to disable the digital input to prevent current leakage. (3) External interrupt line 1. Both rising edge and falling edge are accepted to request interrupt service and configurable by register setting. If bit 0 of ***pbdier*** register is set to "0" to disable digital input, wake-up from power-down by toggling this pin is also disabled.
PC5 / COM1	IO ST / CMOS	The functions of this pin can be: (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. (2) COM1 for LCD. This pin is also the COM1 for LCD bias voltage generator

Pin Name	Pin Type & Buffer Type	Description
PC4 PC1	IO ST / CMOS	Bit 4, 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.
PC0	IO ST / CMOS	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.
VDD / AVDD	VDD/ AVDD	VDD: Digital positive power AVDD: Analog positive power VDD is the IC power supply while AVDD is the ADC power supply. AVDD and VDD are double bonding internally and they have the same external pin.
GND / AGND	GND / AGND	GND: Digital negative power AGND: Analog negative power GND is the IC ground pin while AGND is the ADC ground pin. AGND and GND are double bonding internally and they have the same external pin.
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

All data are acquired under the conditions of Ta = -40 °C ~ 85 °C, Vdd=5.0V, f_{SYS}=2MHz unless noted.

Symbol	Description	Min	Typ	Max	Unit	Conditions (Ta=25 °C)
V_{DD}	Operating Voltage	2.2	5.0	5.5	V	* Subject to LVR tolerance
f_{SYS}	System clock* = IHRC/2 IHRC/4 IHRC/8 ILRC	 0 0 0	 24K	 8M 4M 2M	Hz	Under_20ms_Vdd_ok**=Y/N V_{DD}≧3.0V V_{DD}≧2.5V V_{DD}≧2.2V V_{DD} = 5.0V
I_{OP}	Operating Current		1.7 15		mA uA	f_{SYS}=IHRC/16=1MIPS@5.0V f_{SYS}=ILRC=12kHz@3.3V
I_{PD}	Power Down Current (by ***stopsys*** command)		1.0 0.5		uA uA	f_{SYS}= 0Hz,VDD=5.0V f_{SYS}= 0Hz,VDD=3.3V
I_{PS}	Power Save Current (by ***stopexe*** command)		0.3		mA	V_{DD}=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		V_{DD}+0.3	V	
$I_{INJ\ (PIN)}$	Injected current on pin			1	mA	V_{DD}+0.3≧V_{IN}≧ -0.3
R_{PH}	Pull-up Resistance		62 100 210		KΩ	V_{DD}=5.0V V_{DD}=3.3V V_{DD}=2.2V
V_{LVR}	Low Voltage Detect Voltage ***** (Brown-out voltage)	3.86 3.35 2.84 2.61 2.37 2.04 1.86 1.67	4.15 3.60 3.05 2.80 2.55 2.20 2.00 1.80	4.44 3.85 3.26 3.00 2.73 2.35 2.14 1.93	V	
V_{BG}	Band-gap Reference Voltage (before calibration)	1.11	1.20	1.29	V	V_{DD}=5V, 25°C
	Band-gap Reference Voltage ***** (after calibration)	 1.140***** 1.145*****	 1.20***** 1.20*****	 1.260***** 1.255*****		V_{DD}=2.2V ~ 5.5V, -40°C <Ta<85°C* -20°C <Ta<70°C*
f_{IHRC}	Frequency of IHRC after calibration *****	15.76*	16*	16.24*	MHz	25°C, V_{DD}=2.2V~5.5V
		 14.72* 15.04*	 16* 16*	 17.28* 16.96*		V_{DD}=2.2V~5.5V, -40°C <Ta<85°C* -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	V_{DD}=5.0V, Ta=25°C
		15.6*	24*	32.4*		V_{DD}=5.0V, -40°C <Ta<85°C*
		16.8*	24*	31.2*		V_{DD}=5.0V, -20°C <Ta<70°C*
		10.2*	12*	13.8*		V_{DD}=3.3V, Ta=25°C
		4.55*	7*	9.45*		V_{DD}=2.2V, -40°C <Ta<85°C*
		4.90*	7*	9.10*		V_{DD}=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		V_{DD}	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 Non-Linearity		±2*		LSB	
AD INL	ADC Integral Non-Linearity		±4*		LSB	
ADos	ADC offset*		3		mV	
V_{DR}	RAM data retention voltage*	1.5			V	PMC232/PMS232 in stop mode.
$R_{(VDD/2)}$	(VDD/2) pull-up / pull-down resistance	2.5	5	10	KΩ	
$\Delta V_{(VDD/2)}$	Deviation of (VDD/2) output voltage		±1%	±3%		@V_{DD}=5V
t_{WDT}	Watchdog timeout period		2408		T_{ILRC}	misc[1:0]=00 (default)
			4096			misc[1:0]=01
			16384			misc[1:0]=10
			256			misc[1:0]=11
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}	Since IHRC starts up more quickly in the beginning, so the actual wake-up time will be shorter than standard 128 Sysclk
	Fast wake-up by IO toggle from STOPSYS suspend, ILRC is the system clock		128		T_{SYS}	Where T_{SYS} is ILRC-based time period of system clock
	Normal wake-up from STOPEXE or STOPSYS suspend		1024		T_{ILRC}	Where T_{ILRC} is the clock period of ILRC

Symbol	Description	Min	Typ	Max	Unit	Conditions (Ta=25°C)
t_{SBP}	System boot-up period from power-on		1024		T_{ILRC}	Where T_{ILRC} is the clock period of ILRC
t_{RST}	External reset pulse width	120			us	@V_{DD}=5V

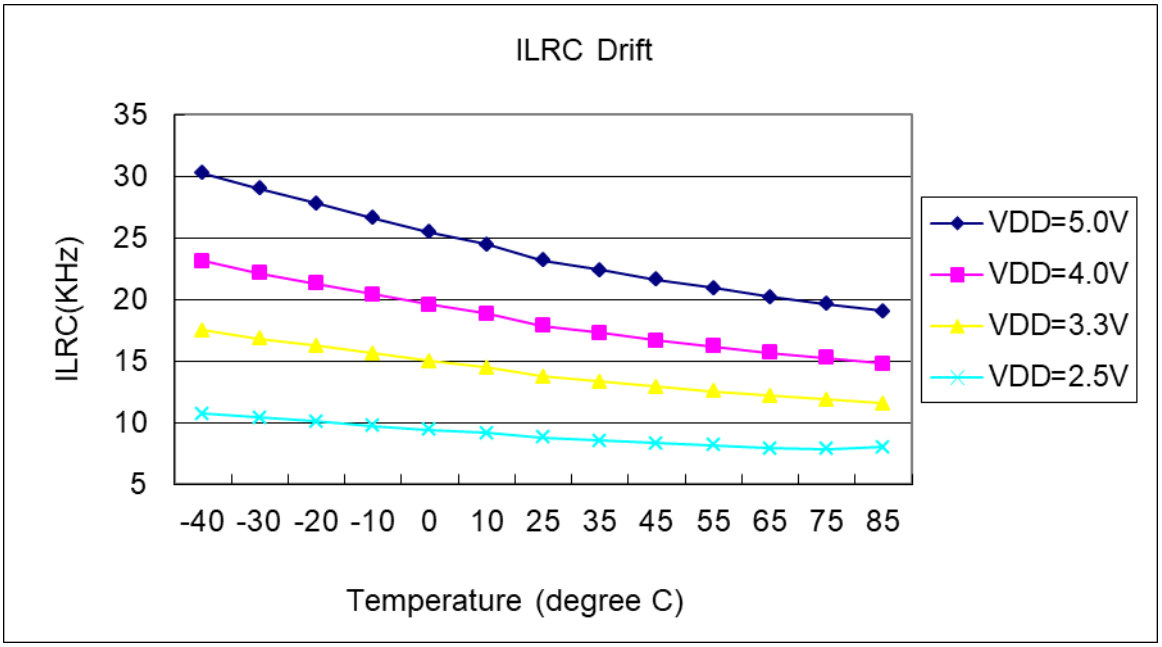
*These parameters are for design reference, not tested for each chip.

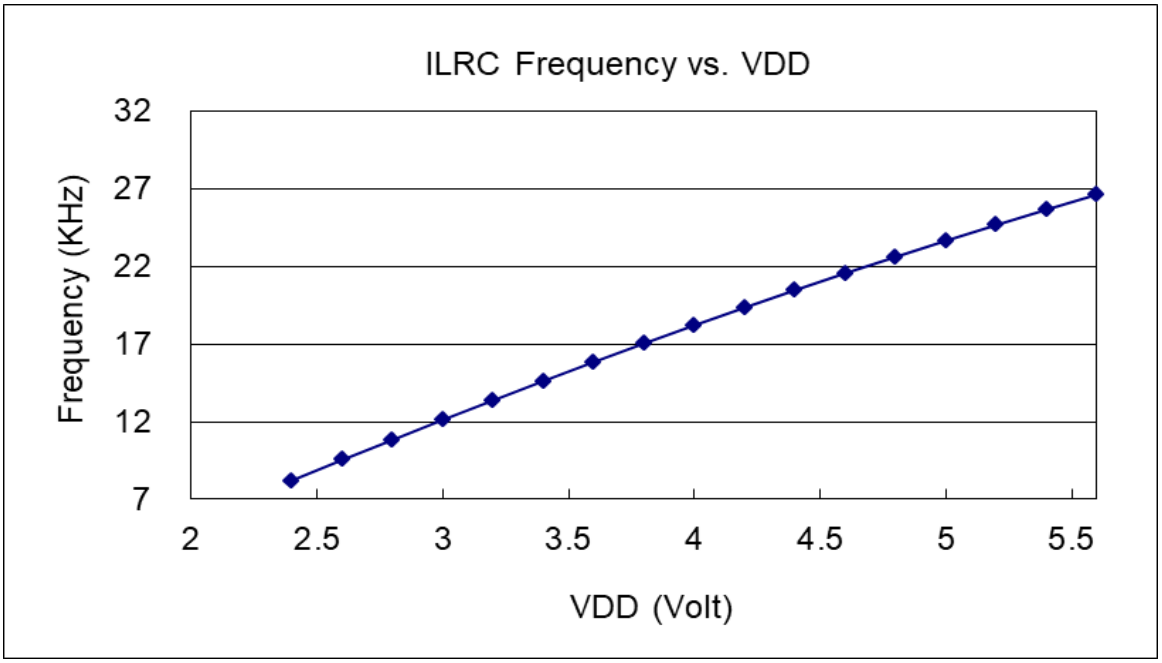
** Under_20ms_V_{DD}_Ok is a checking condition for the V_{DD} rising from 0V to the stated voltage within 20ms.

4-2. Absolute Maximum Ratings

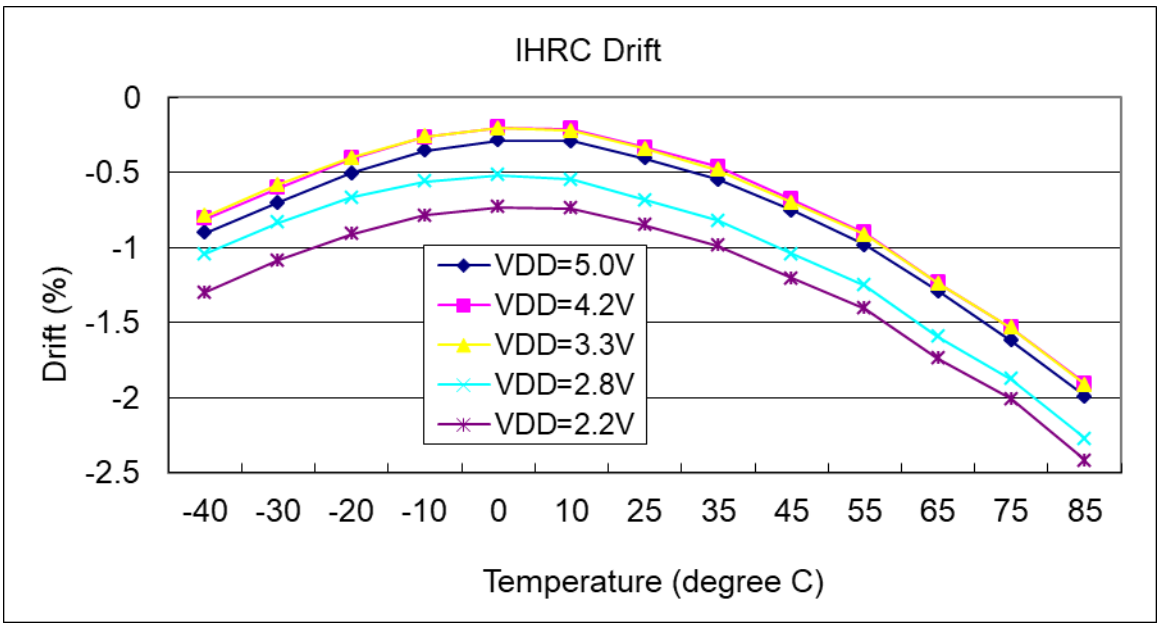
- Supply Voltage …… 2.2V ~ 5.5V
- Input Voltage …… -0.3V ~ VDD + 0.3V
- Operating Temperature …….. PMC232 Series: -40°C ~ 85°C
  PMS232 Series: -20°C ~ 70°C
- Junction Temperature …….. 150°C
- Storage Temperature …….. -50°C ~ 125°C

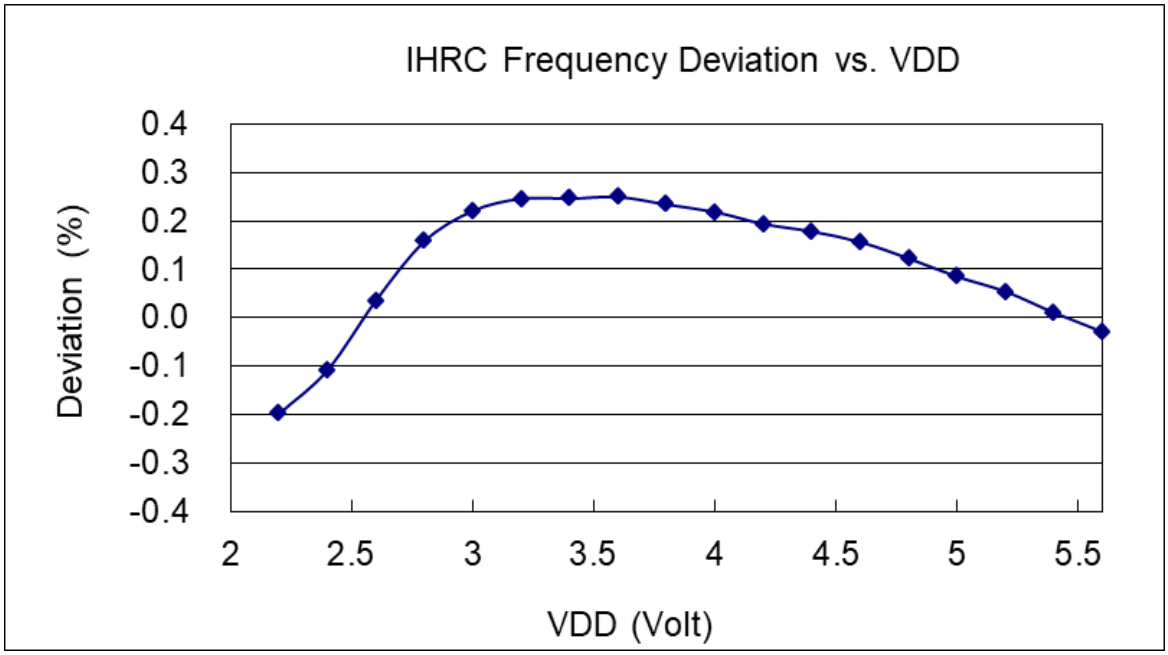
4-3. Typical ILRC frequency vs. VDD and temperature





4-4. Typical IHRC frequency deviation vs. VDD and temperature (calibrated to 16MHz)





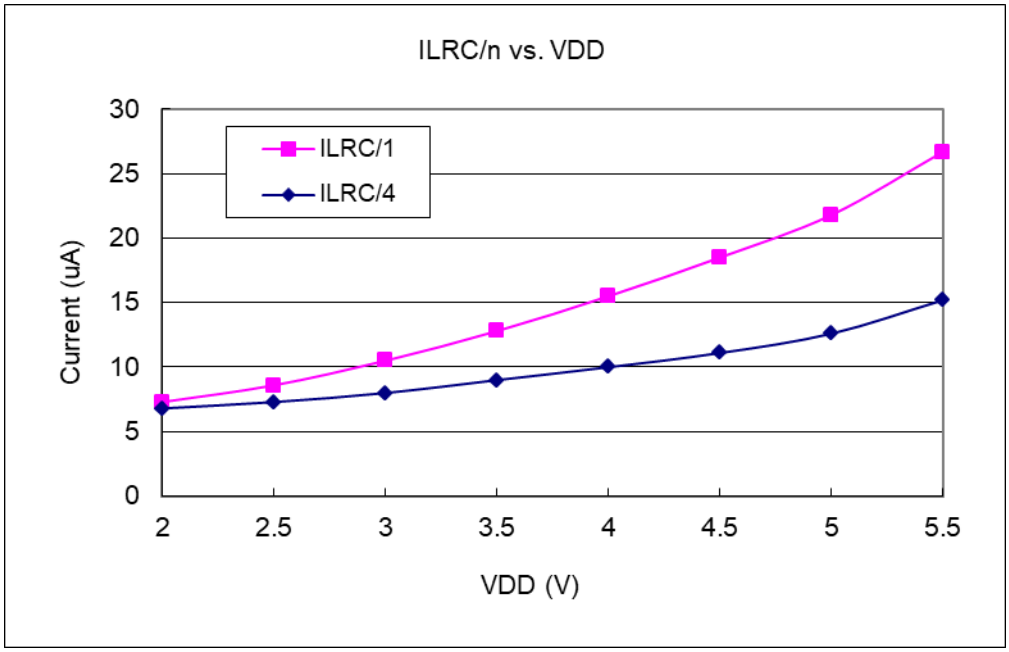
4-5. Typical operating current vs. VDD @ system clock = ILRC/n

Conditions:

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

ON: ILRC; **OFF**: Band-gap, LVR, IHRC, EOSC, T16, TM2, ADC modules;

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



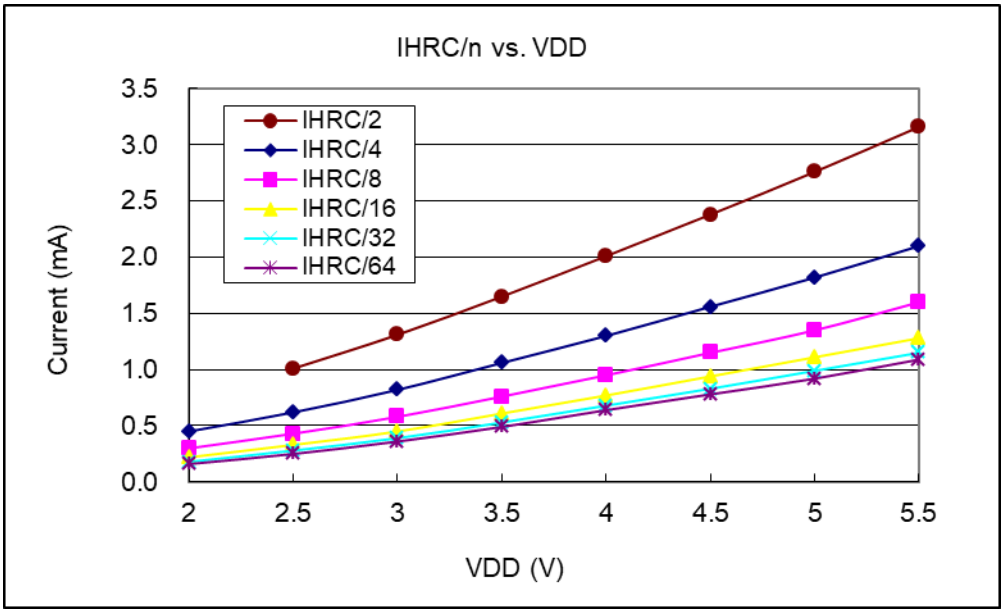
4-6. Typical operating current vs. VDD @ system clock = IHRC/n

Conditions:

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

ON: Band-gap, LVR, IHRC; **OFF**: ILRC, EOSC, T16, TM2, ADC modules;

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



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

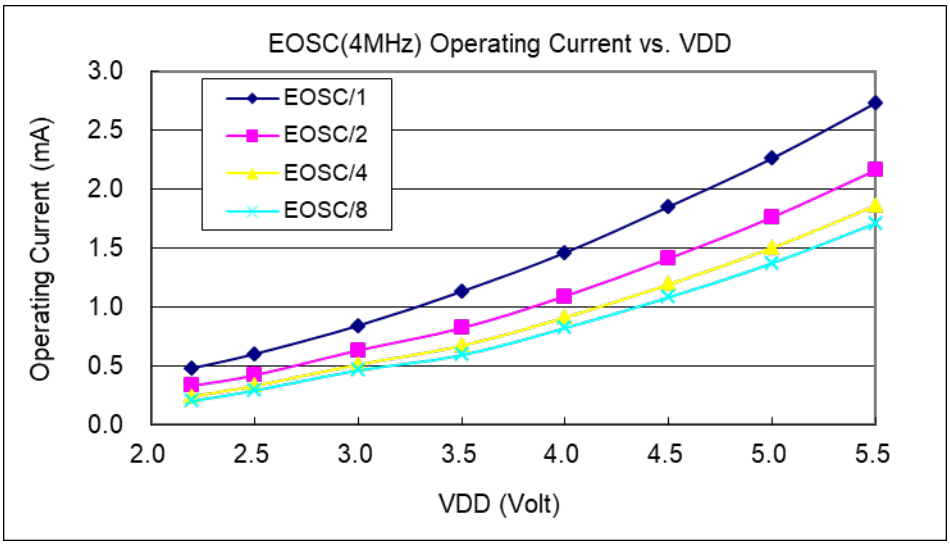
Conditions:

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

ON: EOSC, MISC.6 = 1;

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

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



4-8. Typical operating current vs. VDD @ system clock = 32kHz EOSC / n

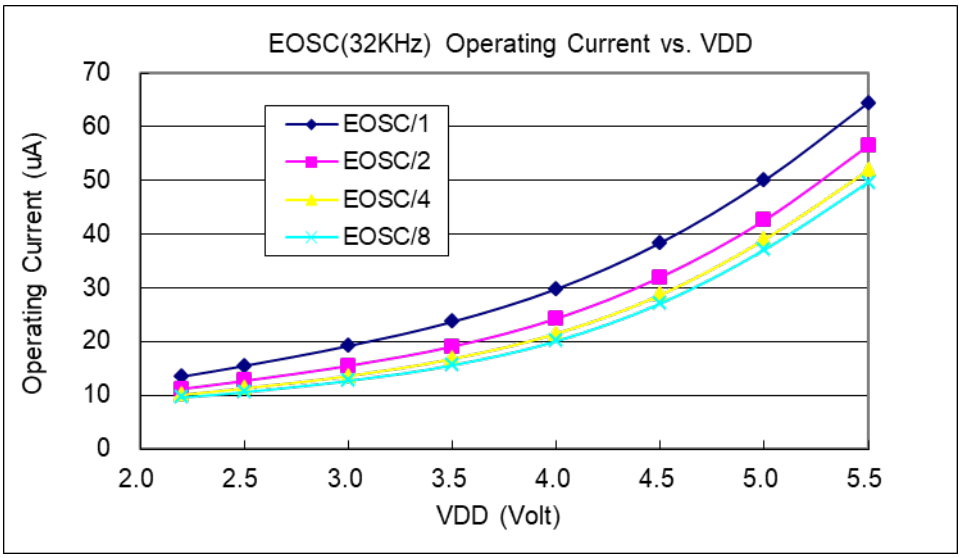
Conditions:

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

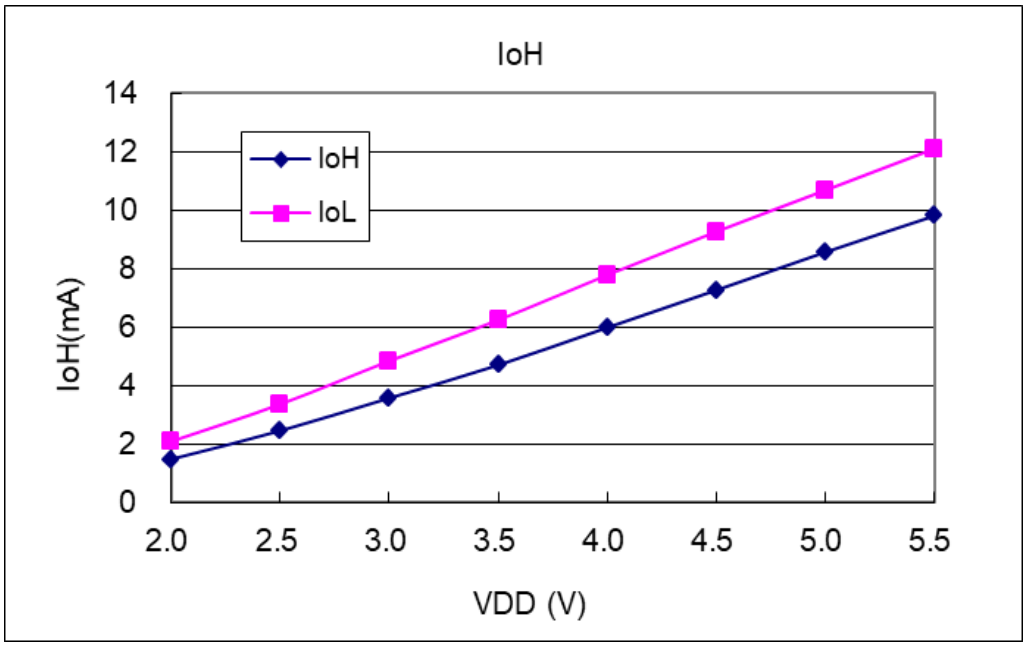
ON: EOSC, MISC.6 = 1;

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

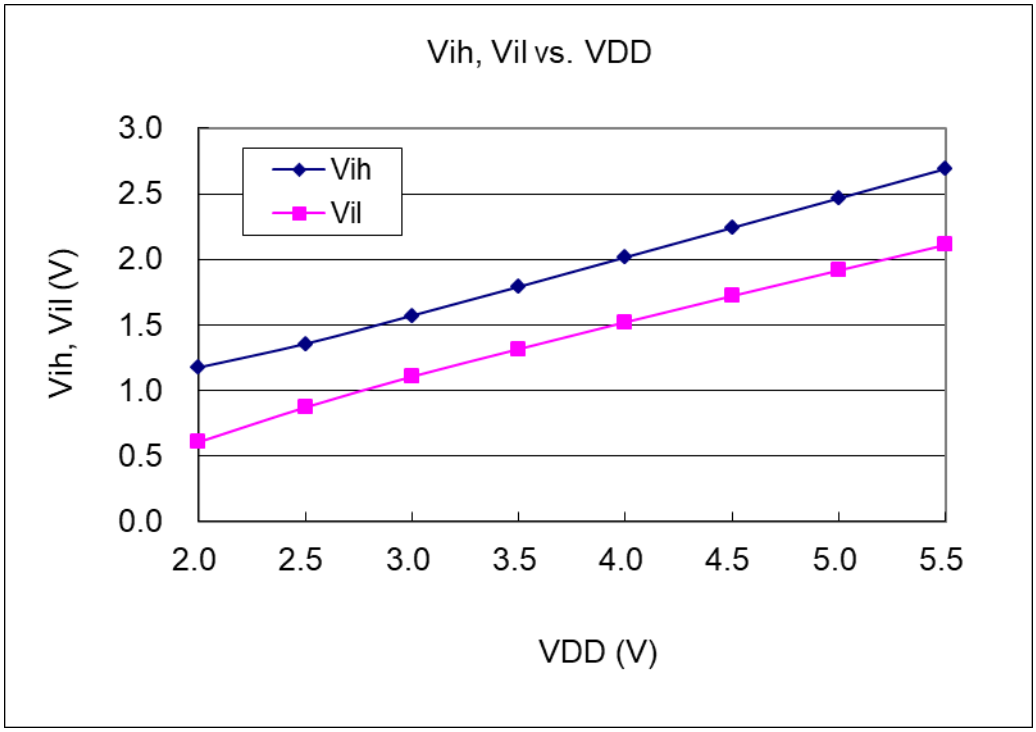
IO: PA0:0.5Hz output toggle and no loading, others: input and no floating



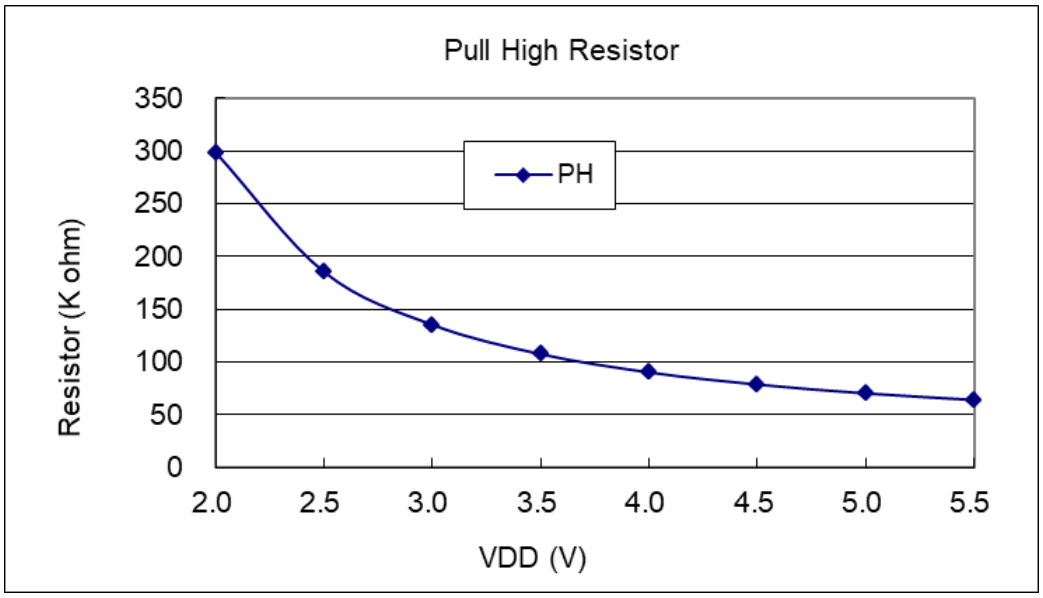
4-9. Typical IO driving current (I_{OH}) and sink current (I_{OL})



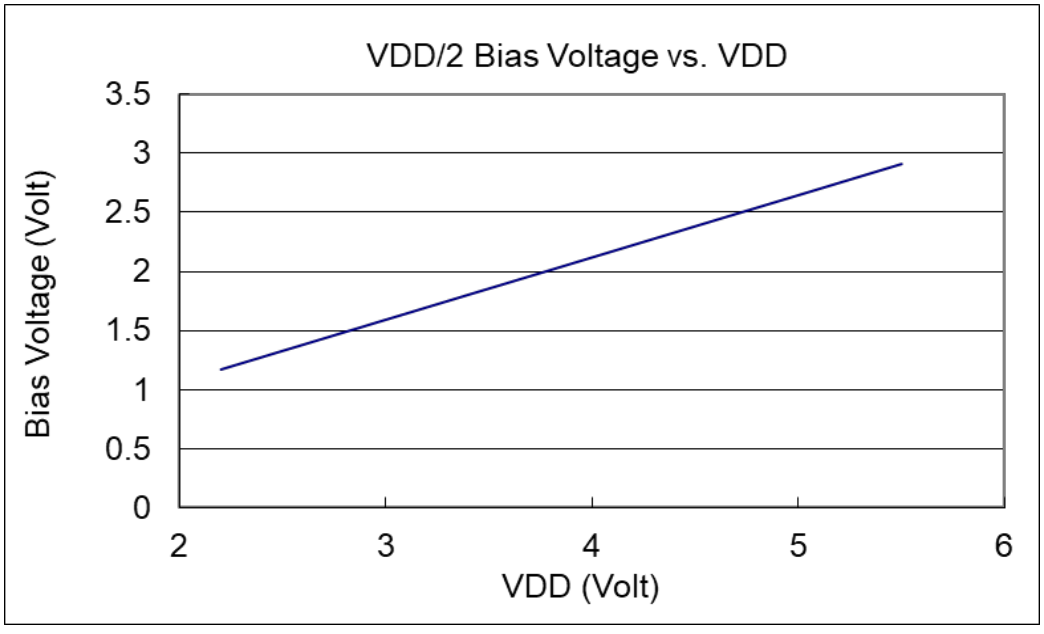
4-10. Typical IO input high/low threshold voltage (V_{IH}/V_{IL})



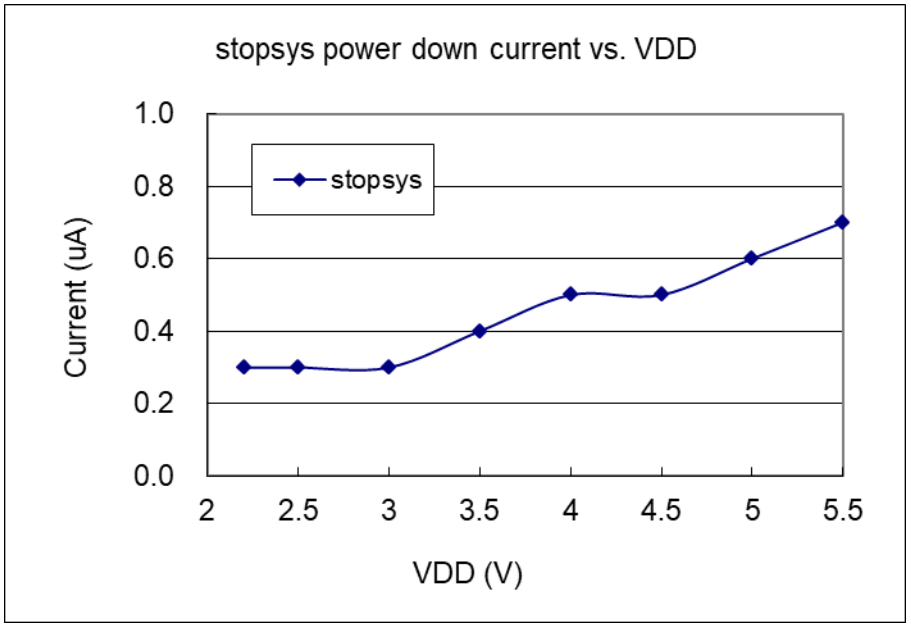
4-11. Typical resistance of IO pull high device

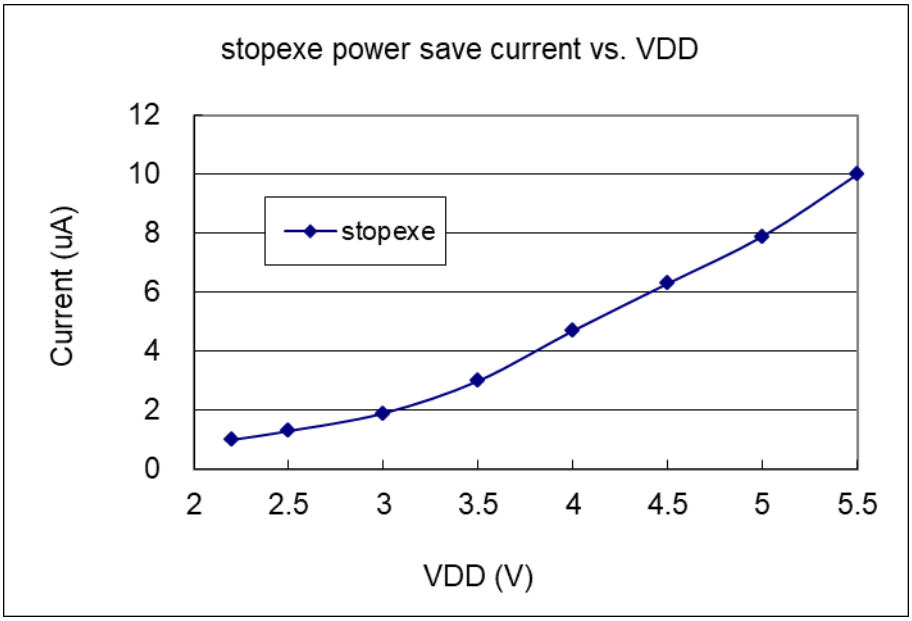


4-12. Typical VDD/2 Bias output voltage

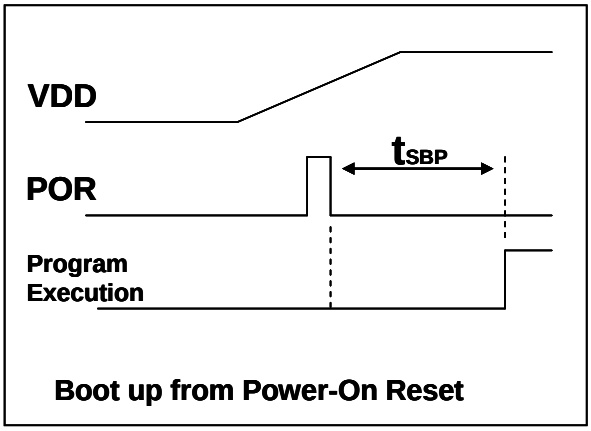


4-13. Typical power down current (I_{PD}) and power save current (I_{PS})

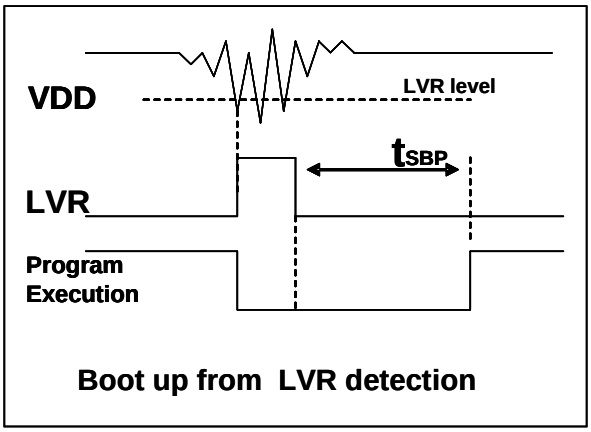




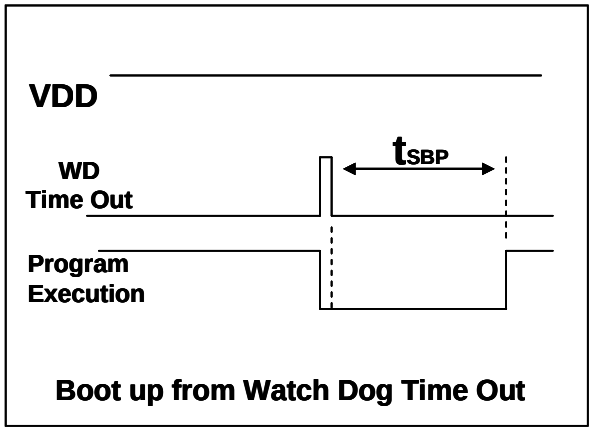
4-14. Timing charts for boot up conditions



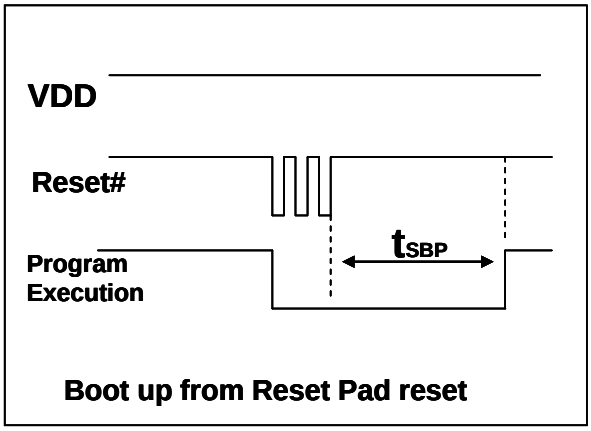
Boot up from Power-On Reset



Boot up from LVR detection



Boot up from Watch Dog Time Out



Boot up from Reset Pad reset

5. Functional Description

5-1. Processing Units

There are two processing units (FPP unit) inside the chip of PMC232/PMS232. 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 2k×16 bits OTP program memory, 160 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.

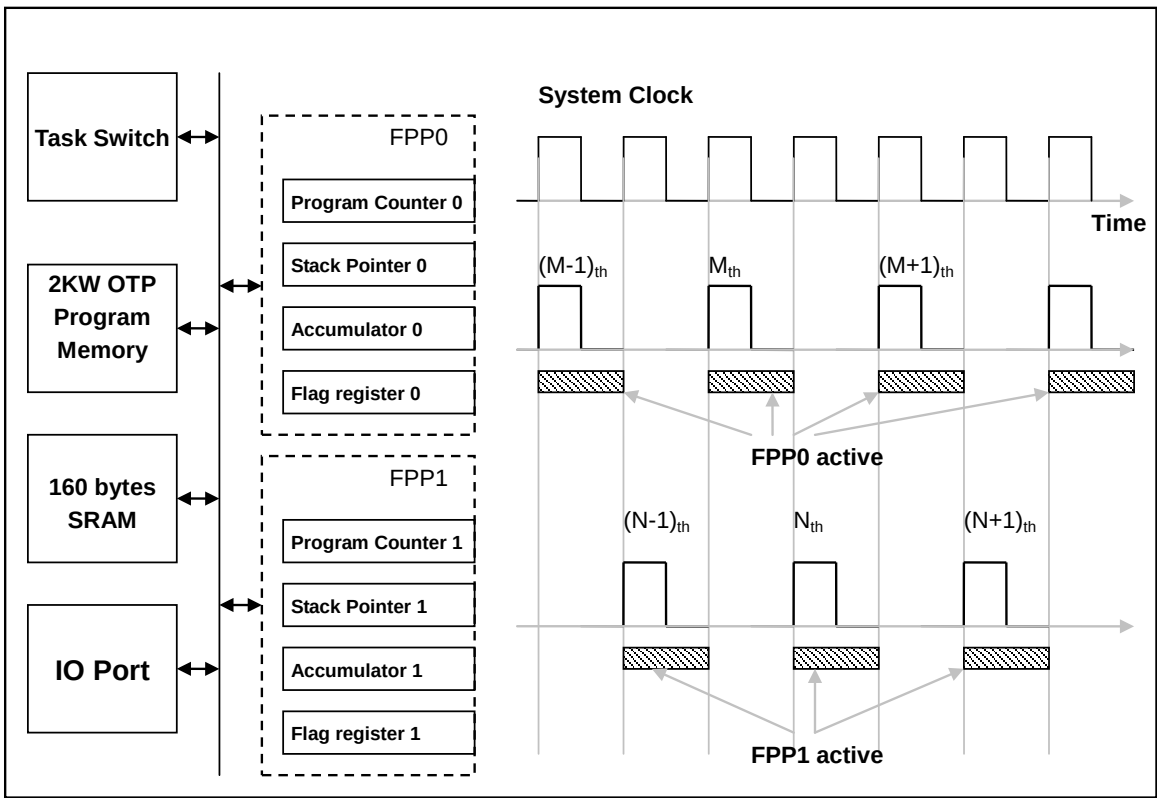


Fig.1: Hardware and Timing Diagram of FPP unit

Each FPP unit has half computing power of whole system; for example, FPP0 and FPP1will 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 PMC232/PMS232. 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 'h10 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 160 bytes and should be defined within 160 bytes from 0x00h address. The stack memory of PMC232/PMS232 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.

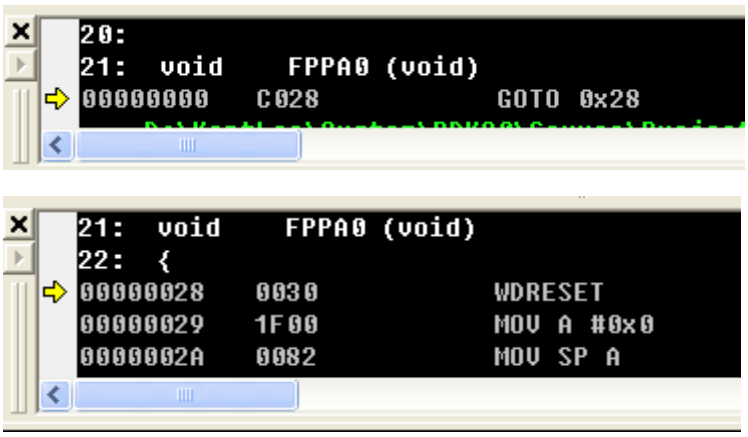


Fig.2: Stack Assignment in Mini-C project

5-1-3. Single FPP mode

For traditional MCU user who does not need parallel processing capability, PMC232/PMS232 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,. Please notice that ***wait*** and ***delay*** instructions are NOT supported when single FPP mode is chosen.

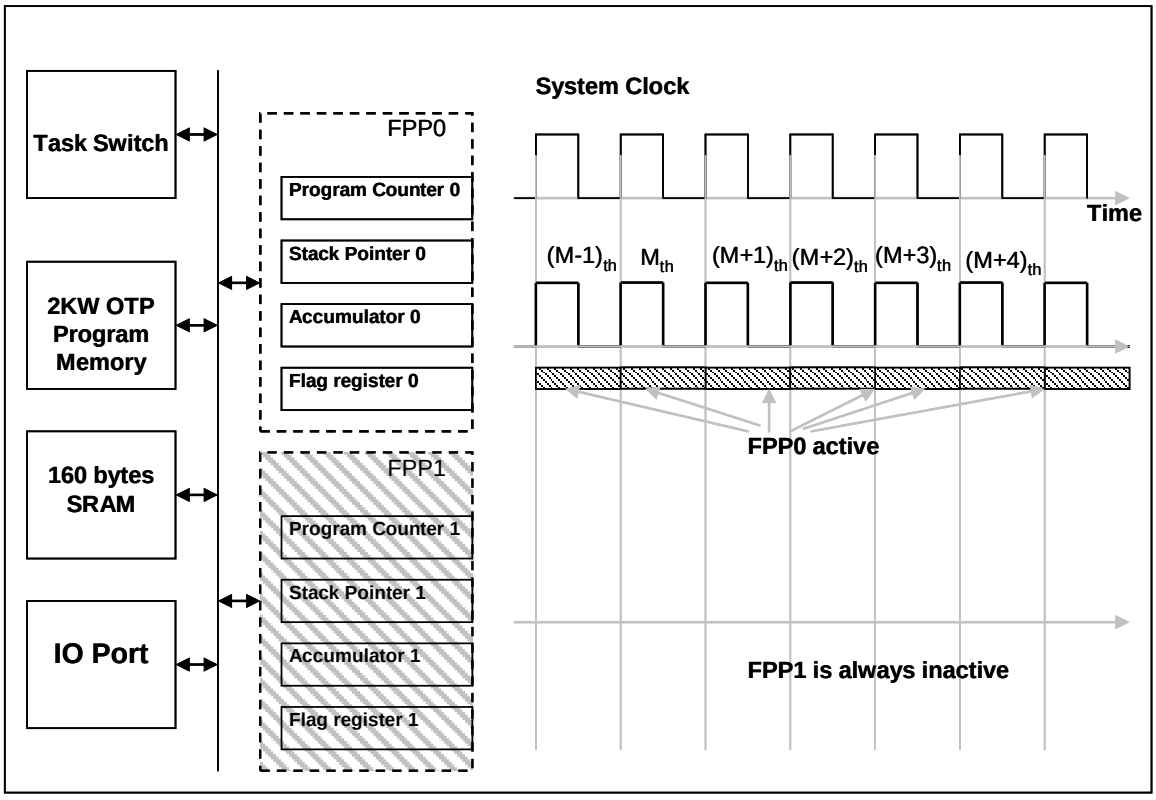


Fig.3: Timing Diagram of single FPP mode

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 ‘h0 and ‘h1 for FPP1. The interrupt entry is ‘h10 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 PMC232/PMS232 is a 2kx16 bit that is partitioned as Table 1. The OTP memory from address ‘h7F8 to ‘h7FF is for system using, address space from ‘h002 to ‘h00F and from ‘h011 to ‘h7F7 are user program space. The address ‘h001 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
•	•
0x7F7	User program
0x7F8	System Using
•	•
0x7FF	System Using

Table 1: Program Memory Organization of PMC232/PMS232

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 ‘h020)
001	Begin of FPP1 program
•	•
00F	goto ‘h1A1 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
•	•
7F7	End of FPP1 program
7F8	System Using
•	•
7FF	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
•	•
•	•
7F7	user program
7F8	System Using
•	•
7FF	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 PMC232/PMS232 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.

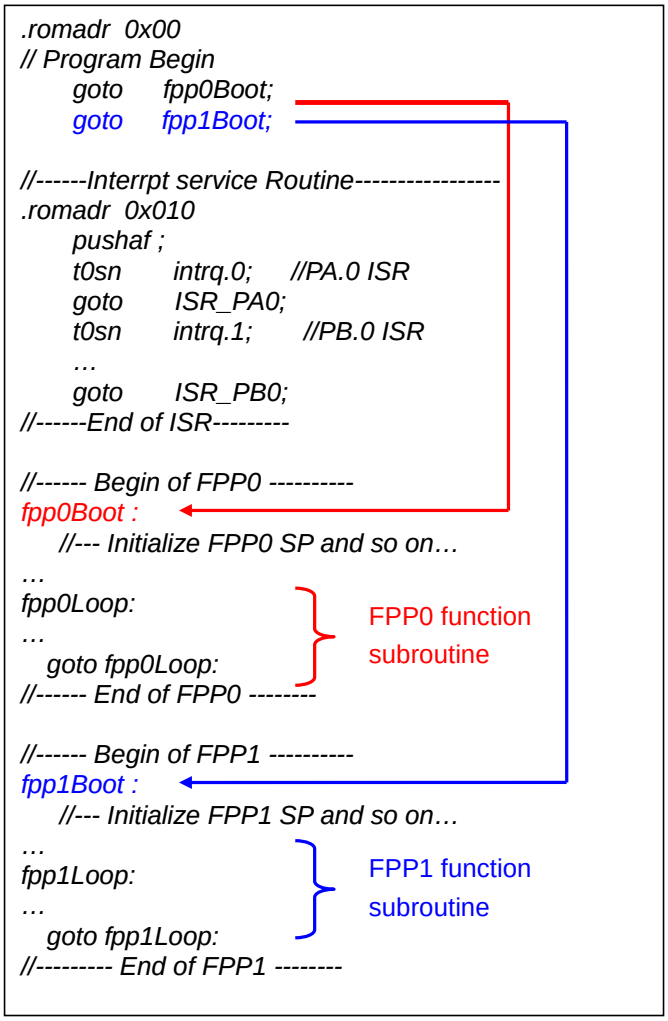


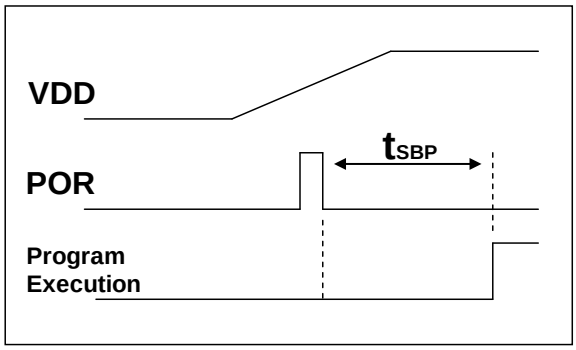
Fig.4: Program Structure

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 PMC232/PMS232 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.

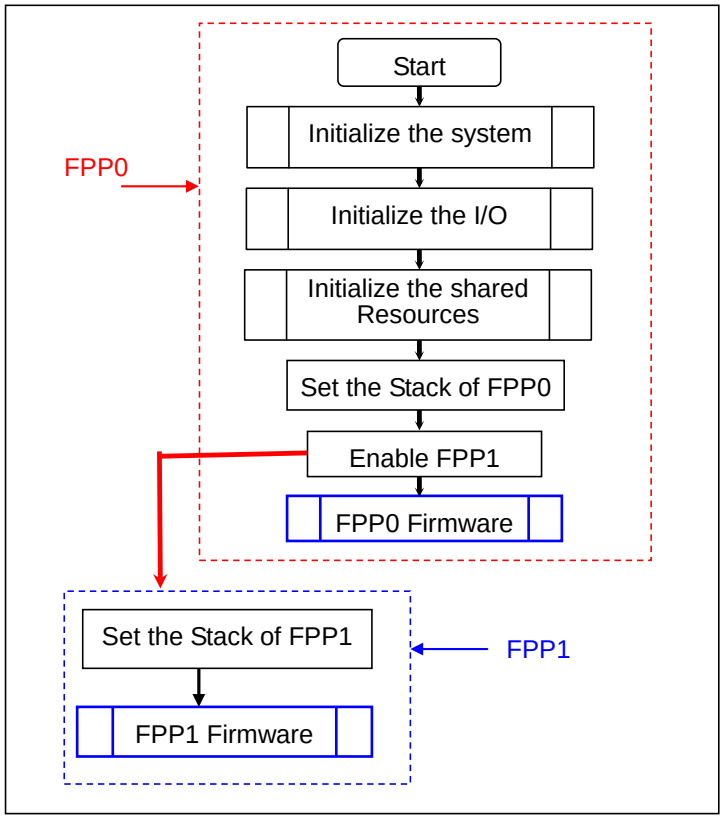


Fig.6: Boot Procedure

5-5. Data Memory -- SRAM

Fig. 7 shows the SRAM data memory organization of PMC232/PMS232, 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 160 bytes data memory of PMC232/PMS232 can be accessed by indirect access mechanism.

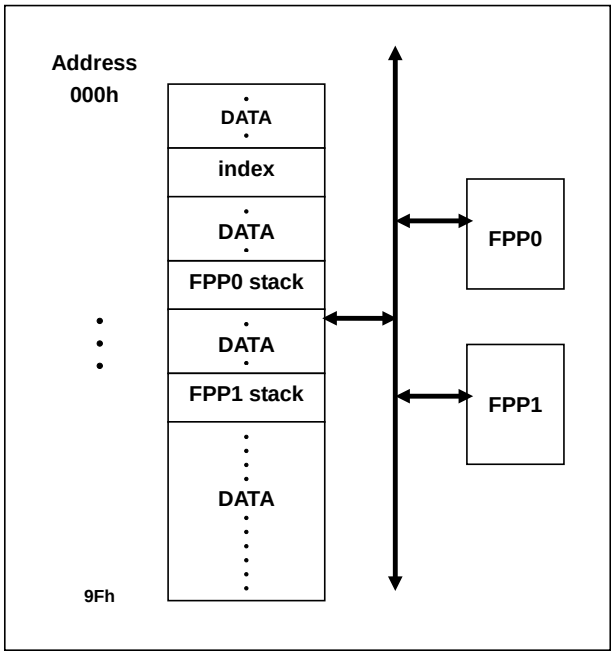


Fig.7: Data Memory Organization

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 PMC232/PMS232: 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
EOSC	**eoscr**.7
IHRC	**clkmd**.4
ILRC	**clkmd**.2

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. Please refer to the measurement chart for IHRC frequency verse VDD and IHRC frequency verse temperature. The frequency of ILRC 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, PMC232/PMS232 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
○ Set IHRC / 2	= 34h (IHRC / 2)	Calibrated	IHRC calibrated to 16MHz, CLK=8MHz (IHRC/2)
○ Set IHRC / 4	= 14h (IHRC / 4)	Calibrated	IHRC calibrated to 16MHz, CLK=4MHz (IHRC/4)
○ Set IHRC / 8	= 3Ch (IHRC / 8)	Calibrated	IHRC calibrated to 16MHz, CLK=2MHz (IHRC/8)
○ Set IHRC / 16	= 1Ch (IHRC / 16)	Calibrated	IHRC calibrated to 16MHz, CLK=1MHz (IHRC/16)
○ Set IHRC / 32	= 7Ch (IHRC / 32)	Calibrated	IHRC calibrated to 16MHz, CLK=0.5MHz (IHRC/32)
○ Set ILRC	= E4h (ILRC / 1)	Calibrated	IHRC calibrated to 16MHz, CLK=ILRC
○ 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 PMC232/PMS232 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 disabled, 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.

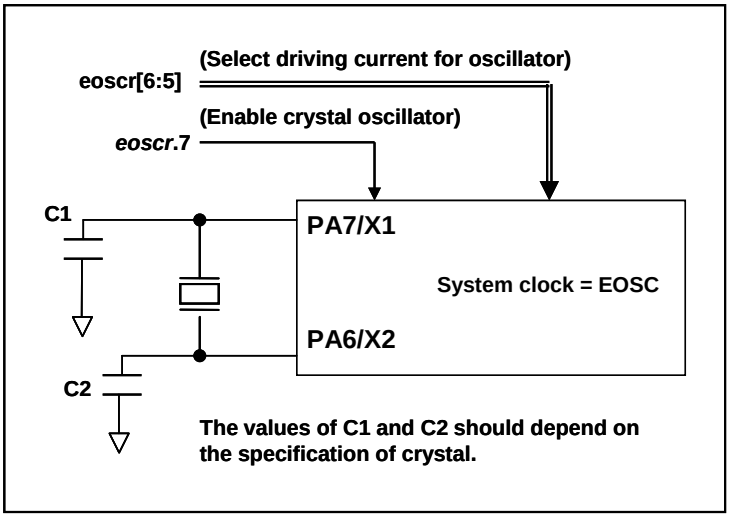


Fig.8: Connection of crystal oscillator

Besides crystal, external capacitor and options of PMC232/PMS232 should be fine tuned in *eoscr* (0x0a) 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	(*eoscr*[6:5]=11, *misc*.6=0)
1MHz	10pF	10pF	11ms	(*eoscr*[6:5]=10, *misc*.6=0)
32kHz	22pF	22pF	450ms	(*eoscr*[6:5]=01, *misc*.6=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 EOSC,
                                        // Intrq.T16 =>1, crystal EOSC Is stable
    WORD    count     =     0;
    stt16count;
    Intrq.T16  =     0;
    wait1      Intrq.T16;               // count from 0x0000 to 0x2000, then set INTRQ.T16
    clkmd      =     0xB4;              // switch system clock to EOSC;
   Clkmd.4 = 0;                         // disable IHRC
    ...
```

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 32 KHz 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 PMC232/PMS232 is shown as Fig. 9.

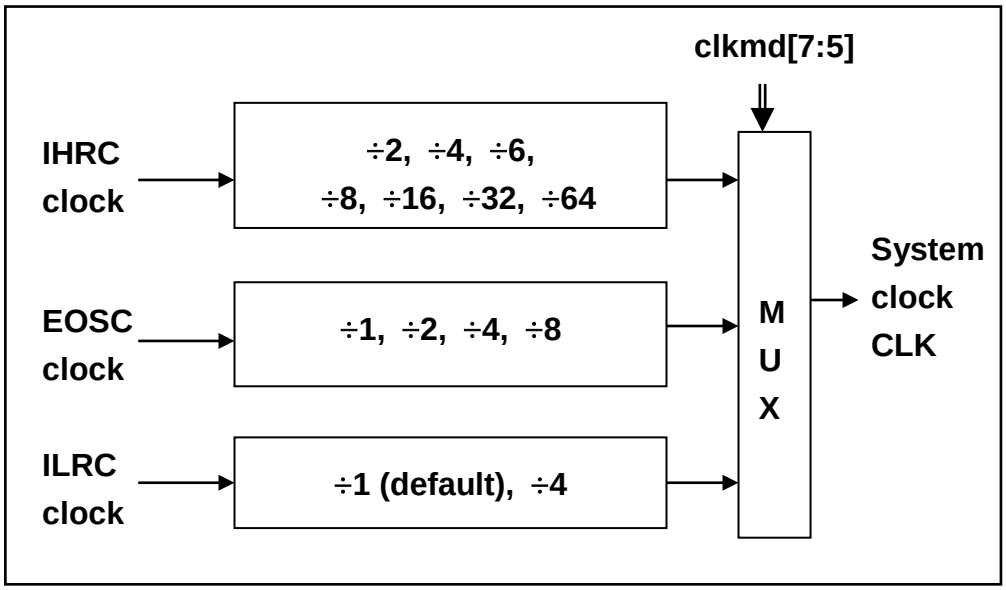


Fig.9: Options of System Clock

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, and the lowest LVR levels can be chosen for different operating frequencies. Please refer to Section 4.1.

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 PMC232/PMS232 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 PMC232/PMS232, 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 5 of ***integs*** register (IO address 0x0C).

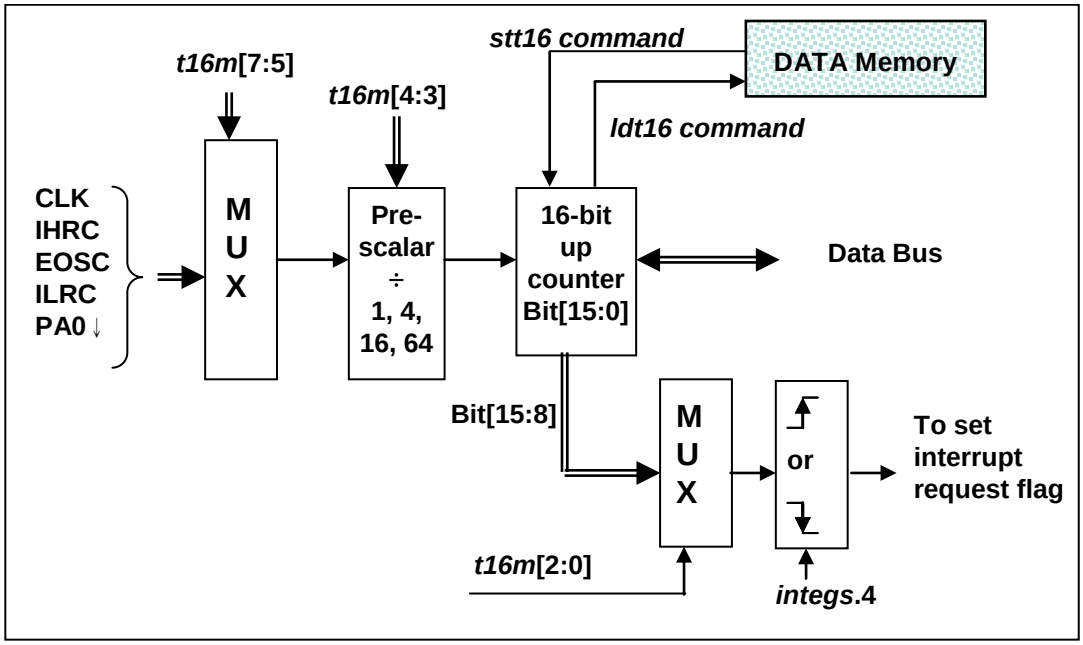


Fig.10: Hardware diagram of Timer16

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-scalar 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 SYSCLK, /64, BIT15;
// choose (SYSCLK/64) as clock source, every 2^16 clock to set INTRQ.2=1
// if using System Clock = IHRC / 2 = 8 MHz
// SYSCLK/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:

$$F_{INTRQ\_T16M} = F_{clock\ source} \div P \div 2^{n+1}$$

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 n^{th} 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 PMC232/PMS232, please refer to Fig. 10 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 and PA4, 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. According to the setting of register tm2c[3:2], Timer2 output can be selectively output to PA2 or PA3. At this point, regardless of whether PA2 or PA3 is the input or output state, Timer2 signal will be forced to output. 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. 11 shows the timing diagram of Timer2 for both period mode and PWM mode.

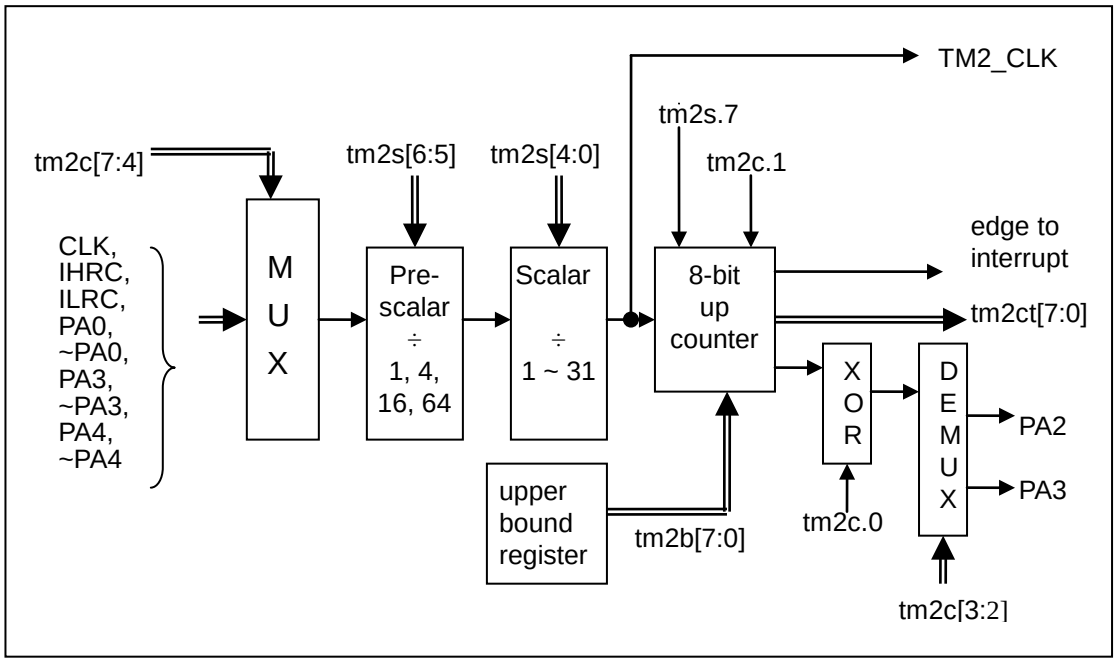
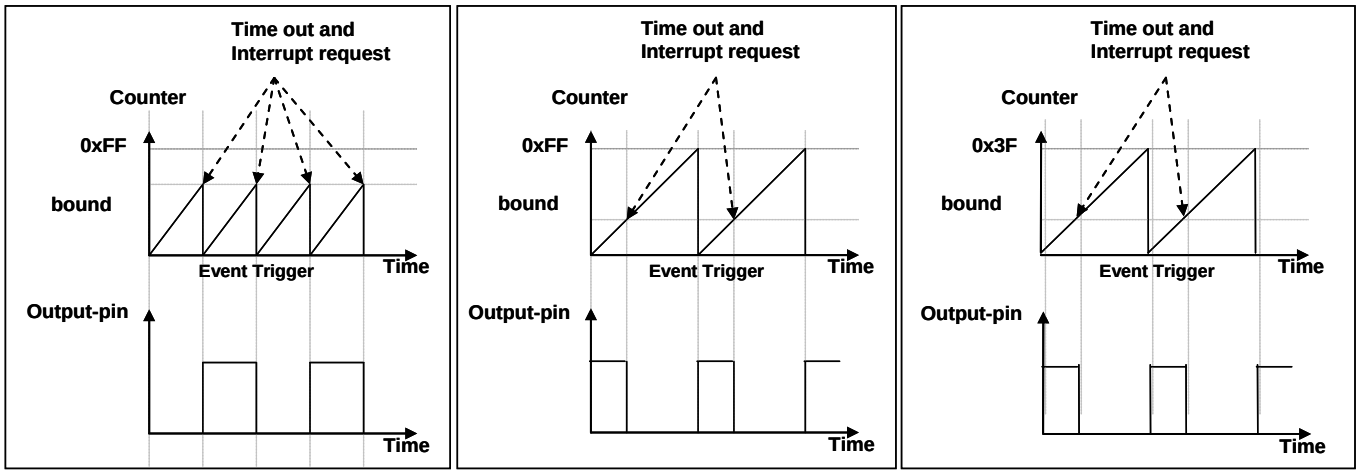


Fig.11: Timer2 hardware diagram



Mode 0 – Period Mode **Mode 1 – 8-bit PWM Mode** **Mode 1 – 6-bit PWM Mode**

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:

Frequency of Output = Y ÷ [2 × (K+1) × S1 × (S2+1)]

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_1000, Y=8MHz

tm2b = 0b0111_1111, K=127

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

➔ frequency of output = 8MHz ÷ [2 × (127＋1) × 1 × (0＋1)] = 31.25kHz

Example 2:

tm2c = 0b0001_1000, Y=8MHz

tm2b = 0b0111_1111, K=127

tm2s[7:0] = 0b0_11_11111, S1=64 , S2 = 31

➔ frequency = 8MHz ÷ (2 × (127＋1) × 64 × (31＋1)) =15.25Hz

Example 3:

tm2c = 0b0001_1000, Y=8MHz

tm2b = 0b0000_1111, K=15

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

➔ frequency = 8MHz ÷ (2 × (15+1) × 1 × (0+1)) = 250kHz

Example 4:

tm2c = 0b0001_1000, Y=8MHz

tm2b = 0b0000_0001, K=1

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

➔ frequency = 8MHz ÷ (2 × (1+1) × 1 × (0+1)) =2MHz

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 = 0x00;
    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:

Frequency of Output = Y ÷ [256 × S1 × (S2+1)]

Duty of Output = [(K＋1) ÷ 256]×100%

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_1010, Y=8MHz
tm2b = 0b0111_1111, K=127
tm2s = 0b0_00_00000, S1=1, S2=0
➔ frequency of output = 8MHz ÷ (256 × 1 × (0+1)) = 31.25kHz
➔ duty of output = [(127+1) ÷ 256] × 100% = 50%

Example 2:

tm2c = 0b0001_1010, Y=8MHz
tm2b = 0b0111_1111, K=127
tm2s = 0b0_11_11111, S1=64, S2=31
➔ frequency of output = 8MHz ÷ (256 × 64 × (31+1)) = 15.25Hz
➔ duty of output = [(127+1) ÷ 256] × 100% = 50%

Example 3:

tm2c = 0b0001_1010, Y=8MHz
tm2b = 0b1111_1111, K=255
tm2s = 0b0_00_00000, S1=1, S2=0
➔ frequency of output = 8MHz ÷ (256 × 1 × (0+1)) = 31.25kHz
➔ duty of output = [(255+1) ÷ 256] × 100% = 100%

Example 4:

tm2c = 0b0001_1010, Y=8MHz
tm2b = 0b0000_1001, K = 9
tm2s = 0b0_00_00000, S1=1, S2=0
➔ frequency of output = 8MHz ÷ (256 × 1 × (0+1)) = 31.25kHz
➔ duty of output = [(9+1) ÷ 256] × 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 = 0x00;
    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:

Frequency of Output = Y ÷ [64 × S1 × (S2+1)]

Duty of Output = [(K＋1) ÷ 64] × 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_1010, 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_1010, 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_1010, 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_1010, 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, PMC232/PMS232 will be reset to restart the program execution. The relative timing diagram of watchdog timer is shown as Fig. 13.

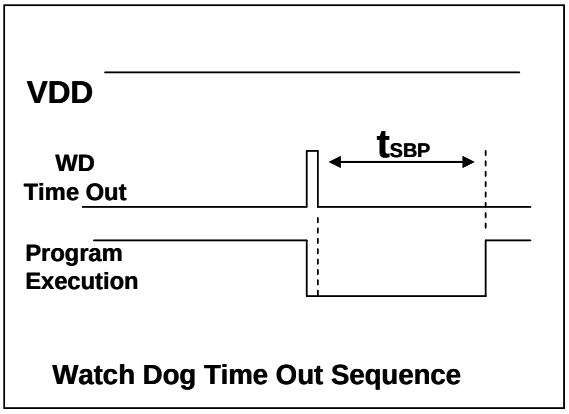


Fig.13: Sequence of Watch Dog Time Out

5-11. Interrupt

There are five interrupt lines for PMC232/PMS232:

- External interrupt PA0
- External interrupt PB0
- ADC interrupt
- Timer16 interrupt
- Timer2 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.

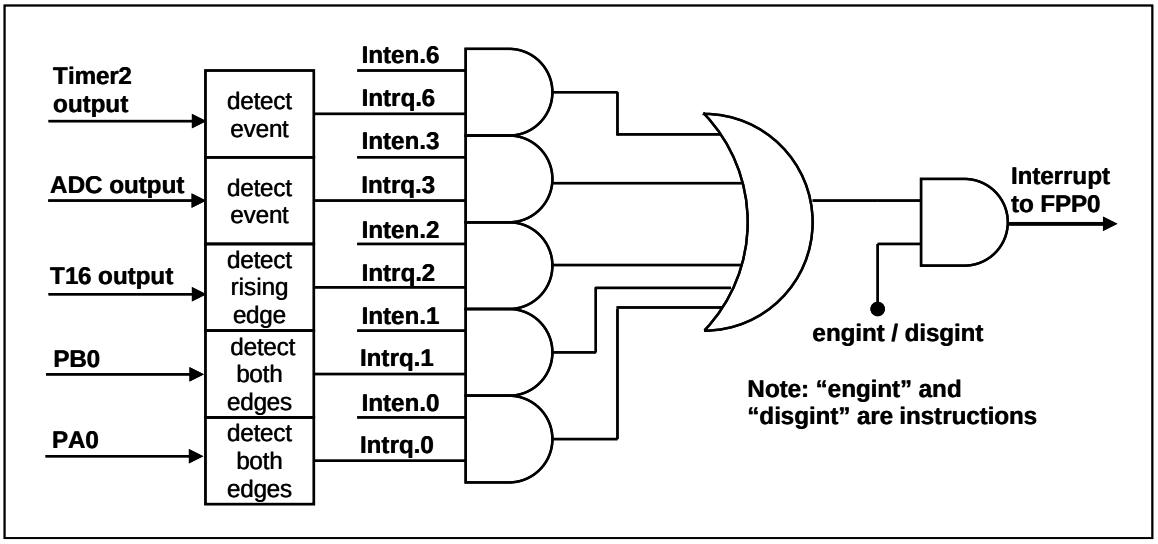


Fig.14: Hardware diagram of interrupt controller

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 0x010.

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

Note: Even if INTEN=0, INTRQ will be still triggered by the interrupt source.

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 INTEN.PA0 will be opened and closed dynamically,
    // user can judge whether INTEN.PA0 =1 or not.
    // Example:   If (INTEN.PA0 && INTRQ.PA0)   {...}

    // If INTEN.PA0 is always enable,
    // user can omit the INTEN.PA0 judgement to speed up interrupt service routine.

    If (INTRQ.PA0)
    {                     // Here for PA0 interrupt service routine
        INTRQ.PA0 = 0;    // Delete corresponding bit (take PA0 for example)
        ...
    }
    ...
    // X : INTRQ = 0;     // It is not recommended to use INTRQ = 0 to clear all at the end of
                          // the interrupt service routine.
                          // It may accidentally clear out the interrupts that have just occurred
                          // and are not yet processed.
    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. Table 6 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

Table 6: 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, the detail information for Power-Save mode shows below:

- IHRC and EOSC oscillator modules: No change, keep active if it was enabled
- ILRC oscillator modules: must remain enabled, need to start with ILRC when be wakening up
- System clock: Disable, therefore, CPU stops execution
- OTP memory is turned off
- T16/ T2: Stop counting if system clock is selected or the corresponding oscillator module is disabled; otherwise, it keeps counting.
- Wake-up sources: IO toggle in digital mode (PxDIER bit is 1) or Timer16 or Timer2.

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;
    ....                            // 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;
          ...
```

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 PMC232/PMS232 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: IO toggle in digital mode (PxDIER bit is 1)

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 PMC232/PMS232 can be resumed to normal operation by toggling IO pins, Timer interrupt is available for Power-Save mode ONLY. Table 7 shows the differences in wake-up sources between STOPSYS and STOPEXE.

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

Table 7: Differences in wake-up sources between Power-Save mode and Power-Down mode

When using the IO pins to wake-up the PMC232/PMS232, registers ***padier*** and ***pbdier*** 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 the 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} : MISC(0x20) / 8 T_{SYS}:MISC(0x28)
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. When wake-up, turning on the watchdog timer **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***), control registers (***pac, pbc, pcc***) and pull-up registers (***paph, pbph, pcph***). 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. 15.

pa.0	*pac.0*	*paph.0*	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 8: PA0 Configuration Table

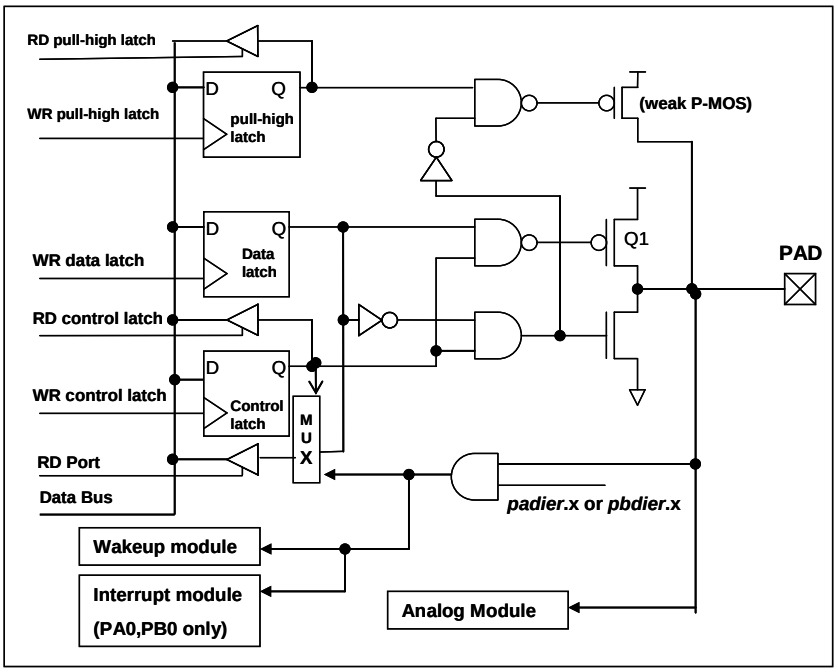


Fig.15: Hardware diagram of IO buffer

Other than PA5, all the IO pins have the same structure; PA5 can is 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 those pins are selected to be analog function. When PMC232/PMS232 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 PMC232/PMS232, once reset is asserted, most of all the registers in PMC232/PMS232 will be set to default values, When reset comes from WDT timeout, ***gdio*** register (IO address 0x7) keeps the same value, system should be restarted once abnormal cases happen, or by jumping program counter to address 'h0. 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 reset

By code option, there are many different levels of LVR for reset. Usually, user selects LVR reset level to be in conjunction with operating frequency and supply voltage.

5-15. VDD/2 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、PC5、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. 16 shows how to use this function.

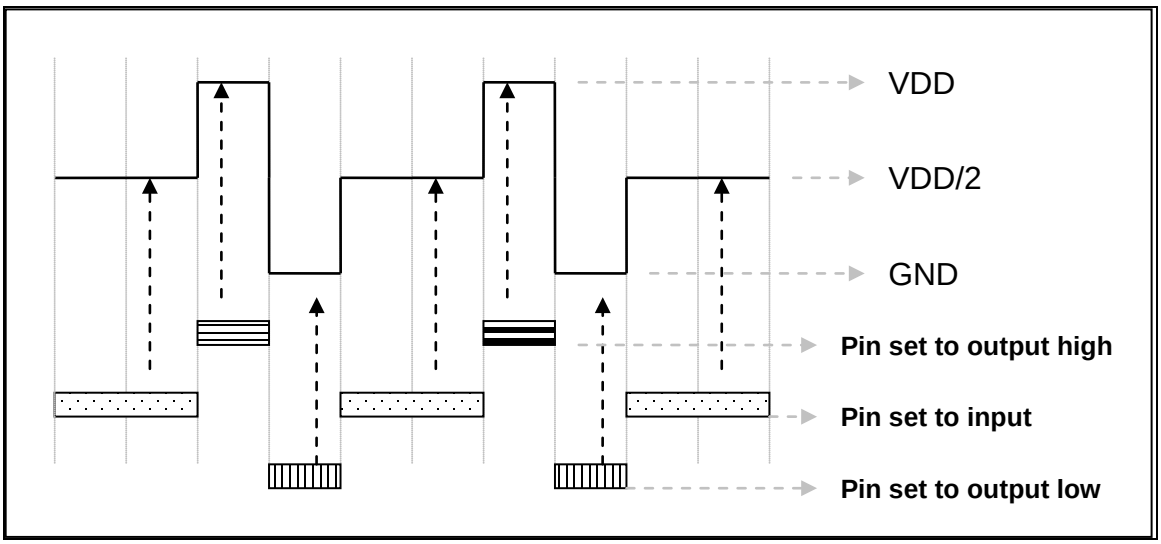


Fig.16: Using VDD/2 bias voltage generator

5-16. Analog-to-Digital Conversion (ADC) module

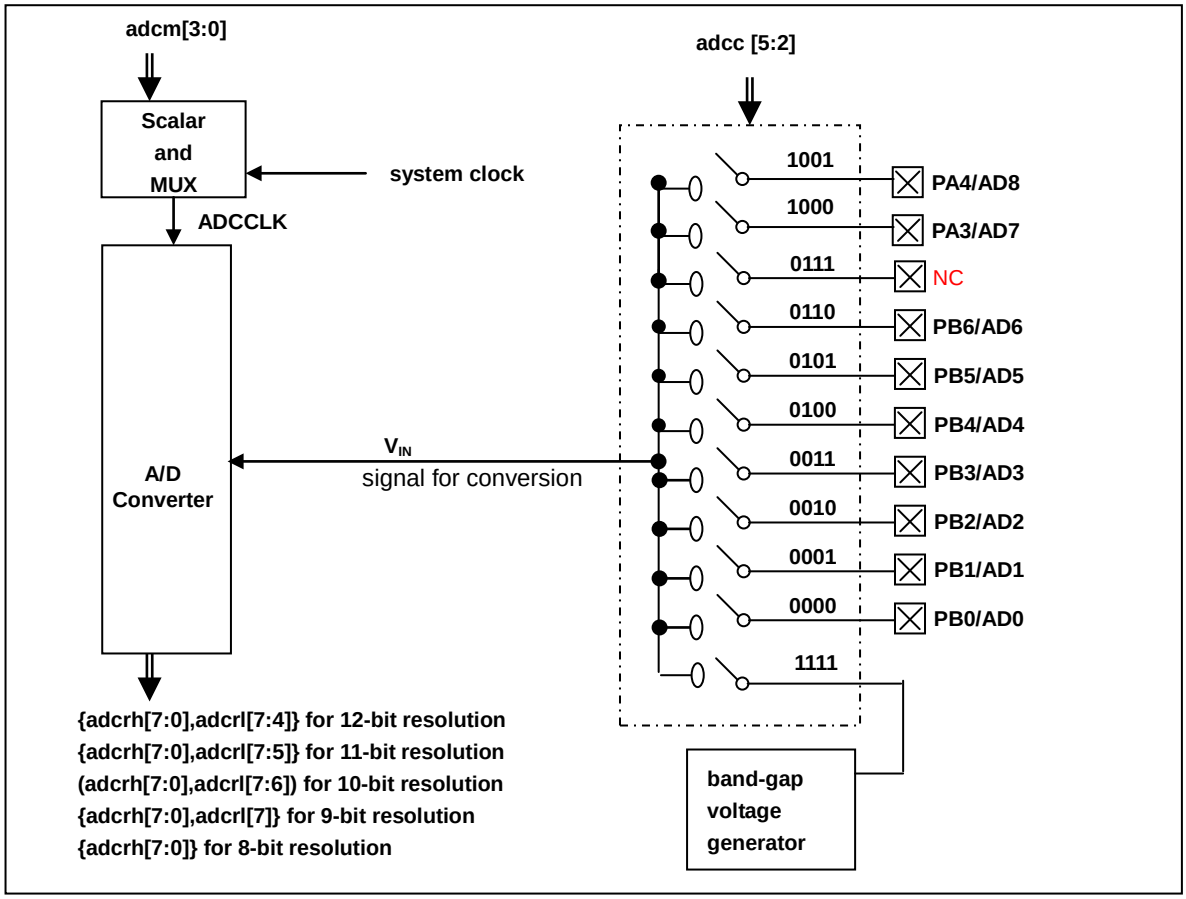


Fig.17: ADC Block Diagram

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***, ***adcrl***)
- Port A/B Digital Input Enable Register (***padier***, ***pbdier***)

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

(1) Configure the ADC module:

- Configure the voltage reference high by ***adcc*** register
- 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***, ***pbdier*** 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* ***addc***.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.

5-16-1. 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. 18, the signal driving source impedance (Rs) and the internal sampling switch impedance (Rss) 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. The recommended maximum impedance for analog driving source is about 10KΩ under 500kHz input frequency and 10-bit resolution requirements, and 10MΩ under 500Hz input frequency and 10-bit resolution.

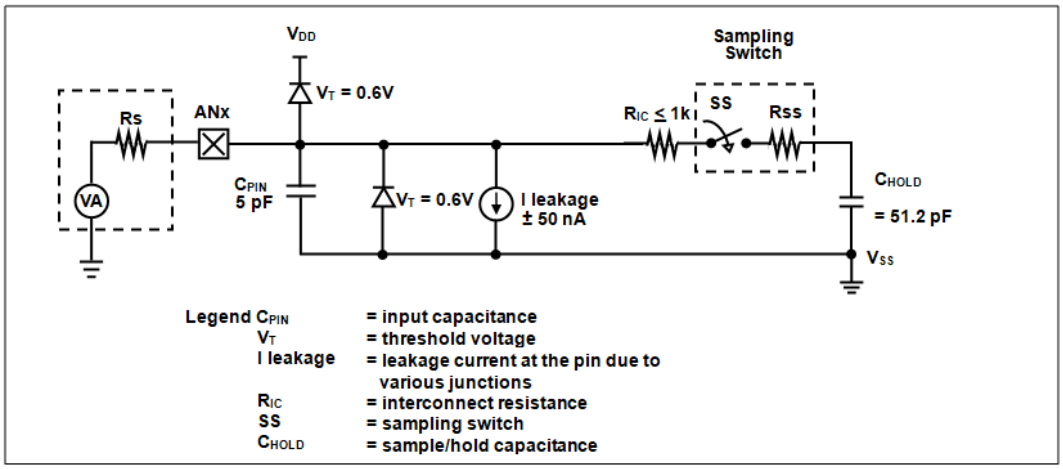


Fig.18: Analog Input Model

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 PMC232/PMS232 series is fixed to one clock period of ADCLK, the selection of ADCLK must be met the minimum signal acquisition time.

5-16-2. Select the ADC bit resolution

The ADC resolution is 12bit. Please configure **adcm** register bit [7:5] to be "100" before staring 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.

5-16-3. 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.

5-16-4. 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}

5-16-5. Configure the analog pins

The 10 analog input signals for ADC shared with Port A[3], Port A[4], and Port B[6:0]. 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-up resistor (3) set the corresponding pin to analog input by port A/B digital input disable register (***padier / pbdier***).

5-16-6. 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-up resistor
PBDIER  =   0B_XXXX_0000;       //  PB0 ~ PB3 digital input is disabled
```

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

```
$   ADCC Enable, PB3;          //  set PB3 as ADC input
$   ADCC Enable, PB2;          //  set PB2 as ADC input
$   ADCC Enable, PB0;          //  set PB0 as ADC input
```

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

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

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
			000: IHRC/4 001: IHRC/2 010: IHRC 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: Reserved 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 : 4	-	-	Reserved.
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 : 4	-	-	Reserved.
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	Timer16 Clock source selection. 000: disable 001: system clock (CLK) 010: reserved 011: reserved 100: IHRC 101: EOSC 110: ILRC 111: PA0 falling edge (from external pin)
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 it will KEEP the old values when reset from watch dog time out. It can perform the IO operation, like *wait0* gdio.x, *wait1* gdio.x and *tog* gdio.x to replace of operations which instructions are supported in memory space (ex: *wait1* mem; *wait0* mem; *tog* mem).

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	-	WO	Bit [7:0] for frequency calibration of IHRC. 0x00 for lowest frequency and 0x7f 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	Reserved
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.

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

Bit	Reset	R/W	Description
7	1	WO	Reserved
6 - 0	0x6F	WO	Enable PB6~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.

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

Bit	Reset	R/W	Description
7 - 0	0x00	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 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.

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

Bit	Reset	R/W	Description
7 - 0	0x00	R/W	Port A pull-up register. This register is used to enable the internal pull-up 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	0x00	R/W	Data register for Port B.

6-18. Port B Control Register (*pbc*), 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-up Register (*pbph*), IO address = 0x16

Bit	Reset	R/W	Description
7 - 0	0x00	R/W	Port B pull-up register. This register is used to enable the internal pull-up 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	0x00	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-up Register (*pcph*), IO address = 0x19

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

6-23. ADC Control Register (*adcc*), IO address = 0x20

Bit	Reset	R/W	Description
7	0	R/W	Enable ADC function. 0/1: Disable/Enable.
6	0	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: reserved 1000: PA3/AD7 1001: PA4/AD8 1111: band-gap 1.20 volt reference voltage Others: reserved
0 - 1	-	-	Reserved. (keep 0 for future compatibility)

6-24. 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] = { ***adcrh***[7:0], ***adcrl***[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

6-25. 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.

6-26. 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

6-27. Miscellaneous Register (*misc*), IO address = 0x3b

Bit	Reset	R/W	Description
7	-	-	Reserved. (keep 0 for future compatibility)
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. The wake-up time is 128(MISC.3=0) / 8(MISC.3=1) 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 will be switched to system clock (for example: 4MHz) when fast wakeup is enabled, therefore, it is recommended to turn off the watchdog timer **before** enabling the fast wakeup and turn on the watchdog timer **after** disabling the fast wakeup.
4	0	WO	Enable to generate half VDD on PA0/PA2/PA3/PC5 pins. 0 / 1 : Disable / Enable
3	0	WO	Recover time from LVR reset. 0: Normal. The system will take about 1024 ILRC clocks to boot up from LVR reset. 1: Fast. The system will take about 64 ILRC clocks to boot up from LVR reset.
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

6-28. 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 : reserved 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) **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.
3–2	00	R/W	Timer2 output selection. 00 : disable 01 : PA2 10 : PA3 11 : reserved
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.

6-29. Timer2 Counter Register (*tm2ct*), IO address = 0x3d

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

6-30. 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.

6-31. Timer2 Bound Register (*tm2b*), IO address = 0x09

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

7. Instructions

Symbol	Description
ACC	Accumulator (Abbreviation of accumulator)
a	Accumulator (symbol of accumulator in program)
sp	Stack pointer
flag	ACC status flag register
I	Immediate data
&	Logical AND
|	Logical OR
←	Movement
^	Exclusive logic OR
+	Add
−	Subtraction
~	NOT (logical complement, 1's complement)
∓	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

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

mov a, IO	Move data from IO into ACC Example: *mov a, pa ;* Result: a ← pa; Flag Z is set when pa is zero. Affected flags: 『Y』Z 『N』C 『N』AC 『N』OV
mov IO, a	Move data from ACC into IO Example: *mov pb, a;* Result: pb ← a Affected flags: 『N』Z 『N』C 『N』AC 『N』OV
nmov M, a	Take the negative logic (2's complement) of ACC to put on memory Example: *mov MEM, a;* Result: MEM ← 〒a Affected flags: 『N』Z 『N』C 『N』AC 『N』OV Application Example: --- *mov a, 0xf5 ; // ACC is 0xf5* *nmov ram9, a; // ram9 is 0x0b, ACC is 0xf5* ---
nmov a, M	Take the negative logic (2's complement) of memory to put on ACC Example: *mov a, MEM ;* Result: a ← 〒MEM; Flag Z is set when 〒MEM is zero. Affected flags: 『Y』Z 『N』C 『N』AC 『N』OV Application Example: --- *mov a, 0xf5 ;* *mov ram9, a ; // ram9 is 0xf5* *nmov a, ram9 ; // ram9 is 0xf5, ACC is 0x0b* ---
ldtabh index	Load high byte data in OTP program memory to ACC by using index as OTP address. It needs 2T to execute this instruction. Example: *ldtabh index;* Result: a ← {bit 15~8 of OTP [index]}; Affected flags: 『N』Z 『N』C 『N』AC 『N』OV Application Example: --- *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 ;* ---

ldtabl index	Load low byte data in OTP to ACC by using index as OTP address. It needs 2T to execute this instruction. Example: *ldtabl index;* Result: a ← {bit7~0 of OTP [index]}; Affected flags: 『N』Z 『N』C 『N』AC 『N』OV Application Example: --- *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 ;* ---
ldt16 word	Move 16-bit counting values in Timer16 to memory in word. Example: *ldt16 word;* Result: word ← 16-bit timer Affected flags: 『N』Z 『N』C 『N』AC 『N』OV Application Example: --- *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* *...* ---
stt16 word	Store 16-bit data from memory in word to Timer16. Example: *stt16 word;* Result: 16-bit timer ←word Affected flags: 『N』Z 『N』C 『N』AC 『N』OV

	Application Example: --- *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* *...* ---
xch M	Exchange data between ACC and memory Example: *xch MEM ;* Result: MEM ← a , a ← MEM Affected flags: 『N』Z 『N』C 『N』AC 『N』OV
idxm a, index	Move data from specified memory to ACC by indirect method. It needs 2T to execute this instruction. Example: *idxm a, index;* Result: a ← [index], where index is declared by word. Affected flags: 『N』Z 『N』C 『N』AC 『N』OV 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)* *...* *idxm a, RAMIndex ; // move data in address 0x5B to ACC* ---
Idxm index, a	Move data from ACC to specified memory by indirect method. It needs 2T to execute this instruction. Example: *idxm index, a;* Result: [index] ← a; where index is declared by word. Affected flags: 『N』Z 『N』C 『N』AC 『N』OV 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

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

addc a, M	Add data in memory with ACC and carry bit, then put result into ACC Example: *addc a, MEM ;* Result: a ← a + MEM + C Affected flags: 『Y』Z 『Y』C 『Y』AC 『Y』OV
addc M, a	Add data in memory with ACC and carry bit, then put result into memory Example: *addc MEM, a ;* Result: MEM ← a + MEM + C Affected flags: 『Y』Z 『Y』C 『Y』AC 『Y』OV
addc a	Add carry with ACC, then put result into ACC Example: *addc a ;* Result: a ← a + C Affected flags: 『Y』Z 『Y』C 『Y』AC 『Y』OV
addc M	Add carry with memory, then put result into memory Example: *addc MEM ;* Result: MEM ← MEM + C Affected flags: 『Y』Z 『Y』C 『Y』AC 『Y』OV
nadd a, M	Add negative logic (2's complement) of ACC with memory Example: *nadd a, MEM ;* Result: a ← ∓a + MEM Affected flags: 『Y』Z 『Y』C 『Y』AC 『Y』OV
nadd M, a	Add negative logic (2's complement) of memory with ACC Example: *nadd MEM, a ;* Result: MEM ← ∓MEM + a Affected flags: 『Y』Z 『Y』C 『Y』AC 『Y』OV
sub a, I	Subtraction immediate data from ACC, then put result into ACC. Example: *sub a, 0x0f;* Result: a ← a - 0fh (a + [2's complement of 0fh]) Affected flags: 『Y』Z 『Y』C 『Y』AC 『Y』OV
sub a, M	Subtraction data in memory from ACC, then put result into ACC Example: *sub a, MEM ;* Result: a ← a - MEM (a + [2's complement of M]) Affected flags: 『Y』Z 『Y』C 『Y』AC 『Y』OV
sub M, a	Subtraction data in ACC from memory, then put result into memory Example: *sub MEM, a;* Result: MEM ← MEM - a (MEM + [2's complement of a]) Affected flags: 『Y』Z 『Y』C 『Y』AC 『Y』OV

subc a, M	Subtraction data in memory and carry from ACC, then put result into ACC Example: *subc a, MEM;* Result: a ← a – MEM - C Affected flags: 『Y』Z 『Y』C 『Y』AC 『Y』OV
subc M, a	Subtraction ACC and carry bit from memory, then put result into memory Example: *subc MEM, a ;* Result: MEM ← MEM – a - C Affected flags: 『Y』Z 『Y』C 『Y』AC 『Y』OV
subc a	Subtraction carry from ACC, then put result into ACC Example: *subc a;* Result: a ← a - C Affected flags: 『Y』Z 『Y』C 『Y』AC 『Y』OV
subc M	Subtraction carry from the content of memory, then put result into memory Example: *subc MEM;* Result: MEM ← MEM - C Affected flags: 『Y』Z 『Y』C 『Y』AC 『Y』OV
inc M	Increment the content of memory Example: *inc MEM ;* Result: MEM ← MEM + 1 Affected flags: 『Y』Z 『Y』C 『Y』AC 『Y』OV
dec M	Decrement the content of memory Example: *dec MEM;* Result: MEM ← MEM - 1 Affected flags: 『Y』Z 『Y』C 『Y』AC 『Y』OV
clear M	Clear the content of memory Example: clear MEM ; Result: MEM ← 0 Affected flags: 『N』Z 『N』C 『N』AC 『N』OV

7-3. Shift Operation Instructions

sr a	Shift right of ACC Example: *sr a ;* Result: a (0,b7,b6,b5,b4,b3,b2,b1) ← a (b7,b6,b5,b4,b3,b2,b1,b0), C ← a(b0) Affected flags: 『N』Z 『Y』C 『N』AC 『N』OV

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

7-4. Logic Operation Instructions

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

xor a, M	Perform logic XOR on ACC and memory, then put result into ACC Example: *xor a, MEM ;* Result: a ← a ^ RAM10 Affected flags: 『Y』Z 『N』C 『N』AC 『N』OV
xor M, a	Perform logic XOR on ACC and memory, then put result into memory Example: *xor MEM, a ;* Result: MEM ← a ^ MEM Affected flags: 『Y』Z 『N』C 『N』AC 『N』OV
not a	Perform 1's complement (logical complement) of ACC Example: *not a ;* Result: a ← ～a Affected flags: 『Y』Z 『N』C 『N』AC 『N』OV Application Example: --- *mov a, 0x38 ; // ACC=0X38* *not a ; // ACC=0XC7* ---
not M	Perform 1's complement (logical complement) of memory Example: *not MEM ;* Result: MEM ← ～MEM Affected flags: 『Y』Z 『N』C 『N』AC 『N』OV Application Example: --- *mov a, 0x38 ;* *mov mem, a ; // mem = 0x38* *not mem ; // mem = 0xC7* ---
neg a	Perform 2's complement of ACC Example: *neg a;* Result: a ← 〒a Affected flags: 『Y』Z 『N』C 『N』AC 『N』OV Application Example: --- *mov a, 0x38 ; // ACC=0X38* *neg a ; // ACC=0XC8* ---
neg M	Perform 2's complement of memory Example: *neg MEM;* Result: MEM ← 〒MEM Affected flags: 『Y』Z 『N』C 『N』AC 『N』OV

	Application Example: --- *mov a, 0x38 ;* *mov mem, a ; // mem = 0x38* *neg mem ; // mem = 0xC8* ---
comp a, I	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: --- *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* ---
comp a, M	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: --- *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* ---
comp M, a	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

7-5. Bit Operation Instructions

set0 IO.n	Set bit n of IO port to low Example: *set0 pa.5 ;* Result: set bit 5 of port A to low Affected flags: 『N』Z 『N』C 『N』AC 『N』OV
set1 IO.n	Set bit n of IO port to high Example: *set1 pb.5 ;* Result: set bit 5 of port B to high Affected flags: 『N』Z 『N』C 『N』AC 『N』OV
tog IO.n	Toggle bit state of bit n of IO port Example: *tog pa.5 ;* Result: toggle bit 5 of port A Affected flags: 『N』Z 『N』C 『N』AC 『N』OV
set0 M.n	Set bit n of memory to low Example: *set0 MEM.5* ; Result: set bit 5 of MEM to low Affected flags: 『N』Z 『N』C 『N』AC 『N』OV
set1 M.n	Set bit n of memory to high Example: *set1 MEM.5 ;* Result: set bit 5 of MEM to high Affected flags: 『N』Z 『N』C 『N』AC 『N』OV
swapc IO.n	Swap the nth bit of IO port with carry bit Example: *swapc IO.0;* Result: C ← IO.0 , IO.0 ← C When IO.0 is a port to output pin, carry C will be sent to IO.0; When IO.0 is a port from input pin, IO.0 will be sent to carry C; Affected flags: 『N』Z 『Y』C 『N』AC 『N』OV Application Example1 (serial output) : --- *...* *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* *...*

	--- 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

ceqsn a, I	Compare ACC with immediate data and skip next instruction if both are equal. Flag will be changed like as (a ← 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
ceqsn a, M	Compare ACC with memory and skip next instruction if both are equal. Flag will be changed like as (a ← a - M) Example: *ceqsn a, MEM;* Result: If a=MEM, skip next instruction Affected flags: 『Y』Z 『Y』C 『Y』AC 『Y』OV
ceqsn M, a	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
cneqsn a, M	Compare ACC with memory and skip next instruction if both are not equal. Flag will be changed like as (a ← a - M) Example: *cneqsn a, MEM;* Result: If a≠MEM, skip next instruction Affected flags: 『Y』Z 『Y』C 『Y』AC 『Y』OV

cneqsn a, I	Compare ACC with immediate data and skip next instruction if both are no equal. Flag will be changed like as (a ← a - I) Example: *cneqsn 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
t0sn IO.n	Check IO bit and skip next instruction if it’s low Example: *t0sn pa.5;* Result: If bit 5 of port A is low, skip next instruction Affected flags: 『N』Z 『N』C 『N』AC 『N』OV
t1sn IO.n	Check IO bit and skip next instruction if it’s high Example: *t1sn pa.5 ;* Result: If bit 5 of port A is high, skip next instruction Affected flags: 『N』Z 『N』C 『N』AC 『N』OV
t0sn M.n	Check memory bit and skip next instruction if it’s low Example: *t0sn MEM.5 ;* Result: If bit 5 of MEM is low, then skip next instruction Affected flags: 『N』Z 『N』C 『N』AC 『N』OV
t1sn M.n	Check memory bit and skip next instruction if it’s high EX: *t1sn MEM.5 ;* Result: If bit 5 of MEM is high, then skip next instruction Affected flags: 『N』Z 『N』C 『N』AC 『N』OV
izsn a	Increment ACC and skip next instruction if ACC is zero Example: *izsn a;* Result: a ← a + 1,skip next instruction if a = 0 Affected flags: 『Y』Z 『Y』C 『Y』AC 『Y』OV
dzsn a	Decrement ACC and skip next instruction if ACC is zero Example: *dzsn a;* Result: A ← A - 1,skip next instruction if a = 0 Affected flags: 『Y』Z 『Y』C 『Y』AC 『Y』OV
izsn M	Increment memory and skip next instruction if memory is zero Example: *izsn MEM;* Result: MEM ← MEM + 1, skip next instruction if MEM= 0 Affected flags: 『Y』Z 『Y』C 『Y』AC 『Y』OV
dzsn M	Decrement memory and skip next instruction if memory is zero Example: *dzsn MEM;* Result: MEM ← MEM - 1, skip next instruction if MEM = 0 Affected flags: 『Y』Z 『Y』C 『Y』AC 『Y』OV

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

7-7. System control Instructions

call label	Function call, address can be full range address space Example: *call function1;* Result: [sp] ← pc + 1 pc ← function1 sp ← sp + 2 Affected flags: 『N』Z 『N』C 『N』AC 『N』OV
goto label	Go to specific address which can be full range address space Example: *goto error;* Result: Go to error and execute program. Affected flags: 『N』Z 『N』C 『N』AC 『N』OV
delay a	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 *delay* instruction is executed, the ACC will be zero. Example: *delay a;* Result: Delay 16 cycles here if ACC=0fh Affected flags: 『N』Z 『N』C 『N』AC 『N』OV 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.
delay I	Delay the (N＋1) cycles which N is specified by the immediate data, the timing is based on the executing FPP unit. After the *delay* instruction is executed, the ACC will be zero. Example: *delay 0x05;* Result: Delay 6 cycles here Affected flags: 『N』Z 『N』C 『N』AC 『N』OV 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.

delay M	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 *delay* instruction is executed, the ACC will be zero. Example: *delay M;* Result: Delay 256 cycles here if M=ffh Affected flags: 『N』Z 『N』C 『N』AC 『N』OV 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.
ret I	Place immediate data to ACC, then return Example: *ret 0x55;* Result: *A ← 55h* *ret ;* Affected flags: 『N』Z 『N』C 『N』AC 『N』OV
ret	Return to program which had function call Example: *ret;* Result: *sp ← sp - 2* *pc ← [sp]* Affected flags: 『N』Z 『N』C 『N』AC 『N』OV
reti	Return to program that is interrupt service routine. After this command is executed, global interrupt is enabled automatically. Example: *reti*; Affected flags: 『N』Z 『N』C 『N』AC 『N』OV
nop	No operation Example: *nop*; Result: nothing changed Affected flags: 『N』Z 『N』C 『N』AC 『N』OV
pcadd a	Next program counter is current program counter plus ACC. Example: *pcadd a;* Result: pc ← pc + a Affected flags: 『N』Z 『N』C 『N』AC 『N』OV Application Example: ---------- *...* *mov a, 0x02 ;* *pcadd a ; // PC <- PC+2* *goto err1 ;* *goto correct ; // jump here* *goto err2 ;* *goto err3 ;* *...* *correct: // jump here* *...* ----------

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

7-8. Summary of Instructions Execution 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, pcadd, ret, reti		2T	2T
Others		1T	1T

7-9. Summary of affected flags by Instructions

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

8. Code Options

Option	Selection	Description
Security	Enable	Security Enable
	Disable	Security Disable
LVR	4.1V	Select LVR = 4.1V
	3.6V	Select LVR = 3.6V
	3.1V	Select LVR = 3.1V
	2.8V	Select LVR = 2.8V
	2.5V	Select LVR = 2.5V
	2.2V	Select LVR = 2.2V
	2.0V	Select LVR = 2.0V
	1.8V	Select LVR = 1.8V
Under_20mS_VDD_OK	Yes	reach normal operating voltage quickly within 20 mS
	No	can’t reach normal operating voltage quickly within 20 mS
FPPA	1-FPPA	Single FPP unit mode
	2-FPPA	Two FPP units mode

9. Special Notes

This chapter is to remind user who use PMC232/PMS232 series IC in order to avoid frequent errors upon operation.

9-1. Warning

User must read all application notes of the IC by detail before using it. Please download the related application notes from the following link:
http://www.padauk.com.tw/tw/technical/index.aspx

9-2. Using IC

9-2-1. IO pin usage and setting

(1) IO pin as digital input

- When IO is set as digital input, the level of Vih and Vil would changes with the voltage and temperature. Please follow the minimum value of Vih and the maximum value of Vil.
- The value of internal pull high resistor would also changes with the voltage, temperature and pin voltage. It is not the fixed value.

(2) IO pin as analog input

- Configure IO pin as input
- Set PADIER and PBDIER registers to configure corresponding IO as analog input
- For some IO pins of PA and PB that are not used, PADIER.1 and PBDIER.7 should be set low in order to prevent current leakage.
- Set PAPH and PBPH registers to disable corresponding IO pull-up resistor
- The functions of PADIER and PBDIER registers of PMC232/PMS232 series IC is contrary to ICE functions

  Please use following program in order to keep ICE emulation consisting with PMC232/PMS232 series IC procedure.

```
$ PADIER 0x0D;
$ PBDIER 0x70;
```

(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

- No internal pull-up resistor for PA5
- Configure PA5 a s 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) PC as digital input pin

- There is no PCDIER register in PMC232/PMS232 series IC, PC is defaulted to enable the digital input function. Therefore, PC can be configured as input to wake up system from power save mode.

(7) 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

Note: Please read the PMC-APN013 carefully. According to PMC-APN013,, the crystal oscillator should be used reasonably. If the following situations happen to cause IC start-up slowly or non-startup, PADAUK Technology is not responsible for this: the quality of the user's crystal oscillator is not good, the usage conditions are unreasonable, the PCB cleaner leakage current, or the PCB layouts are unreasonable.

9-2-2. 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

9-2-3. System clock switching

(1) System 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.

9-2-4. 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
```

9-2-5. TIMER time out

When select $ INTEGS BIT_R (default value) and T16M counter BIT8 to generate interrupt, if T16M counts from 0, 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.

If select $ INTEGS BIT_F(BIT triggers from 1 to 0) and T16M counter BIT8 to generate interrupt, the T16M counter changes to an interrupt every 0x200/0x400/0x600/. Please pay attention to two differences with setting INTEGS methods.

9-2-6. 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 10KΩ.

(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.

9-2-7. 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.

9-2-8. IHRC

(1) The IHRC frequency calibration is performed when IC is programmed by the writer.

(2) Because the characteristic of the Epoxy Molding Compound (EMC) would some degrees affects the IHRC frequency (either for package or COB), if the calibration is done before molding process, the actual IHRC frequency after molding may be deviated or becomes out of spec. Normally , the frequency is getting slower a bit.

(3) It usually happens in COB package or Quick Turnover Programming (QTP). And PADAUK would not take any responsibility for this situation.

(4) Users can make some compensatory adjustments according to their own experiences. For example, users can set IHRC frequency to be 0.5% ~ 1% higher and aim to get better re-targeting after molding.

9-2-9. Program writing

There are 8 pins for using the writer to program: PA0，PA3，PA4，PA5，PA6，PA7， VDD, and GND.

User can program PMC232/PMS232 from using PDK3S-P-002:

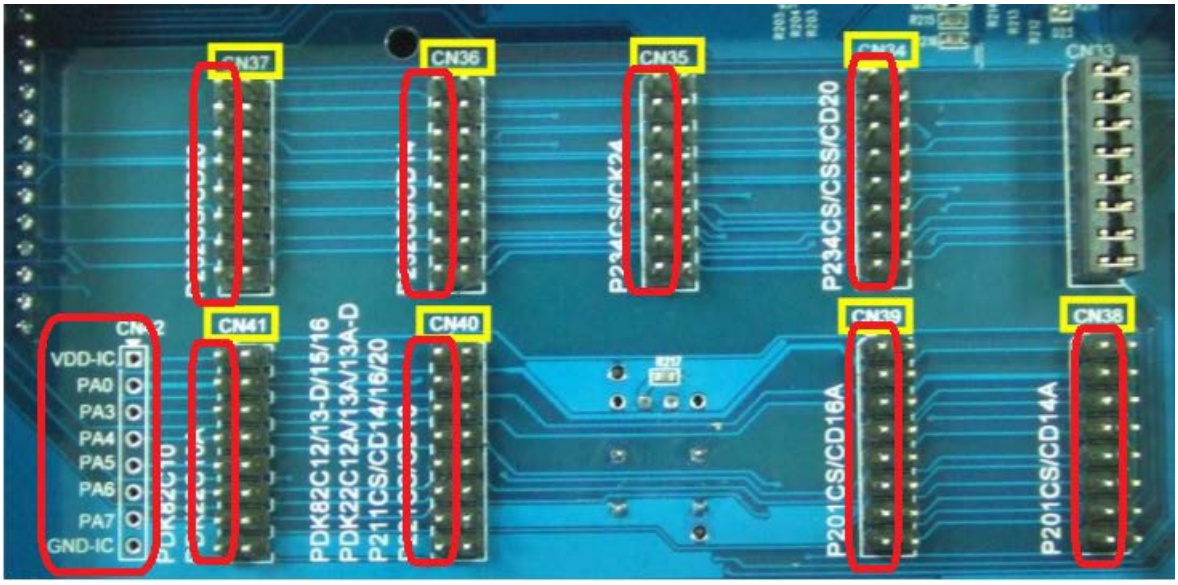
(1) Put the PMC232/PMS232-S14/D14 in the top of the textool over the CN36.

(2) Put the PMC232/PMS232-S20/D20 in the top of the textool over the CN37.

(3) Put the PMC232/PMS232-S18/D18 to move down one space over CN37.

(4) Other packages could be programmed by user's way.

All the left signs behind the jumper are the same (there are VDD, PA0, PA3, PA4, PA5, PA6, PA7, and GND).The following picture is shown:



If user use PDK5S-P-003 or above to program, please follow the instruction.

- Special notes about voltage and current while Multi-Chip-Package(MCP) or On-Board Programming

(1) PA5 (VPP) may be higher than 11V.

(2) VDD may be higher than 6.5V, and its maximum current may reach about 20mA.

(3) All other signal pins level (except GND) are the same as VDD.

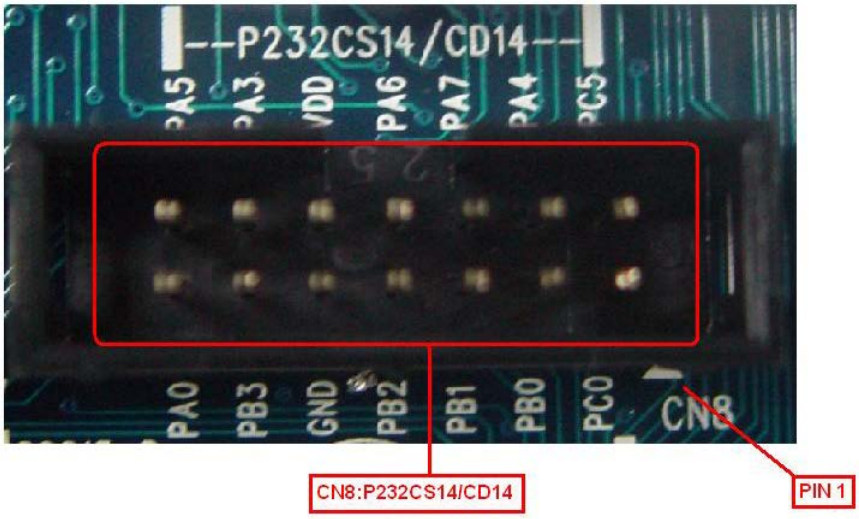
User should confirm when using this product in MCP or On-Board Programming, the peripheral circuit or components will not be destroyed or limit the above voltages.

9-3. Using ICE

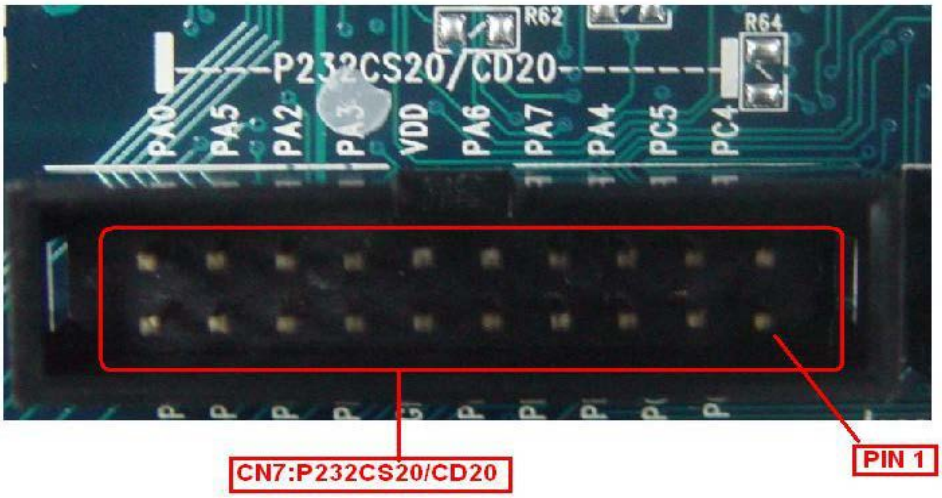
9-3-1. Emulating PMC232/PMS232 series IC on ICE PDK3S-I-001/002/003

PMC232/PMS232 series I/O pin is defined as compatible with P232C series. When user use ICE PDK3S-I-001/002/003 to emulate PMC232/PMS232 series IC, please connect the cable labeled CN8 or CN7 with CN8 or CN7 connector on ICE PDK3S-I-001/002/003 by matching each assembled pins respectively.

(1) Emulating PMC232/PMS232(SOP14/DIP14) : Use cable labeled CN8:P232CS14/CD14 to connect CN8 connector on ICE PDK3S-I-001. Connection is shown as below:



(2) Emulating PMC232/PMS232(SOP20/DIP20) : Use cable labeled CN7:P232CS20/CD20 to connect CN7 connector on ICE PDK3S-I-001/002/003. Connection is shown as below:



9-3-2. Important Notice for ICE operation

(1) When ICE PDK3S-I-001/002/003 emulates LCD 1/2 VDD function of PMC232/PMS232 series IC, the PC5 as COM1 for 1/2 VDD function of PMC232/PMS232 series IC requires jumping wire for adding 5.1K pull-up and down resistance.

(2) ICE PDK3S-I-001/002/003 does not support emulating below IC functions of PMC232/PMS232 series from (a) to (g)**.** Thus, user needs to take PMC232/PMS232 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. PMC232/PMS232 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) PMC232/PMS232 series IC is able to set LVR minimum to 1.8V, total 8 stages 4.0V, 3.5V, 3.0V, 2.75V, 2.5V, 2.2V, 2.0V, 1.8V.

(b) PMC232/PMS232 series IC LVR function can be disabled by register (misc.2).

(c) PMC232/PMS232 series IC is able to support LVR reset and fast-recover by register (misc.3).

(d) PMC232/PMS232 series IC is able to be set as single core operating model.

(e) PMC232/PMS232 series IC supports VDD less than 4V, 3V, 2V voltage level detection and store detecting result to internal register (rstst).

(f) PMC232/PMS232 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]