PMC156/PMS156 Series
8-bit IO-Type Controller
Data Sheet

Version 0.05 – July 07, 2016

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

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

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 that may involve potential risks of death, personal injury, fire or severe property damage.

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

Table of Contents

Revision History:

Revision	Date	Description
0.01	2013/12/10	1st version
0.02	2014/2/10	Add section 5.10.3 Notice for LVR reset ,Add chapter 8 Special Notes
0.03	2014/12/22	Amend PMS156 operating temperature to -40°C ~ 85°C
0.04	2015/6/16	Amend PMS156 operating temperature to -20°C ~ 70°C
0.05	2016/7/7	1. Add section 5.8.3: the description of wake-up 2. Add section 8.3 Warning

1. Features

1.1. Special Features

- PMC156 series:
  - High EFT series
  - Operating temperature range: -40°C ~ 85°C
- PMS156 series:
  - General purpose series
  - Please don't apply to AC RC step-down powered, high power ripple or high EFT requirement application
  - Operating temperature range: -20°C ~ 70°C

1.2. System Features

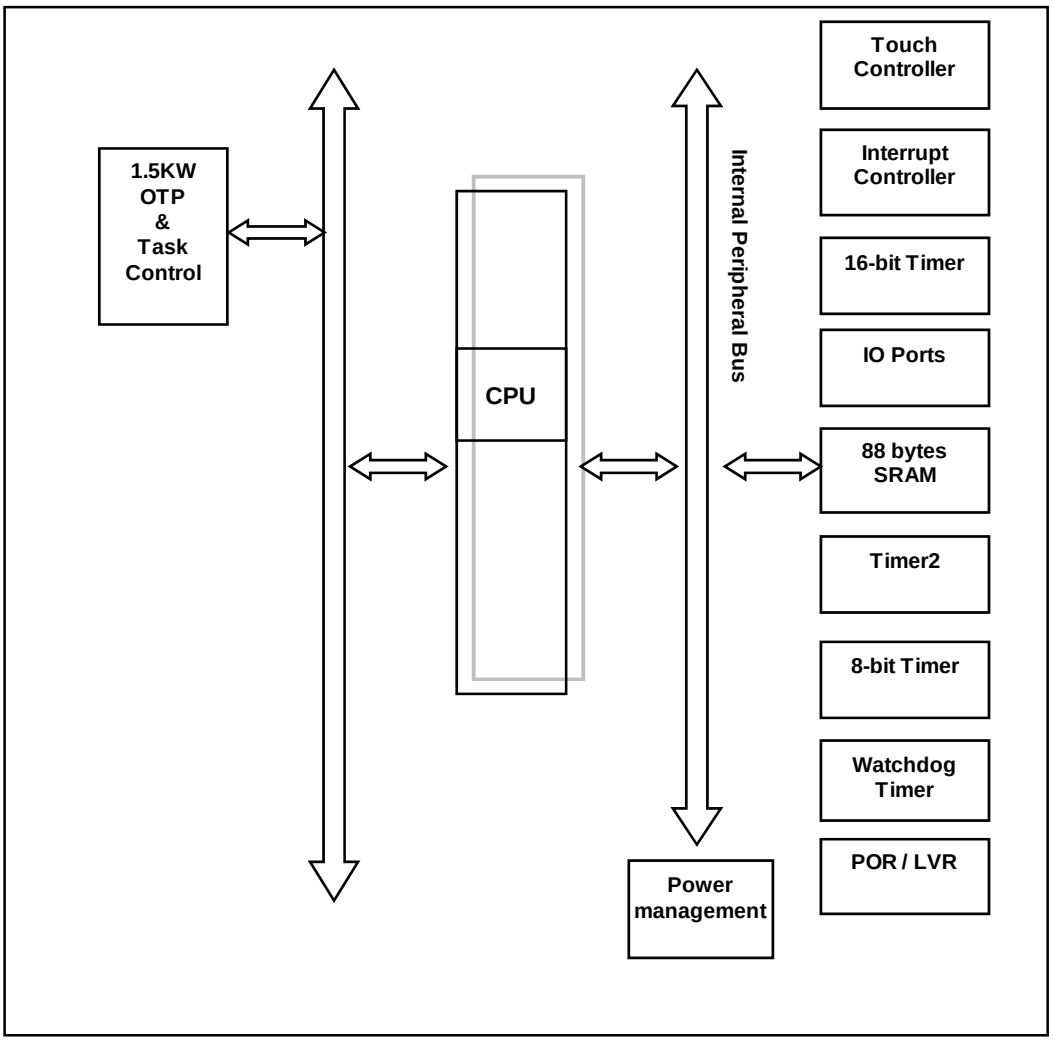
- Clock sources: External crystal oscillator, internal high RC oscillator and internal low RC oscillator
- Built-in Internal High RC Oscillator (IHRC)
- Band-gap circuit to provide 1.20V reference voltage
- One hardware 16-bit timer
- Built-in half VDD bias voltage generator for LCD application
- Support fast wake-up
- Eight levels of LVR reset : 4.1V, 3.6V, 3.1V, 2.8V, 2.5V, 2.2V, 2.0V, 1.8V
- 16 IO pins with 10mA capability and optional pull-high resistor
- Two external interrupt pins
- Every IO pin can be configured to enable wake-up function
- Operating frequency range:
  DC ~ 8MHz@VDD≧3.3V; DC ~ 4MHz@VDD≧2.5V; DC ~ 2MHz@VDD≧2.2V
- Operating voltage range: 2.2V ~ 5.5V
- Low power consumption
- $I_{operating}$ ~ 1.7mA@1MIPS, VDD=5.0V $I_{operating}$ ~ 8uA@ILRC=21kHz, VDD=3.3V
  $I_{powerdown}$ ~ 1uA@VDD=5.0V $I_{powerdown}$ ~ 0.5uA@VDD=3.3V
- Package Information: SOP14/SOP16/SOP18/SOP20/DIP18

1.3. CPU Features

- One processing unit operating mode
- 1KW OTP program memory
- 64 Bytes data RAM
- 79 Powerful instructions
- Most instructions are 1T execution cycle
- Programmable stack pointer and adjustable stack level
- All data memories are available for use as an index pointer
- Separated IO space and memory space

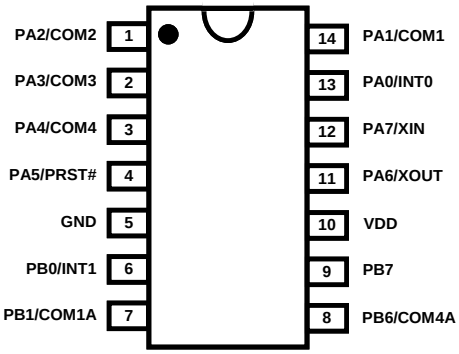
2. General Description and Block Diagram

The PMC156/PMS156 is an IO-Type, OTP-based controller. The PMC156/PMS156 employs RISC architecture and most the instructions are executed in one cycle except that few instructions are two cycles that handle indirect memory access. 1KW bits OTP program memory and 64 bytes data SRAM are inside, one hardware 16-bit timer is also provided in the PMC156/PMS156.

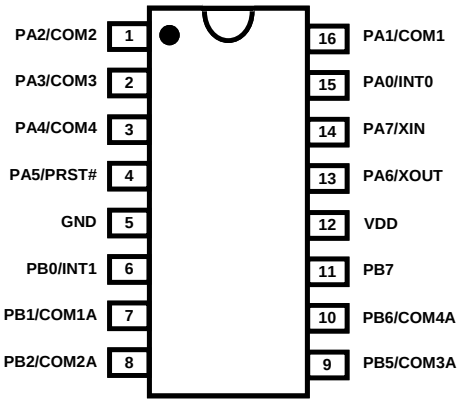


3. Pin Assignment and Functional Description

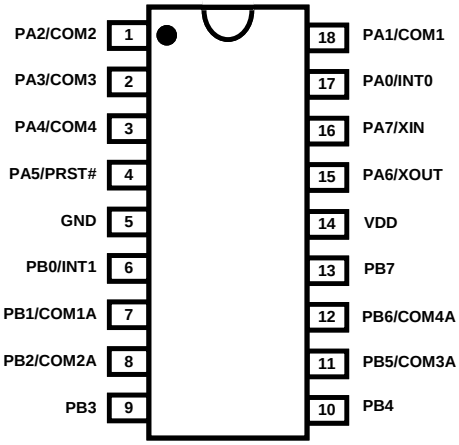
PMC156 series



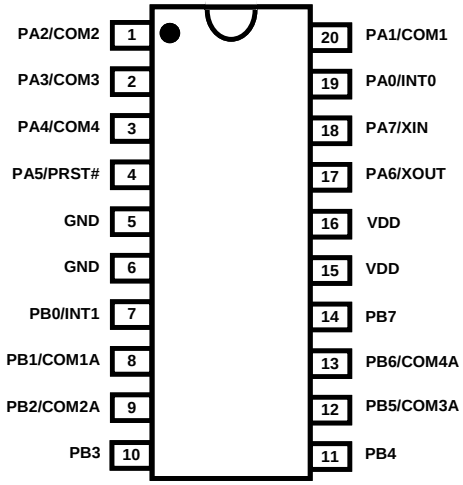
PMC156 (SOP14-150mil)



PMC156 (SOP16-150mil)

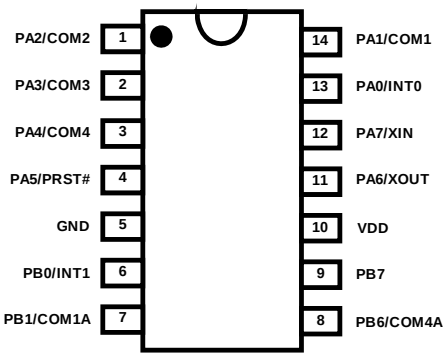


PMC156 (SOP18-300mil)
PMC156 (DIP18-300mil)

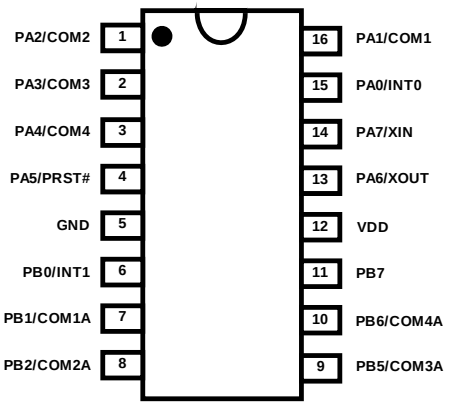


PMC156 (SOP20-300mil)

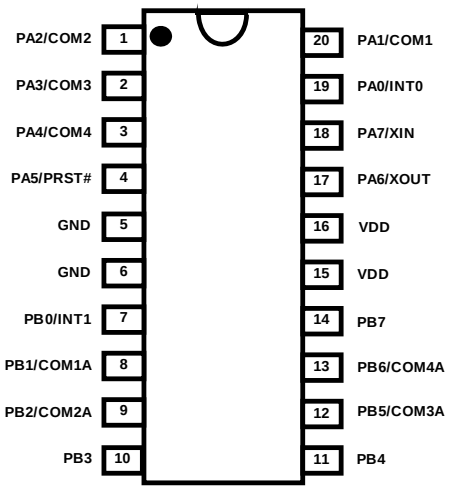
PMS156 series



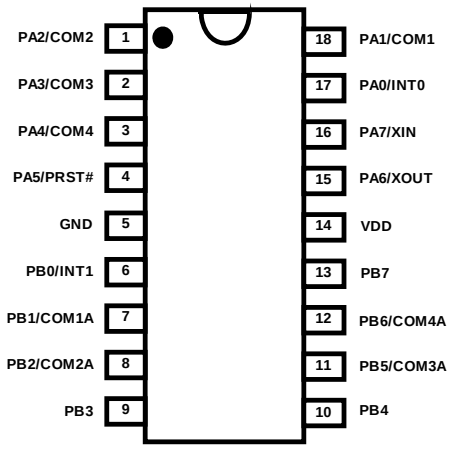
PMS156 (SOP14-150mil)



PMS156 (SOP16-150mil)



PMS156 (SOP20-300mil)



PMS156 (SOP18-300mil)

PMS156 (DIP18-300mil)

Pin Name	Pin & Buffer Type	Description
PA7/ X1	IO ST / CMOS / Analog	The functions of this pin can be: (1) Bit 7 of port A. It can be configured as input or output with pull-up resistor. (2) X1 (input) 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 / Analog	The functions of this pin can be: (1) Bit 6 of port A. It can be configured as input or output with pull-up resistor. (2) X2 (output) 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/ PRST#	IO ST / CMOS	The functions of this pin can be: (1) Bit 5 of port A. It can be configured as 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 / COM4	IO ST / CMOS / Analog	The functions of this pin can be: (1) Bit 4 of port A. It can be configured as input or output with pull-up resistor. (2) COM4 of group 1 for LCD to provide (1/2 VDD) for LCD bias voltage. (3) Channel 0 input. This pin can be used to wake up system during sleep mode; however, wake-up function from this pin is also disabled when bit 4 of ***padier*** register is "0"
PA3 / COM3	IO ST / CMOS / Analog	The functions of this pin can be: (1) Bit 3 of port A. It can be configured as input or output with pull-up resistor. (2) COM3 of group 1 for LCD to provide (1/2 VDD) for LCD bias voltage. This pin can be used to wake up system during sleep mode; however, wake-up function from this pin is also disabled when bit 3 of ***padier*** register is "0"
PA2 / COM2	IO ST / CMOS / Analog	The functions of this pin can be: (1) Bit 2 of port A. It can be configured as input or output with pull-up resistor. (2) COM2 of group 1 for LCD to provide (1/2 VDD) for LCD bias voltage. This pin can be used to wake up system during sleep mode; however, wake-up function from this pin is also disabled when bit 2 of ***padier*** register is "0"
PA1 / COM1	IO ST / CMOS / Analog	The functions of this pin can be: (1) Bit 1 of port A. It can be configured as input or output with pull-up resistor. (2) COM1 of group 1 for LCD to provide (1/2 VDD) for LCD bias voltage. This pin can be used to wake up system during sleep mode; however, wake-up function from this pin is also disabled when bit 1 of ***padier*** register is "0"

Pin Name	Pin Type & Buffer Type	Description
PA0 / INT0	IO ST / CMOS / Analog	The functions of this pin can be: (1) Bit 0 of port A. It can be configured as input or output with pull-up resistor. (2) External interrupt line 0. Both rising edge and falling edge are accepted to request interrupt service. This pin can be used to wake up system during sleep mode; however, wake-up function from this pin is also disabled when bit 0 of ***padier*** register is "0".
PB7	IO ST / CMOS / Analog	The functions of this pin can be: (1) Bit 7 of port B. It can be configured as input or output with pull-up resistor. This pin can be used to wake up system during sleep mode; however, wake-up function from this pin is also disabled when bit 7 of ***pbdier*** register is "0".
PB6 / COM4A	IO ST / CMOS	The functions of this pin can be: (1) Bit 6 of port B. It can be configured as input or output with pull-up resistor. (2) COM4A of group 2 for LCD to provide (1/2 VDD) for LCD bias voltage.. This pin can be used to wake up system during sleep mode; however, wake-up function from this pin is also disabled when bit 6 of ***pbdier*** register is "0".
PB5 / COM3A	IO ST / CMOS	The functions of this pin can be: (1) Bit 5 of port B. It can be configured as input or output with pull-up resistor. (2) COM3A of group 2 for LCD to provide (1/2 VDD) for LCD bias voltage.. This pin can be used to wake up system during sleep mode; however, wake-up function from this pin is also disabled when bit 5 of ***pbdier*** register is "0".
PB4	IO ST / CMOS	The functions of this pin can be: (1) Bit 4 of port B. It can be configured as input or output with pull-up resistor. This pin can be used to wake up system during sleep mode; however, wake-up function from this pin is also disabled when bit 4 of ***pbdier*** register is "0".
PB3	IO ST / CMOS	The functions of this pin can be bit 4 of port B. It can be configured as input or output with pull-up resistor. This pin can be used to wake up system during sleep mode; however, wake-up function from this pin is also disabled when bit 3 of ***pbdier*** register is "0".
PB2 / COM2A	IO ST / CMOS	The functions of this pin can be: (1) Bit 2 of port B. It can be configured as input or output with pull-up resistor. (2) COM2A of group 2 for LCD to provide (1/2 VDD) for LCD bias voltage.. This pin can be used to wake up system during sleep mode; however, wake-up function from this pin is also disabled when bit 2 of ***pbdier*** register is "0".

Pin Name	Pin Type & Buffer Type	Description
PB1 / COM1A	IO ST / CMOS	The functions of this pin can be: (1) Bit 1 of port B. It can be configured as input or output with pull-up resistor. (2) COM1A of group 2 for LCD to provide (1/2 VDD) for LCD bias voltage.. This pin can be used to wake up system during sleep mode; however, wake-up function from this pin is also disabled when bit 1 of ***pbdier*** register is "0".
PB0/INT1	IO ST / CMOS	The functions of this pin can be: (1) Bit 0 of port B. It can be configured as input or output with pull-up resistor. (2) External interrupt line 1. Both rising edge and falling edge are accepted to request interrupt service. This pin can be used to wake up system during sleep mode; however, wake-up function from this pin is also disabled when bit 0 of ***pbdier*** register is "0".
VDD		Positive power
GND		Ground
Notes: IO: Input/Output; ST: Schmitt Trigger input; Analog: Analog input pin; CMOS: CMOS voltage level		

4. Device Characteristics

4.1. DC/AC Characteristics

All data are acquired under the conditions of VDD=5.0V, f_{SYS}=2MHz unless noted.

Symbol	Description	Min	Typ	Max	Unit	Conditions
V_{DD}	Operating Voltage	2.2	5.0	5.5	V	f_{SYS} = 2MHz
f_{SYS}	System clock* = IHRC/2 IHRC/4 IHRC/8 ILRC	 0 0 0	 37K	 8M 4M 2M	Hz	Under_20ms_Vdd_ok**= Y/N VDD≧2.5V / V_{DD}≧3.1V VDD≧2.2V / V_{DD}≧2.5V VDD≧2.2V / V_{DD}≧2.2V VDD = 5.0V
I_{OP}	Operating Current		1.7 8		mA uA	f_{SYS}=1MIPS@5.0V f_{SYS}=ILRC=21kHz@3.3V
I_{PD}	Power Down Current (by ***stopsy***s command)		1 0.5		uA uA	f_{SYS}= 0Hz,VDD=5.0V f_{SYS}= 0Hz,VDD=3.3V
I_{PS}	Power Save Current (by ***stopexe*** command)		0.4		mA	VDD=5.0V; Band-gap, LVR, IHRC, ILRC, Timer16 modules are ON.
V_{IL}	Input low voltage for IO lines	0		0.3 VDD	V	
V_{IH}	Input high voltage for IO lines	0.7 VDD		VDD	V	
I_{OL}	IO lines sink current	7	10	13	mA	VDD =5.0V, V_{OL}=0.5V
I_{OH}	IO lines drive current	-5	-7	-9	mA	VDD =5.0V, V_{OH}=4.5V
$I_{INJ\ (PIN)}$	Injected current on pin			1	mA	VDD+0.3≧V_{IN}≧ -0.3
R_{PH}	Pull-high Resistance		62 100 210		KΩ	VDD =5.0V VDD =3.3V VDD =2.2V
V_{LVR}	Low Voltage Reset 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	
f_{IHRC}	Frequency of IHRC after calibration *	15.84* 15.20* 15.28*	16* 16* 16*	16.16* 16.80* 16.72	MHz	@25°C VDD=2.2V~5.5V, -40°C <Ta<85°C* -20°C <Ta<70°C*

Symbol	Description	Min	Typ	Max	Unit	Conditions
f_{ILRC}	Frequency of ILRC *	31.3*	37*	41.9*	KHz	VDD=5.0V, Ta=25°C
		24.0* 25.9*	37* 37*	50.0* 48.1*		VDD=5.0V, -40°C <Ta<85°C* VDD=5.0V, -20°C <Ta<70°C*
		18.3*	21*	24.5*		VDD=3.3V, Ta=25°C
		14.0* 14.7*	21* 21*	29.0* 27.3*		VDD=3.3V, -40°C <Ta<85°C* VDD=3.3V, -20°C <Ta<70°C*
t_{INT}	Interrupt pulse width	30			ns	V_{DD} = 5.0V
V_{DR}	RAM data retention voltage*	1.5			V	In power-down mode.
t_{WDT}	Watchdog timeout period		2048		ILRC clock period	misc[1:0]=00 (default)
			4096			misc[1:0]=01
			16384			misc[1:0]=10
t_{SBP}	System boot-up period from power-on		29 48		ms	@VDD=5V, ILRC~37kHz @VDD=3.3V, ILRC~21kHz
t_{WUP}	System wake-up period					
	Fast wake-up by IO toggle from STOPEXE suspend		128		T_{SYS}	Where T_{SYS} is the time period of system clock
	Fast wake-up by IO toggle from STOPSYS suspend, IHRC is the system clock		128 T_{SYS} + T_{SIHRC}			Where T_{SIHRC} is the stable time of IHRC from power-on.
	Normal wake-up from STOPEXE or STOPSYS suspend		1024		T_{ILRC}	Where T_{ILRC} is the clock period of ILRC
t_{RST}	External reset pulse width	120			us	@VDD=5V

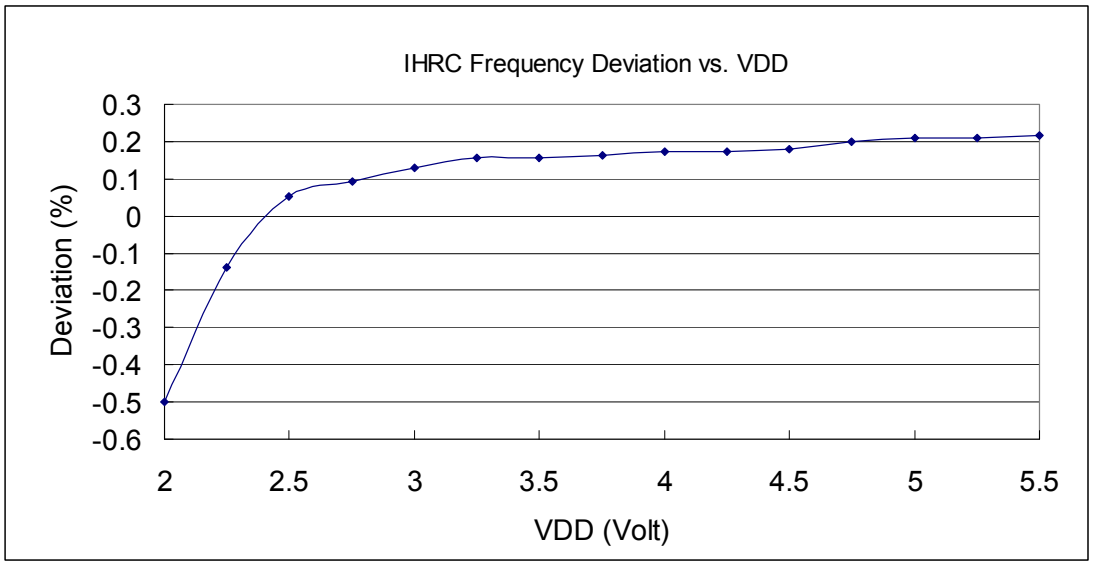
*These parameters are for design reference, not tested for every chip.

** Under_20ms_Vdd_Ok is a checking condition for the VDD rising from 0V to the stated voltage within 20ms.

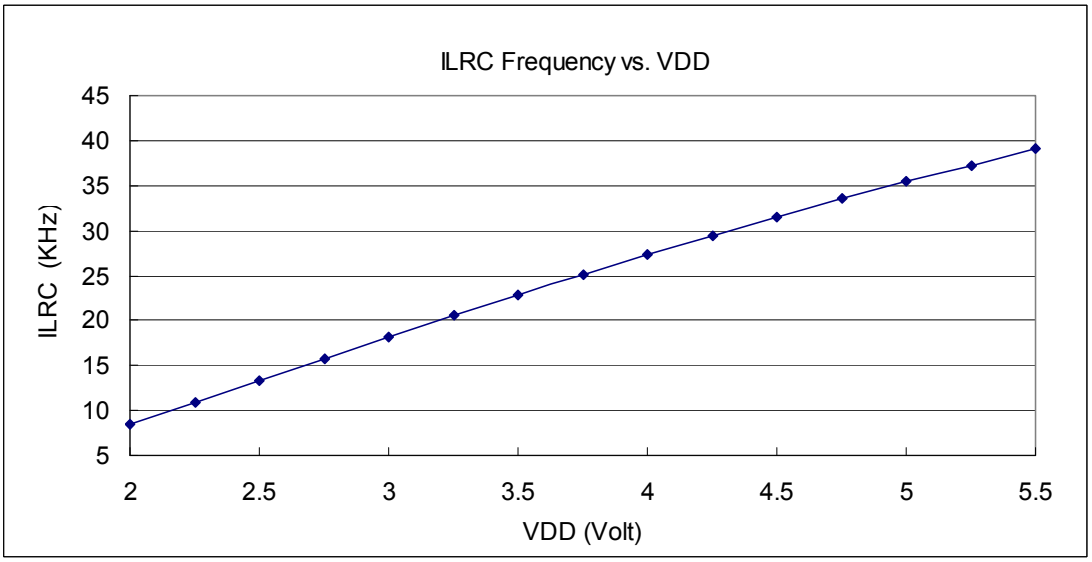
4.2. Absolute Maximum Ratings

- Supply Voltage ……..... 2.2V ~ 5.5V
- Input Voltage …….. -0.3V ~ VDD + 0.3V
- Operating Temperature …… PMC156 series:-40°C ~ 85°C ;
  PMS156 series:-20°C ~ 70°C
- Storage Temperature …… -50°C ~ 125°C
- Junction Temperature …… 150°C

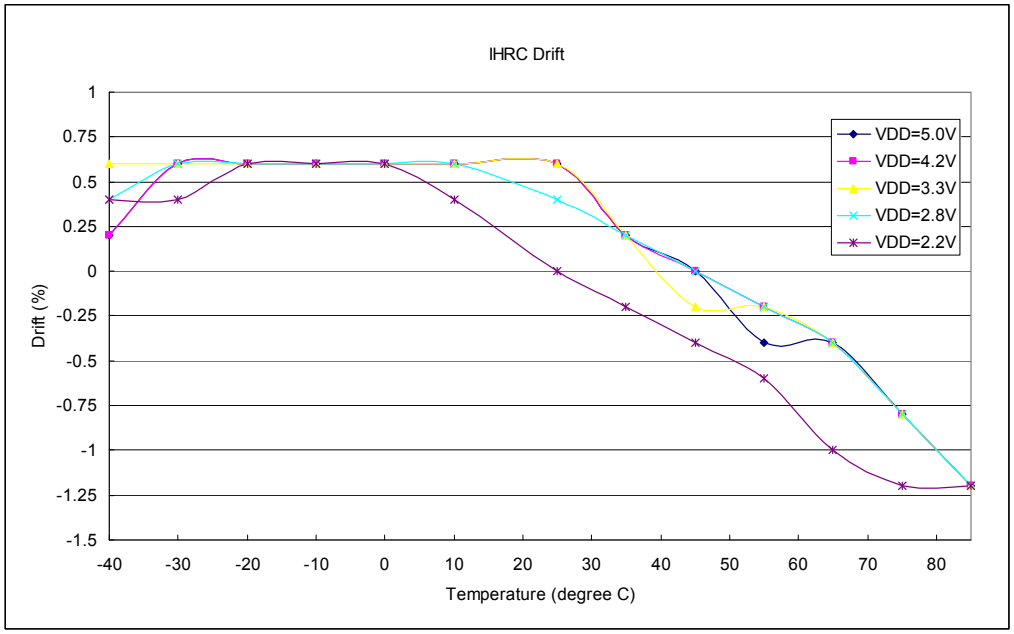
4.3. Typical IHRC Frequency vs. VDD (calibrated to 16MHz)



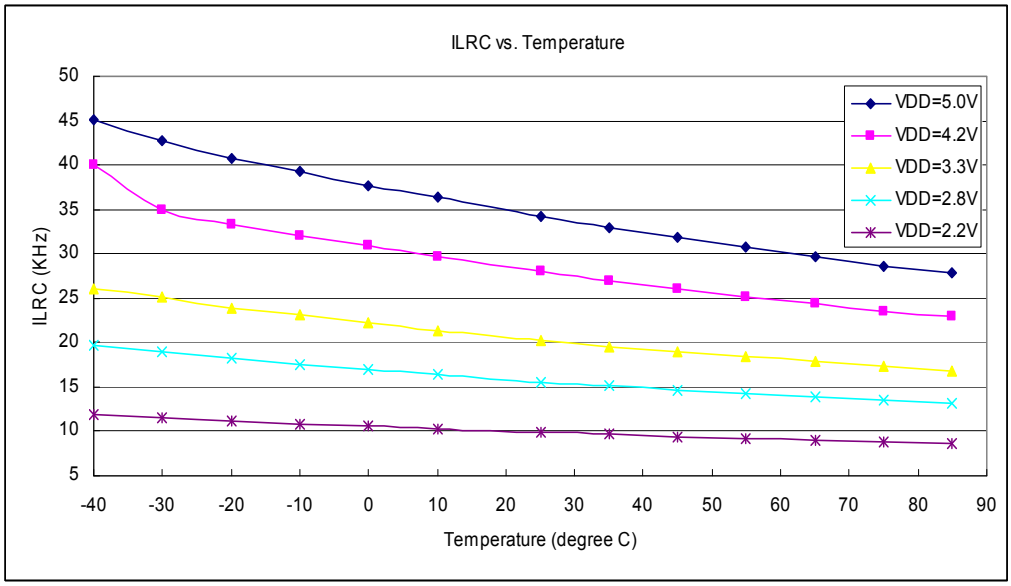
4.4. Typical ILRC Frequency vs. VDD



4.5. Typical IHRC Frequency vs. Temperature (calibrated to 16MHz)



4.6. Typical ILRC Frequency vs. Temperature

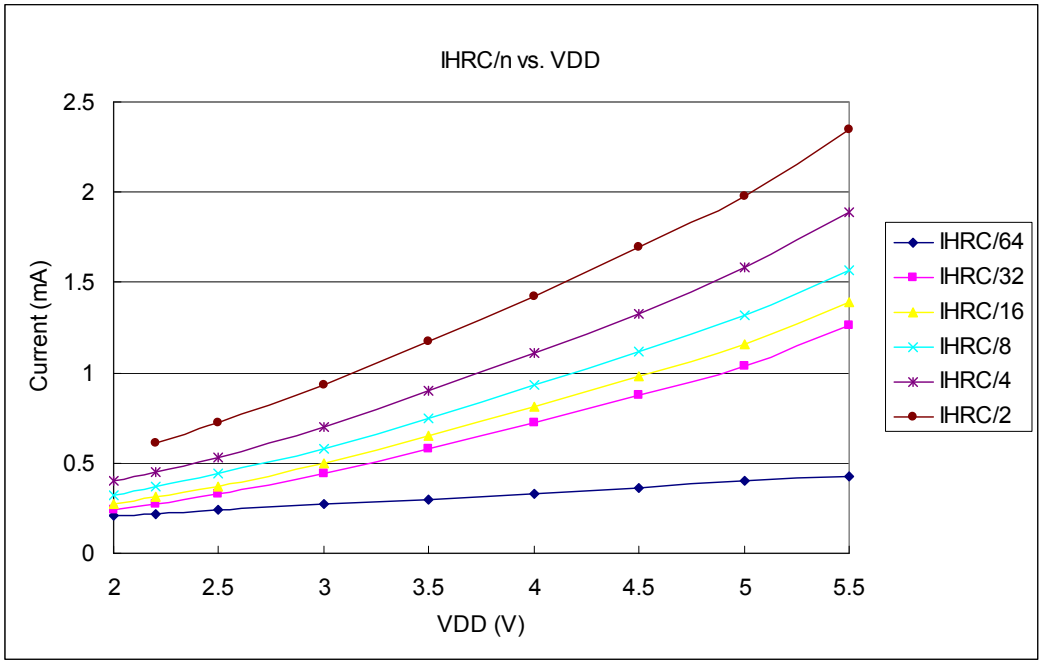


4.7. Typical Operating Current vs. VDD and CLK=IHRC/n

- Conditions:

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

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

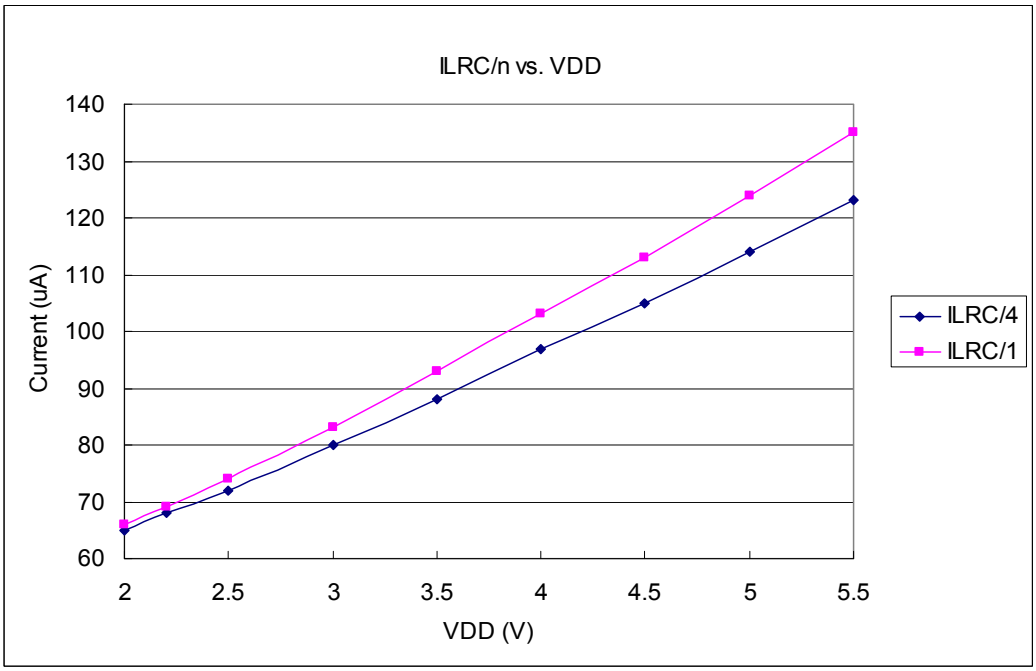


4.8. Typical Operating Current vs. VDD and CLK=ILRC/n

- Conditions:

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

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

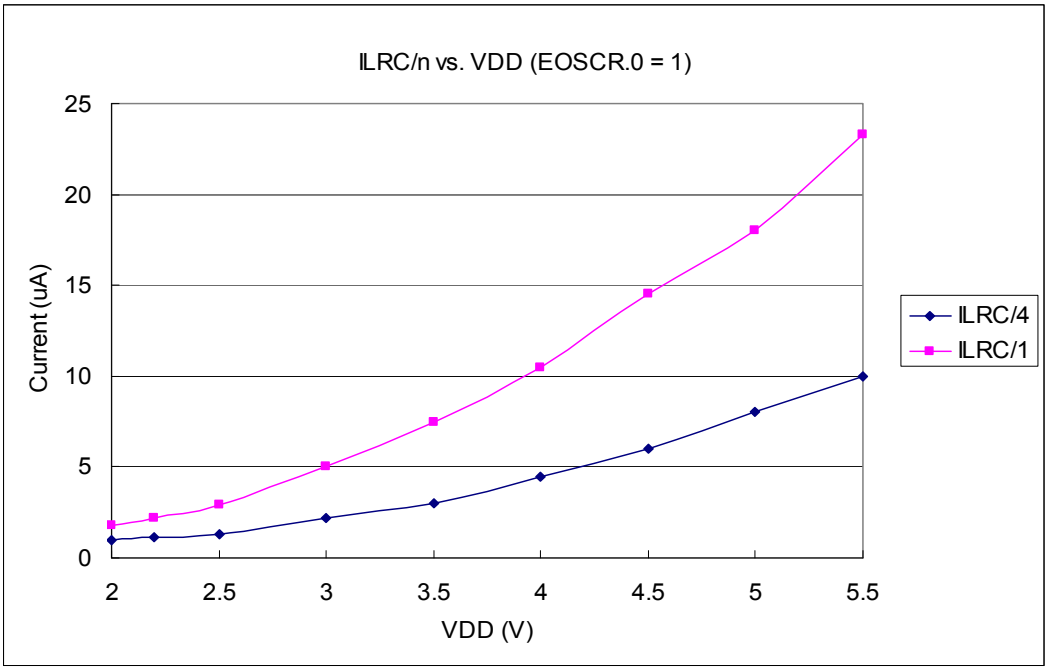


4.9. Typical Lowest Operating Current vs. VDD and CLK=ILRC/n

- Conditions:

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

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



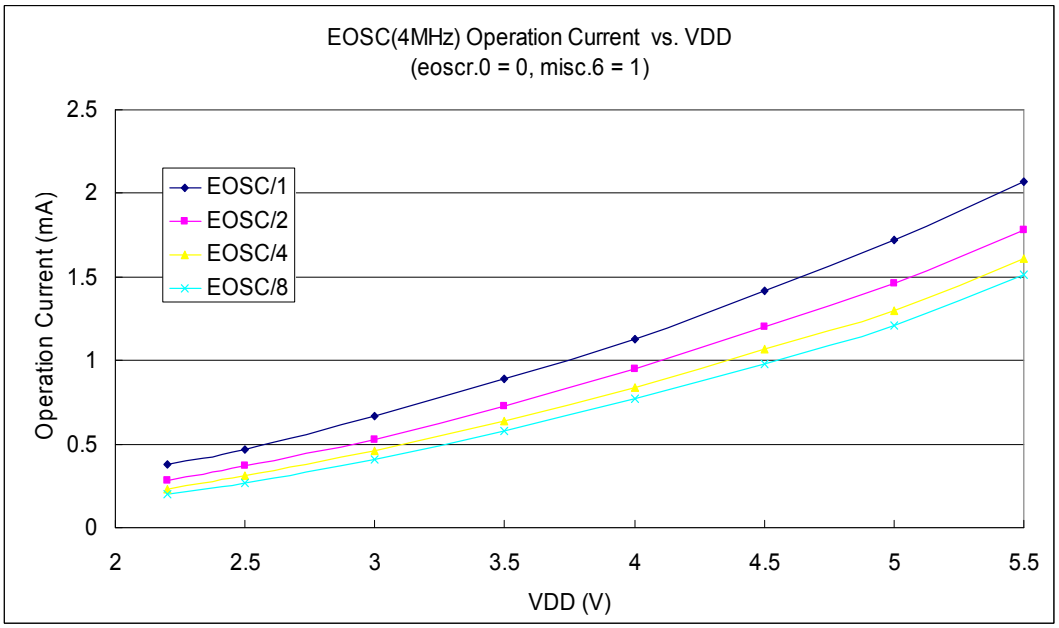
4.10. Typical operating current vs. VDD @ system clock = 4MHz EOSC / n

- Conditions:

  ON: EOSC, MISC.6 = 1;

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

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



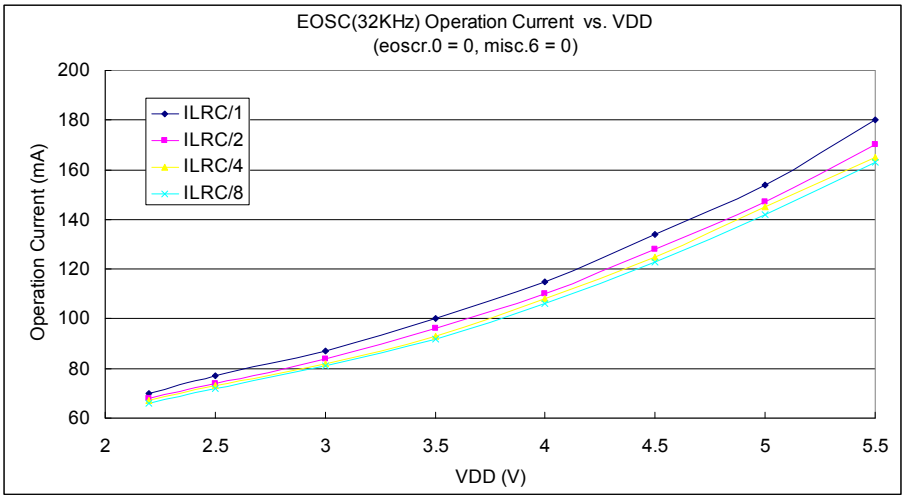
4.11. Typical operating current vs. VDD @ system clock = 32kHz EOSC / n

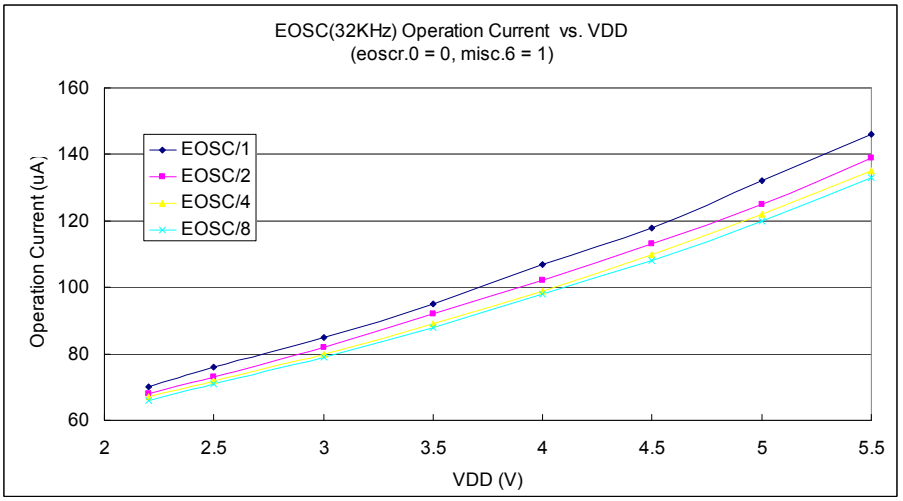
- Conditions:

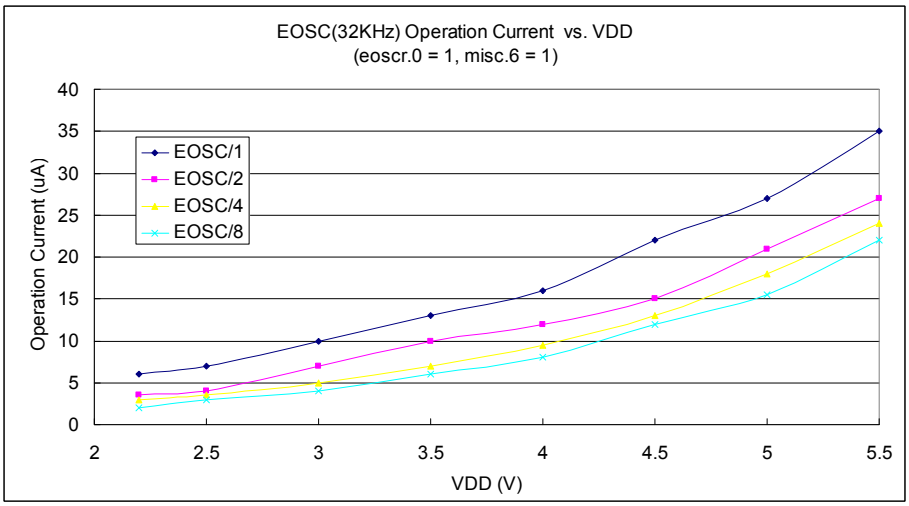
  ON: EOSC;

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

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







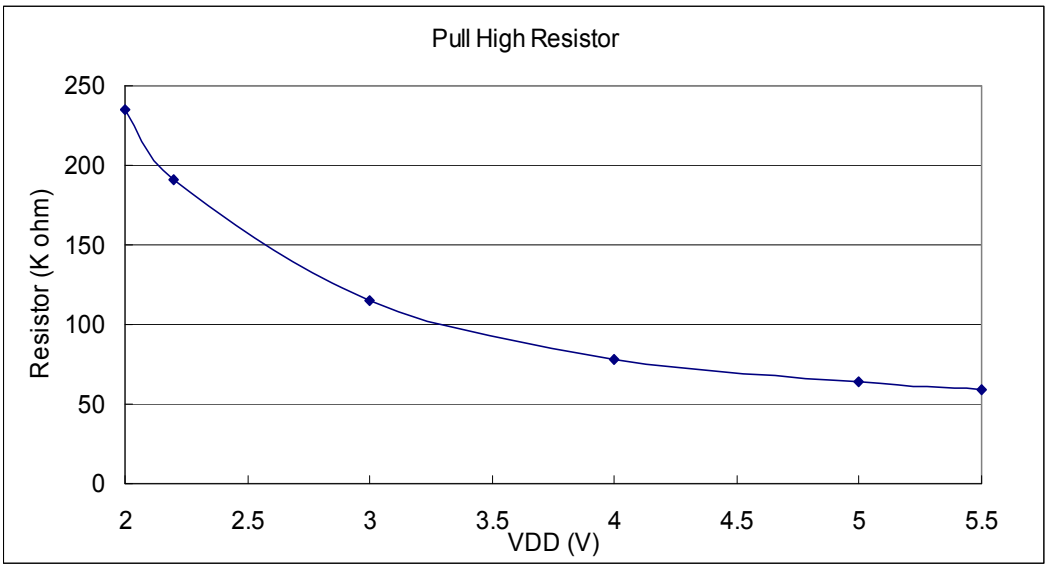
4.12. Typical IO pull high resistance

- Conditions:

  ON: EOSC, MISC.6 = 1;

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

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



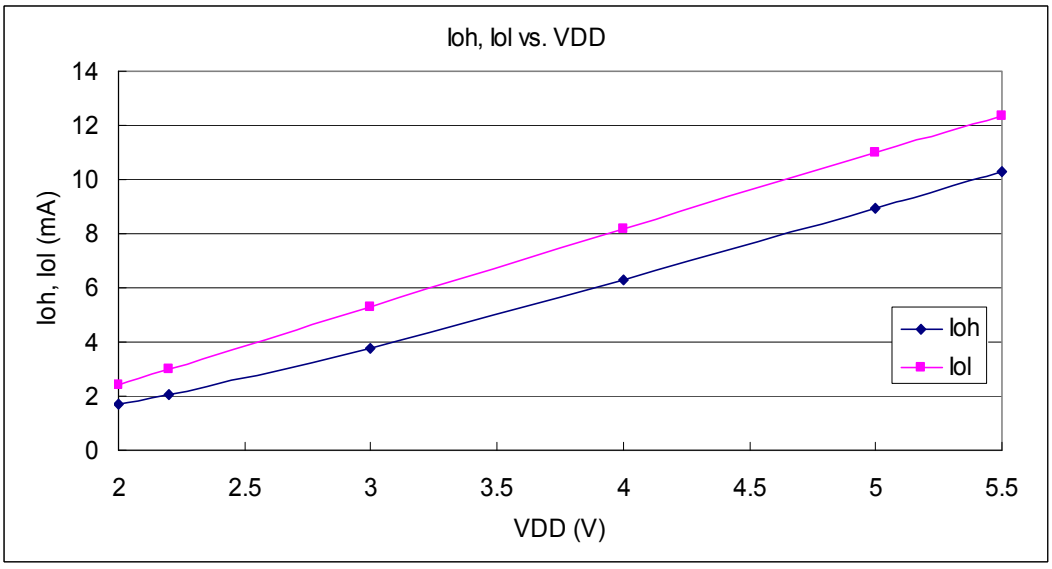
4.13. Typical IO driving current (I_{OH}) and sink current (I_{OL})

- Conditions:

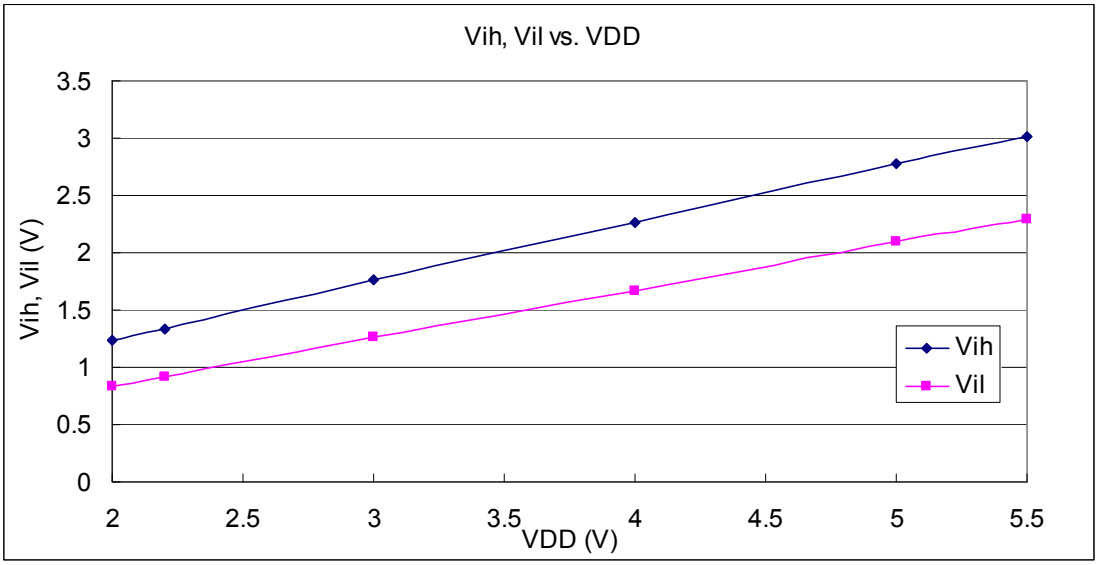
  ON: EOSC, MISC.6 = 1;

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

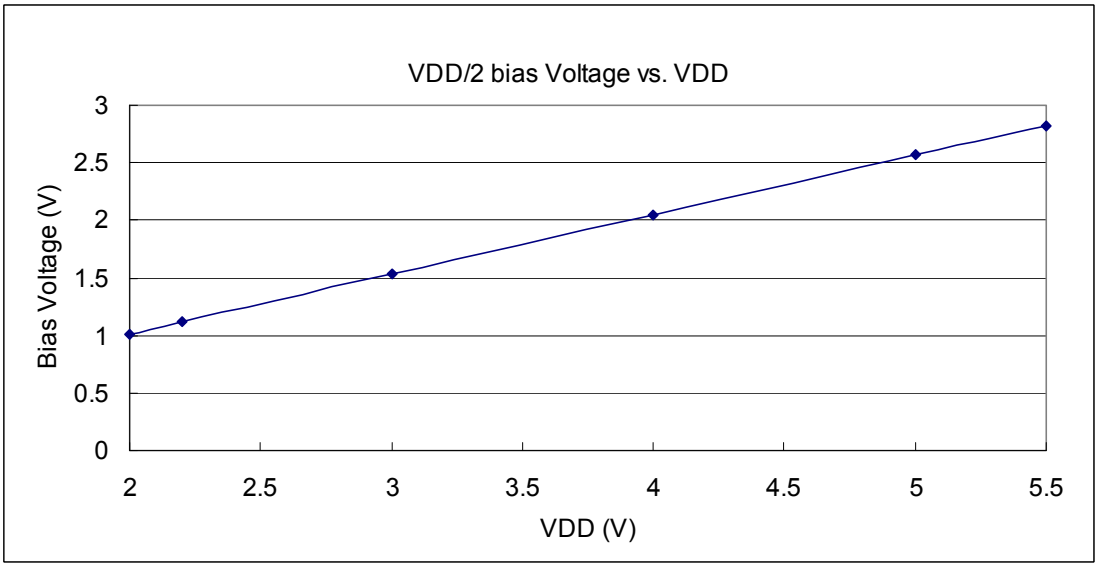
  IO: PA0:0.5Hz output toggle and no loading, others: input and no floating



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



4.15. Typical VDD/2 Bias output voltage



5. Functional Description

5.1. Program Memory – OTP

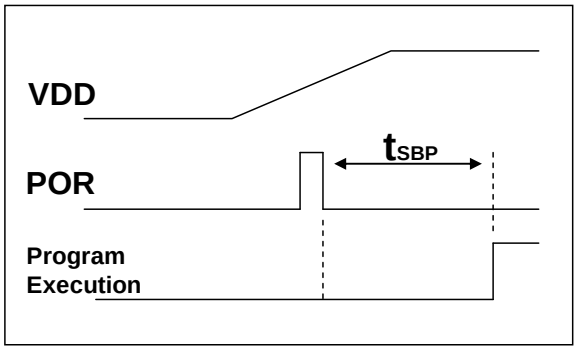
The OTP (One Time Programmable) program memory is used to store the program instructions to be executed. The OTP program memory may contains the data, tables and interrupt entry. After reset, the initial address for CPU is 0x000. The interrupt entry is 0x010 if used, the last eight addresses are reserved for system using, like checksum, serial number, etc. The OTP program memory for PMC156/PMS156 is a 1KW that is partitioned as Table 1. The OTP memory from address 0x3F8 to 0x3FF is for system using, address space from 0x001 to 0x00F and from 0x011 to 0x3F7 are user program spaces.

Address	Function
0x000	Reset – goto instruction
0x001	User program
•	•
•	•
0x00F	User program
0x010	Interrupt entry address
0x011	User program
•	•
0x3F7	User program
0x3F8	System Using
•	•
0x3FF	System Using

Table 1: Program Memory Organization of PMC156/PMS156

5.2. Boot Up

POR (Power-On-Reset) is used to reset PMC156/PMS156 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. 1.



Boot up from Power-On Reset

Fig. 1: Power Up Sequence

5.3. Data Memory – SRAM

The access of data memory 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.
The stack memory is defined in the data memory. The stack pointer is defined in the stack pointer register; 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. All the 64 bytes data memory of PMC156/PMS156 can be accessed by indirect access mechanism.

5.4. Oscillator and clock

There are three oscillator circuits provided by PMC156/PMS156: 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 applications.

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

Table2: Three Oscillator Circuits provided by PMC156/PMS156

5.4.1. 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 2% normally after calibration and it still drifts slightly with supply voltage and operating temperature, the total drift rate is about±5% for VDD=2.2V~5.5V and -40°C~85°C operating conditions. Please refer to the measurement chart for IHRC frequency verse VDD and IHRC frequency verse temperature. The frequency of ILRC is around 37 kHz, however, its frequency will vary by process, supply voltage and temperature, please refer to DC specification and do not use for accurate timing application.

5.4.2. IHRC calibration

The IHRC frequency may be different chip by chip due to manufacturing variation, PMC156/PMS156 provide the IHRC frequency 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
Where,
p1=2, 4, 8, 16, 32; In order to provide different system clock.
p2=16 ~ 18; In order to calibrate the chip to different frequency, 16MHz is the usually one.
p3=2.2 ~ 5.5; In order to calibrate the chip under different supply voltage.

5.4.3. IHRC Frequency Calibration and System Clock

During compiling the user program, the options for IHRC calibration and system clock are shown as Table 3:

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 3: 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 stating 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 PMC156 for different option:

(1) .ADJUST_IC SYSCLK=IHRC/2, IHRC=16MHz, VDD=5V

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

(2) .ADJUST_IC SYSCLK=IHRC/4, IHRC=16MHz, VDD=3.3V

After boot, 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

(3) .ADJUST_IC SYSCLK=IHRC/8, IHRC=16MHz, VDD=2.5V

After boot, 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

(4) .ADJUST_IC SYSCLK=IHRC/16, IHRC=16MHz, VDD=2.2V

After boot, CLKMD = 0x1C：

- IHRC frequency is calibrated to 16MHz@VDD=2.2V and IHRC module is enabled
- System CLK = IHRC/16 = 1MHz
- Watchdog timer is disabled, ILRC is enabled, PA5 is in input mode

(5) .ADJUST_IC SYSCLK=IHRC/32, IHRC=16MHz, VDD=5V

After boot, 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

After boot, CLKMD = 0XE4：

- IHRC frequency is calibrated to 16MHz@VDD=5V and IHRC module is disabled
- System CLK = ILRC
- Watchdog timer is enabled, ILRC is enabled, PA5 is in input mode

(7) .ADJUST_IC DISABLE

After boot, 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

5.4.4. External Crystal Oscillator

If crystal oscillator is used, a crystal or resonator is required between X1 and X2. Fig. 2 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.

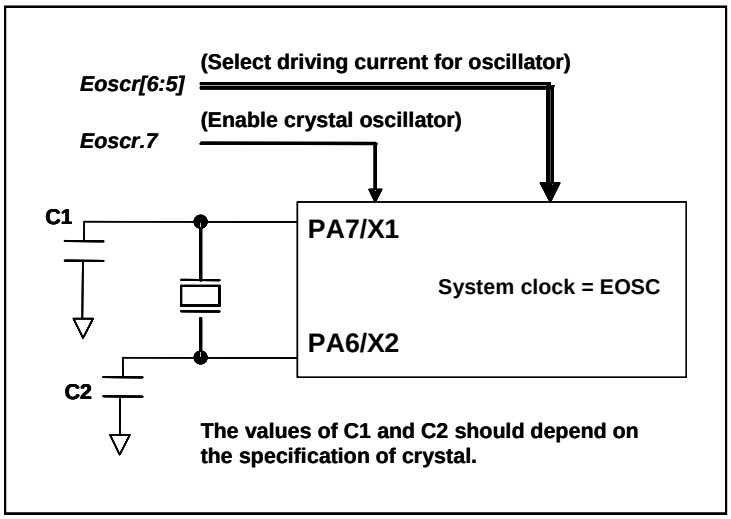


Fig. 2: Connection of crystal oscillator

Besides crystal, external capacitor and options of PMC156 should be fine tuned in *eoscr* (0x0b) register to have good sinusoidal waveform. The *eoscr*.7 is used to enable crystal oscillator module, *eoscr*.6 and *eoscr*.5 are used to set the different driving current to meet the requirement of different frequency of crystal oscillator:

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

Table 4 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 4: 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    CPU (void)
{
  . ADJUST_IC   SYSCLK=IHRC/16, IHRC=16MHz, VDD=5V, Band-gap=On
  // If Band-gap is not calibrated, it can use ". ADJUST_IC    DISABLE" ...
  $   EOSCR   Enable, 4Mhz;       // EOSCR = 0b110_00000;
  $   T16M    EOSC, /1, BIT13;    // T16 receive 2^14=16384 clocks
                                        of crystal osc.,
                                    // Intrq.T16 =>1, crystal osc. Is stable
  WORD   count    =   0;
  stt16count;
  Intrq.T16  =   0;
  wait1      Intrq.T16;       // count fm 0x0000 to 0x2000, then setINTRQ.T16
  clkmd      =   0xA4;            // switch system clock to EOSC;
  ...
```

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

5.4.5. System Clock and LVR levels

The clock source of system clock comes from EOSC, IHRC and ILRC, the hardware diagram of system clock in the PMC156 is shown as Fig. 3.

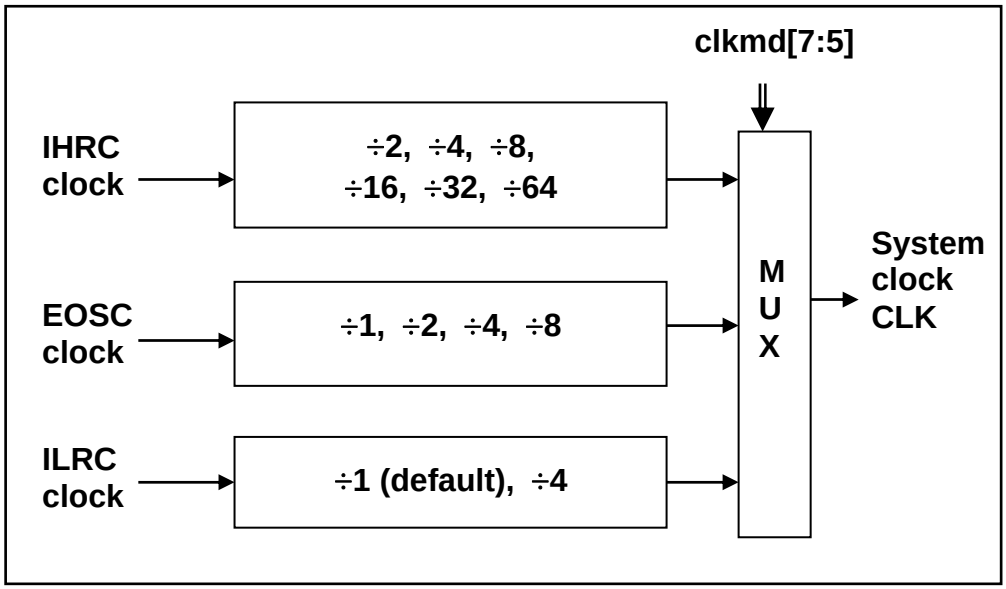


Fig. 3: Options of System Clock

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

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

5.4.6. 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 PMC156 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 -> "IC introduction" -> " Register introduction" -> "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
```

5.5 16-bit Timer (Timer16)

A 16-bit hardware timer (Timer16) is implemented in the PMC156/PMS156, 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. 4. The interrupt source of Timer16 comes from one of bit 8 to 15 of 16-bit counter, and the interrupt type can be rising edge trigger or falling edge trigger which is specified in the bit 4 of ***integs*** register (IO address 0x0C).

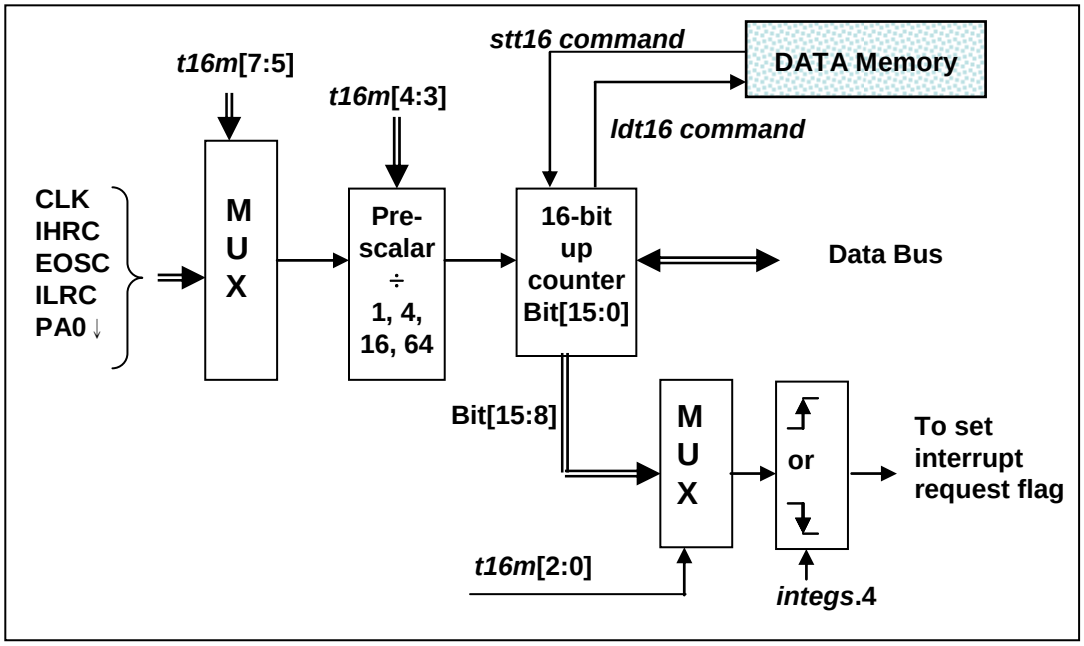


Fig. 4: Hardware diagram of Timer16

When using the Timer16, the syntax for Timer16 has been defined in the .INC file. There are three parameters to define the Timer16 using; 1st parameter is used to define the clock source of Timer16, 2nd parameter is used to define the pre-scalar and the 3rd one is to define the interrupt source.

```
T16M        IO_RW       0x06
    $ 7~5:  STOP, SYSCLK, EOSC,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.

5.6 Watchdog Timer

The watchdog timer (WDT) is a counter with clock coming from ILRC and its frequency is about 37kHz@5V. There are three different timeout periods of watchdog timer can be chosen by setting the ***misc*** register, they are:

- 16384 ILRC clock period when misc[1:0]=10
- 4096 ILRC clock period when misc[1:0]=01
- 2048 ILRC clock period when misc[1:0]=00 (default)

The frequency of ILRC may drift a lot due to the variation of manufacture, supply voltage and temperature; user should reserve guard band for safe operation. WDT can be cleared by power-on-reset or by command ***wdreset*** at any time. When WDT is timeout, PMC156 will be reset to restart the program execution. The relative timing diagram of watchdog timer is shown as Fig. 5.

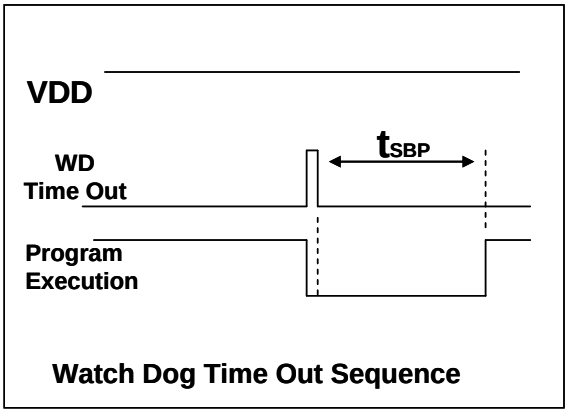


Fig. 5: Sequence of Watch Dog Time Out

5.7 Interrupt

There are three interrupt lines for PMC156:

- External interrupt PA0
- External interrupt PB0
- Timer16 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. 6. 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. 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 could be fully specified by user to achieve maximum flexibility of system.

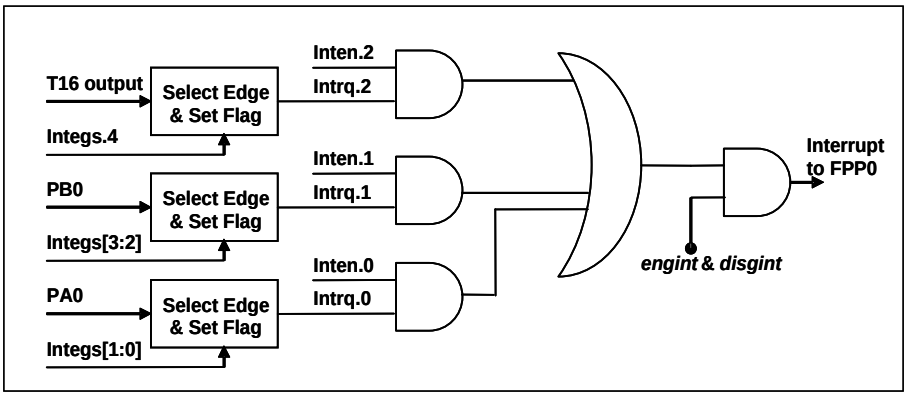


Fig. 6: Hardware diagram of Interrupt controller

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.

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

```
void        Interrupt  (void)    //   interrupt service routine
{
      PUSHAF                     //   store ALU and FLAG register
      If  (INTRQ.0)
      {                          //   Here for PA0 interrupt service routine
        // INTRQ   =   0;        //   User can not use this instruction
           INTRQ.0 =   0;        //   User is recommended to use this instruction
        ...
      }
     ...
     POPAF                      //   restore ALU and FLAG register
}
```

5.8 Power-Save and Power-Down

There are three operational modes defined by hardware: ON mode, Power-Save mode and Power-Down modes. ON mode is the state of normal operation with all functions ON, Power-save mode ("***stopexe***") is the state to reduce operating current and CPU keeps ready to continue, Power-Down mode ("***stopsys***") is used to save power deeply. Therefore, Power-save mode is used in the system which needs low operating power with wake-up occasionally and Power-Down mode is used in the system which needs power down deeply with seldom wake-up. Fig. 7 shows the differences in oscillator modules between Power-Save mode ("***stopexe***") and Power-Down mode ("***stopsys***").

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

Fig. 7: Differences in oscillator modules between STOPSYS and STOPEXE

5.8.1. Power-Save mode ("*stopexe*")

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

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

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

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

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

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

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

5.8.2. 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 PMC156 in detail when "***stopsys***" command is issued:

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

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

```
CMKMD    =    0xF4;    //    change clock from IHRC to ILRC, disable watchdog timer
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
```

5.8.3. Wake-up

After entering the Power-Down or Power-Save modes, the PMC156 can be resumed to normal operation by toggling IO pins, Timer16 interrupt is available for Power-Save mode ONLY. Fig. 8 shows the differences in wake-up sources between STOPSYS and STOPEXE.

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

Fig. 8: Differences in wake-up sources between Power-Save mode and Power-Down mode

When using the IO pins to wake-up the PMC156/PMS156, 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 there is no fast wake-up mode whenever EOSC is enabled.

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

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

To avoid unable wake-up problem happening from drifted process, please switch the system operating frequency to ILRC/1 before executing STOPSYS/STOPEXE instruction, and then switch to the original system operating frequency after waking-up, the example is shown as below:

```
....

$ CLKMD      ILRC/1,En_IHRC,En_ILRC          //SysClk swtch to ILRC

stopsys;                                      //Use stopsys or stopexe

$ CLKMD      IHRC/n,En_IHRC,En_ILRC          //Switch to SysClk after waking-up
```

5.9 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***), control registers (***pac, pbc***) and pull-high registers (***paph, pbph)***. 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 5 shows the configuration table of bit 0 of port A. The hardware diagram of IO buffer is also shown as Fig. 9.

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 5: PA0 Configuration Table

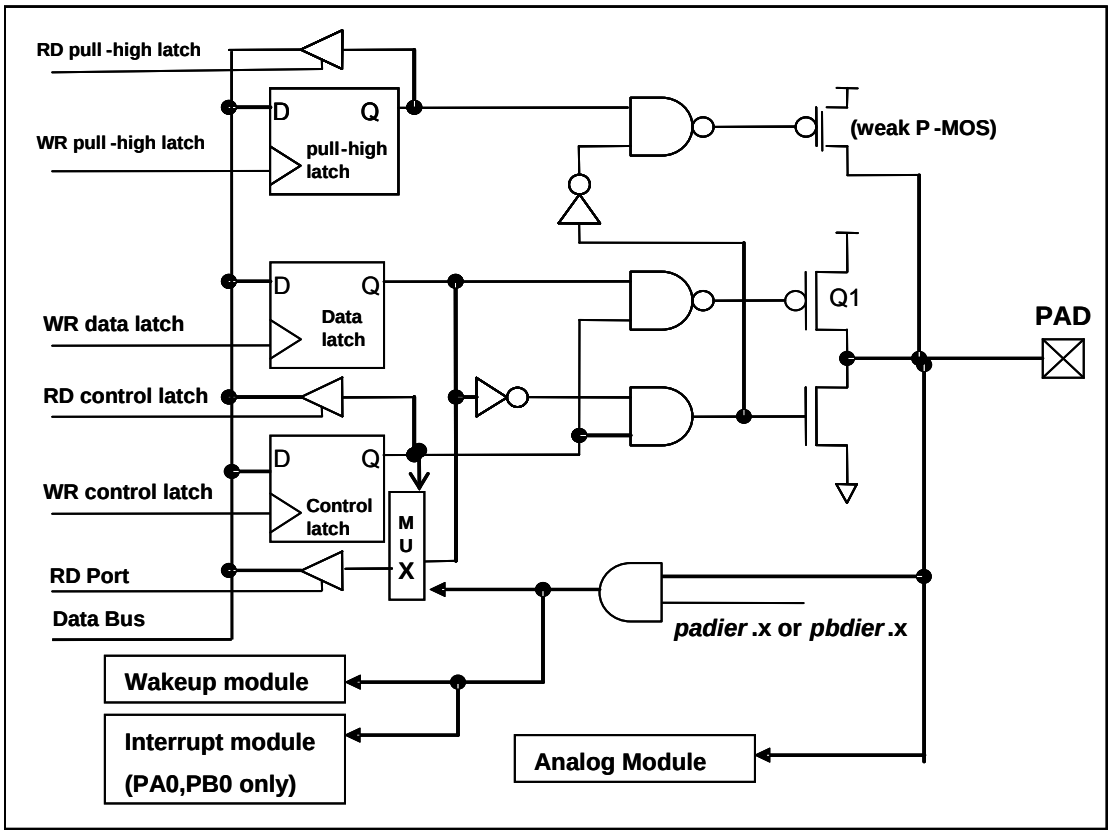


Fig. 9: Hardware diagram of IO buffer

Other than PA5, all the IO pins have the same structure; PA5 can is open-drain ONLY when setting to output mode (without Q1). When PMC156 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*** or ***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.

5.10 Reset and LVR

5.10.1. Reset

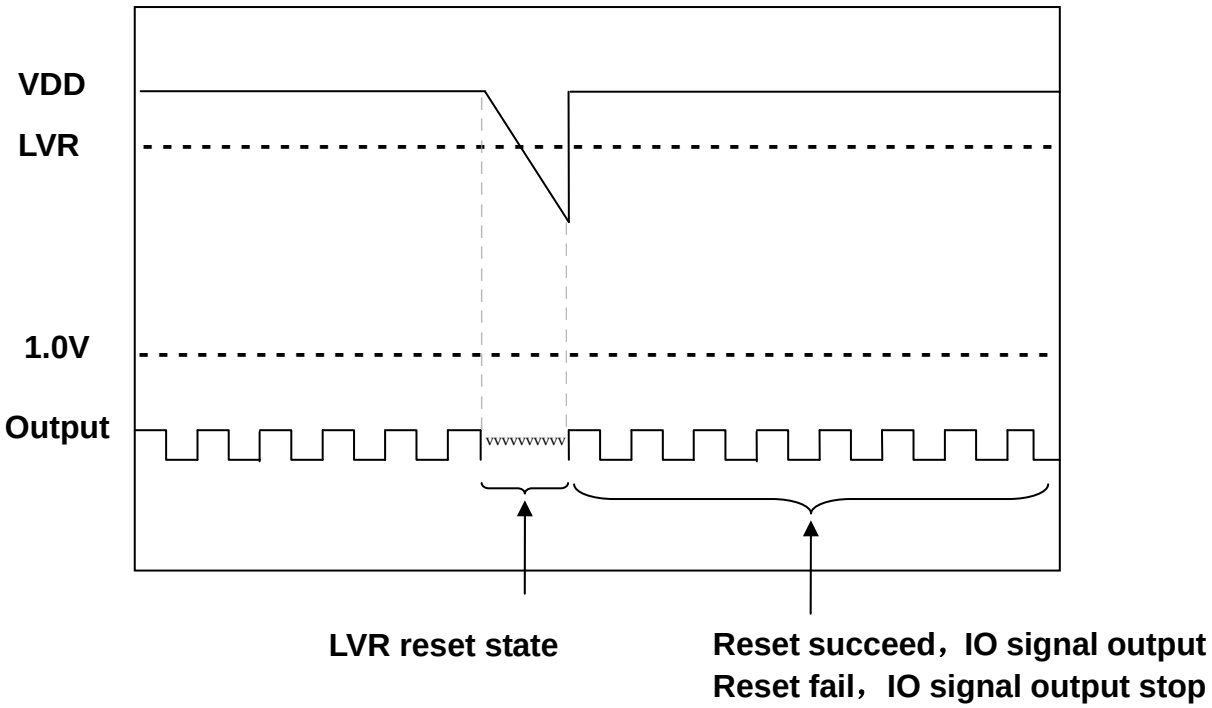
There are many causes to reset the PMC156/PMS156, once reset is asserted, all the registers in PMC156 will be set to default values, system should be restarted once abnormal cases happen, or by jumping program counter to address 'h0. The data memory is in uncertain state when reset comes from power-up and LVR; however, the content will be kept when reset comes from PRST# pin or WDT timeout.

5.10.2. LVR reset

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

5.10.3. Notice for LVR reset

In some applications, the power VDD may change rapidly because of quick switching the power source manually or strong power noise. In case, when the power VDD drops to the level that is lower than the LVD level but higher than 1.0V, if at this time the power VDD is pulled up again to be over LVD level (just see the diagram below), there may be some chances that cause the MCU malfunction or hanged.



To avoid the above problem, please follow the below steps in your program:

Step 1. Insert the below two instructions just after the ***.ADJUST_IC*** instruction

SET1 inten.7

Notice: IDE 0.57 or above version will insert this instruction automatically.

Intrq = 0;

Notice: IDE 0.59 or above version will insert this instruction automatically.

Step 2. Never clear the ***inten.7*** through out the whole program. Please pay special attention in accidental clear ***inten.7*** by writing operation to the whole ***inten*** register. Please consider using ***set1/set0*** instruction to change other interrupt enable flags.

Notice: IDE 0.57 or above version will block the reset operation of ***inten.7*** automatically.

Step 3. When ***wdreset*** instruction is being used:

Please modify the ***wdreset*** instruction inside the main loop of the program:

C language: ***If (inten.7==0) reset; else {wdreset;}***

Assembly language:
```
t1sn    inten.7;
reset
wdreset
```
or use as below :

.wdreset （for IDE 0.57 or above version only）

Step 4. When ***clkmd*** is being used:

When ***clkmd*** instruction is set inside the main loop of the program and ***clkmd***.1 = 0, please insert below instructions afterward.

C language: ***If (inten.7==0) reset;***

Assembly language:
```
t1sn    inten.7;
reset
```
or use as below to set ***clkmd***:

.***clkmd*** = 0x hh;

(“ hh” is a hexadecimal value. For IDE 0.59 or above version only)

5.11 LCD Bias Voltage Generator

This function can be enabled by bit 4 of ***misc*** register. Those pins which are defined to output VDD/2 voltage are PA4、PA3、PA2、PA1（PB6、PB5、PB2、PB1） 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. 10 shows how to use this function.

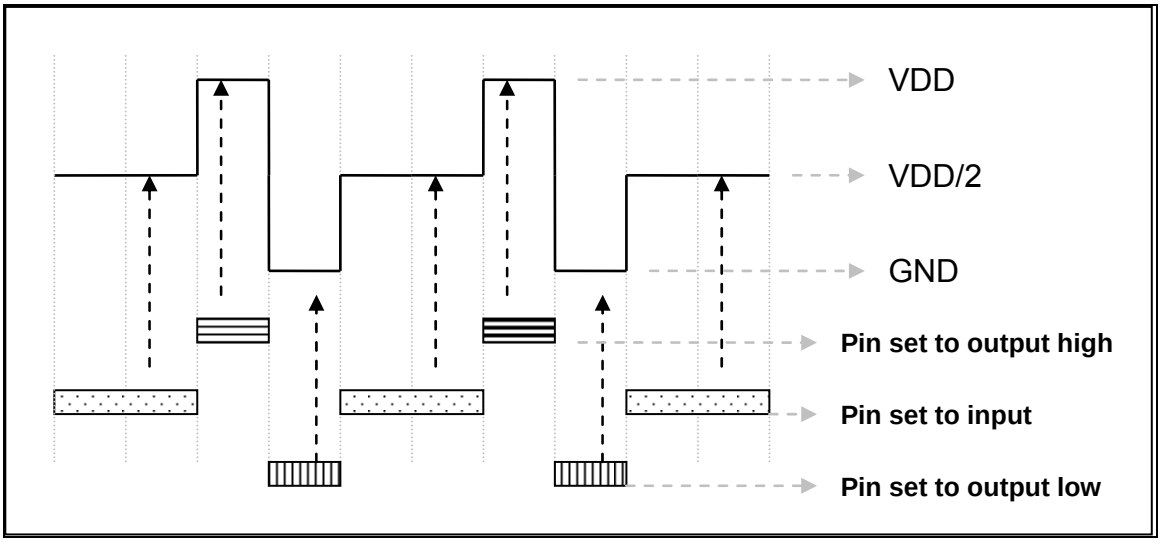


Fig. 10: Using VDD/2 bias voltage generator

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 read.
3	-	R/W	OV (Overflow). This bit is set whenever the sign operation is overflow.
2	-	R/W	AC (Auxiliary Carry). 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	-	R/W	C (Carry). 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	-	R/W	Z (Zero). This bit will be set when the result of arithmetic or logic operation is zero; Otherwise, it is cleared.

6.2 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. Please notice that bit 0 should be kept 0 due to program counter is 16 bits.

6.3 Clock Mode Register (*clkmd*), IO address = 0x03

<table>
<tr><th>Bit</th><th>Reset</th><th>R/W</th><th colspan="2">Description</th></tr>
<tr><td rowspan="3">7 – 5</td><td rowspan="3">111</td><td rowspan="3">R/W</td><td colspan="2">System clock selection:</td></tr>
<tr><td>Type 0, clkmd[3]=0</td><td>Type 1, clkmd[3]=1</td></tr>
<tr><td>000: IHRC/4
001: IHRC/2
010: reserved
011: EOSC/4
100: EOSC/2
101: EOSC
110: ILRC/4
111: ILRC (default)</td><td>000: IHRC/16
001: IHRC/8
010: reserved
011: IHRC/32
100: IHRC/64
101: EOSC/8
11X: reserved</td></tr>
<tr><td>4</td><td>1</td><td>R/W</td><td colspan="2">IHRC oscillator Enable. 0 / 1: disable / enable</td></tr>
<tr><td>3</td><td>0</td><td>RW</td><td colspan="2">Clock Type Select. This bit is used to select the clock type in bit [7:5].
0 / 1: Type 0 / Type 1</td></tr>
<tr><td>2</td><td>1</td><td>R/W</td><td colspan="2">ILRC Enable. 0 / 1: disable / enable
If ILRC is disabled, watchdog timer is also disabled.</td></tr>
<tr><td>1</td><td>1</td><td>R/W</td><td colspan="2">Watch Dog Enable. 0 / 1: disable / enable</td></tr>
<tr><td>0</td><td>0</td><td>R/W</td><td colspan="2">Pin PA5/PRST# function. 0 / 1: PA5 / PRST#.</td></tr>
</table>

6.4 Interrupt Enable Register (inten), IO address = 0x04

Bit	Reset	R/W	Description
7 – 3	-	R/W	Reserved.
2	-	R/W	Enable interrupt from Timer16 overflow. 0 / 1: disable / enable.
1	-	R/W	Enable interrupt from PB0. 0 / 1: disable / enable.
0	-	R/W	Enable interrupt from PA0. 0 / 1: disable / enable.

6.5 Interrupt Request Register (*intrq*), IO address = 0x05

Bit	Reset	R/W	Description
7 – 3	-	R/W	Reserved.
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

6.6 Timer 16 mode Register (t16m), IO address = 0x06

Bit	Reset	R/W	Description
7 – 5	000	R/W	Timer Clock source selection 000: Timer 16 is disabled 001: CLK (system clock) 01x: reserved 100: IHRC 101: EOSC 110: ILRC 111: PA0 falling edge (from external pin)
4 – 3	00	R/W	Internal clock divider. 00: /1 01: /4 10: /16 11: /64
2 – 0	000	R/W	Interrupt source selection. Interrupt event happens when selected bit is changed. 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

6.7 External Oscillator setting Register (*eoscr, write only*), 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.

6.8 IHRC oscillator control Register (*ihrcr, write only*), IO address = 0x0b

Bit	Reset	R/W	Description
7 – 0	--	WO	Bit [7:0] of internal high RC oscillator for frequency calibration. For system using only, please user do NOT write this register.

6.9 Interrupt Edge Select Register (*integs*), IO address = 0x0c

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

6.10 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. 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. If this bit is set to low, PA3 can NOT be used to wake-up the system. Note: For ICE emulation, the function is disabled when this bit is “1” and “0” is enabled.
2	1	WO	Enable PA2 wake-up event. 1 / 0 : enable / disable. This bit can be set to low to disable wake-up from PA2 toggling. **Note**: For ICE emulation, wakeup is disabled when this bit is “1” and “0” is enabled.
1	1	WO	Enable PA1 wake-up event. 1 / 0 : enable / disable. This bit can be set to low to disable wake-up from PA1 toggling. **Note**: For ICE emulation, wakeup is disabled when this bit is “1” and “0” is enabled.
0	1	WO	Enable PA0 wake-up event and interrupt request. 1 / 0 : enable / disable. This bit can be set to low to disable wake-up from PA0 toggling and interrupt request from this pin. **Note**: For ICE emulation, the function is disabled when this bit is “1” and “0” is enabled.

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

“$ PADIER 0xhh” ;

For example:

$ PADIER 0xF0;

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

6.11 Port B Digital Input Enable Register (*pbdier*), IO address = 0x0e

Bit	Reset	R/W	Description
7	1	WO	Enable PB7 digital input and wake-up event. 1 / 0 : enable / disable. If this bit is set to low, PB7 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 PB6 wake-up event. 1 / 0 : enable / disable. This bit can be set to low to disable wake-up from PB6 toggling. Note: For ICE emulation, wakeup is disabled when this bit is “1” and “0” is enabled.
5	1	WO	Enable PB5 wake-up event. 1 / 0 : enable / disable. This bit can be set to low to disable wake-up from PB5 toggling. Note: For ICE emulation, wakeup is disabled when this bit is “1” and “0” is enabled.
4	1	WO	Enable PB4 wake-up event. 1 / 0 : enable / disable. This bit can be set to low to disable wake-up from PB4 toggling. Note: For ICE emulation, wakeup is disabled when this bit is “1” and “0” is enabled.
3	1	WO	Enable PB3 digital input and wake-up event. 1 / 0 : enable / disable. If this bit is set to low, PB3 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 PB2 digital input and wake-up event. 1 / 0 : enable / disable. If this bit is set to low, PB2 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.
1	1	WO	Enable PB1 digital input and wake-up event. 1 / 0 : enable / disable. If this bit is set to low, PB1 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.
0	1	WO	Enable PB0 digital input, wake-up event and interrupt request. 1 / 0 : enable / disable. If this bit is set to low, PB0 can NOT be used to wake-up the system and interrupt request from this pin. Note: For ICE emulation, the function is disabled when this bit is “1” and “0” is enabled.

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

“$ PBDIER 0xhh” ;

For example:

$ PBDIER 0x F0;

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

6.12 Port A Data Registers (*pa*), IO address = 0x10

Bit	Reset	R/W	Description
7 – 0	0x00	R/W	Data registers for Port A.

6.13 Port A Control Registers (*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.

6.14 Port A Pull-High Registers (*paph*), IO address = 0x12

Bit	Reset	R/W	Description
7–0	0x00	R/W	Port A pull-high registers. This register is used to enable the internal pull-high device on each corresponding pin of port A. 0 / 1 : disable / enable Please note that the PA5 does not have pull-up resistor.

6.15 Port B Data Registers (*pb*), IO address = 0x14

Bit	Reset	R/W	Description
7–0	0x00	R/W	Data registers for Port B.

6.16 Port B Control Registers (*pbc*), IO address = 0x15

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

6.17 Port B Pull-High Registers (*pbph*), IO address = 0x16

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

6.18 MISC Register (misc), IO address = 0x3b

Bit	Reset	R/W	Description
7	-	-	Reserved
6	0	WO	Enable extremely low current for 32kHz crystal oscillator AFTER oscillation. 0: Normal. 1: Low driving current for 32kHz crystal oscillator.
5	0	WO	Enable fast Wake-up. Fast wake-up is NOT supported when EOSC is enabled. 0: Normal wake-up. The wake-up time is 1024 ILRC clocks 1: Fast wake-up. The wake-up time is 128 CLKs (system clock) + oscillator stable time. If wake-up from STOPEXE suspend, there is no oscillator stable time; If wake-up from STOPSYS suspend, it will be IHRC or ILRC stable time from power-on. Please notice that the clock source 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	-	-	Enable to generate half VDD on PA1/PA2/PA3/PA4 or PB1/PB2/PB5/PB6 pins. User can choose group 1 or 2 of 1/2 VDD from Code Option. 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: Reserved

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 CPU
word	Only addressed in 0~0x1F (0~31) is allowed
M.n	Only addressed in 0~0xF (0~15) is allowed

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

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 Timer 16
	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
	…
	--- 	

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 address (MSB), should be 0
	mov hb@RAMIndex, a ; // save pointer to RAM (MSB)
	…
	idxm a, RAMIndex ; // mov memory 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 address (MSB), should be 0
	mov hb@RAMIndex, a ; // save pointer to RAM (MSB)
	…
	mov a, 0xA5 ;
	idxm RAMIndex, a ; // mov 0xA5 to memory in address 0x5B
	--- 	

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

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
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, shift 0 to bit 7 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 bit 7 to flag 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, shift 0 to bit 7 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 bit 7 to flag 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 shift 0 to bit 0 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 bit 0 to flag 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, shift 0 to bit 0 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 bit 0 to flag 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

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 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
	not mem ; // mem = 0xC8
	--- 	

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
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

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
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

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
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 CPU Affected flags: 『N』Z 『N』C 『N』AC 『N』OV
disgint	Disable global interrupt enable Example: *disgint* ; Result: Interrupt request is blocked from CPU 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

2T	*goto, call, , idxm*
1T/2T	*ceqsn, t0sn, t1sn, dzsn, izsn*
1T	Others

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	-	-	-	-	*ldt16* word	-	-	-	-
stt16 word	-	-	-	-	*idxm* a, index	-	-	-	-	*idxm* index, a	-	-	-	-
xch M	-	-	-	-	*pushaf*	-	-	-	-	*popaf*	Y	Y	Y	Y
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	*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	-	-	-	-	*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 IO, a	-	-	-	-	*xor* a, M	Y	-	-	-	*xor* M, a	Y	-	-	-
not a	Y	-	-	-	*not* M	Y	-	-	-	*neg* a	Y	-	-	-
neg M	Y	-	-	-	*set0* IO.n	-	-	-	-	*set1* IO.n	-	-	-	-
set0 M.n	-	-	-	-	*set1* M.n	-	-	-	-	*ceqsn* a, I	Y	Y	Y	Y
ceqsn a, M	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
call label	-	-	-	-	*goto* label	-	-	-	-	*ret* I	-	-	-	-
ret	-	-	-	-	*reti*	-	-	-	-	*nop*	-	-	-	-
pcadd a	-	-	-	-	*engint*	-	-	-	-	*disgint*	-	-	-	-
stopsys	-	-	-	-	*stopexe*	-	-	-	-	*reset*	-	-	-	
wdreset	-	-	-	-										

8. Special Notes

This chapter is to remind user who use PMC153/PMS153 series IC in order to avoid frequent errors upon operation.

8.1. Using IC

8.1.1. IO pin usage and setting

(1) If IO pin is set to be digital input and enable wake-up function

- Configure IO pin as input
- Set corresponding bit to “1” in PADIER
- The function of PADIER register of PMC156/PMS156 series IC is contrary to that of ICE. In order to keep ICE emulation consisting with PMC156/PMS156 real chip, please use the following procedure:

```
$ PADIER 0xF0;

$ PBDIER 0x07;
```

(2) PA5 is set to be output pin

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

(3) PA5 is set to be PRST# input pin

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

(4) PA5 is set to be input pin and 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 in such application.

(5) PA7 and PA6 as external crystal oscillator

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

8.1.2. Interrupt

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

Step1: Set INTEN register, enable the interrupt control bit

Step2: Clear INTRQ register

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

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

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

* Use DISGINT in the main program to disable all interrupts
* When interrupt service routine starts, use PUSHAF instruction to save ALU and FLAG register. POPAF instruction is to restore ALU and FLAG register before RETI as below:

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

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

8.1.3. 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 荡。
// Delay for EOSC establishment
$ T16M EOSC,/1,BIT10
```

```
Word Count = 0;
Stt16 Count;
Intrq.T16 = 0;
wait1       Intrq.T16;
CLKMD   =   0xA4;                   // ILRC -> EOSC;
CLKMD.2 =   0;                      // turn off ILRC but not necessarily routine
```

The delay duration should be adjusted in accordance with the characteristic of the crystal and PCB.To measure the oscillator signal by the oscilloscope, please select (x10) on the probe and measure through PA6(X2) pin to avoid the interference on the oscillator.

8.1.4. Power down mode, wakeup and watchdog

(1) Watchdog will be inactive once ILRC is disabled

(2) Please turn off watchdog before executing STOPSYS or STOPEXE instruction, otherwise IC will be reset due to watchdog timeout. It is the same as in ICE emulation.

(3) The clock source of Watchdog is ILRC if the fast wakeup is disabled; otherwise, the clock source of Watchdog will be the system clock and the reset time from watchdog becomes much shorter. It is recommended to disable Watchdog and enable fast wakeup before entering STOPSYS mode. When the system is waken up from power down mode, please firstly disable fast wakeup function, and then enable Watchdog. It is to avoid system to be reset after being waken up.

(4) If enable Watchdog during programming and also wants the fast wakeup, the example as below:

```
CLKMD.En_WatchDog   =   0;          // disable watchdog timer
$ MISC   Fast_Wake_Up;
stopexe;
nop;
$ MISC   WT_xx;                     // Reset Watchdog time to normal wake-up
Wdreset;
CLKMD.En_WatchDog   =   1;          // enable watchdog timer
```

8.1.5. TIMER time out

When select T16M counter BIT8 as 1 to generate interrupt, the first interrupt will occur when the counter reaches to 0x100 (BIT8 from 0 to 1) and the second interrupt will occur when the counter reaches 0x300（BIT8 from 0 to 1）. Therefore, selecting BIT8 as 1 to generate interrupt means that the interrupt occurs every 512 counts. Please notice that if T16M counter is restarted, the next interrupt will occur once Bit8 turns from 0 to 1.

8.1.6. LVR

(1) VDD must reach or above 2.0V 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.

(3) User can set EOSCR.0 as “1” to disable LVR. However, VDD must be kept as exceeding the lowest working voltage of chip; Otherwise IC may work abnormally.

8.1.7. Instructions

(1) There are 80 instructions are provided by PMC156/PMS156.

(2) Only single FPPA is built inside the PMC156/PMS156, the executing cycles for different instructions are shown as below:

Instruction	Condition	CPU
goto, call		**2T**
ceqsn, cneqsn, t0sn, t1sn, dzsn, izsn	Condition is fulfilled	**2T**
	Condition is not fulfilled	1T
idxm		2T
Others		1T

8.1.8. RAM definition

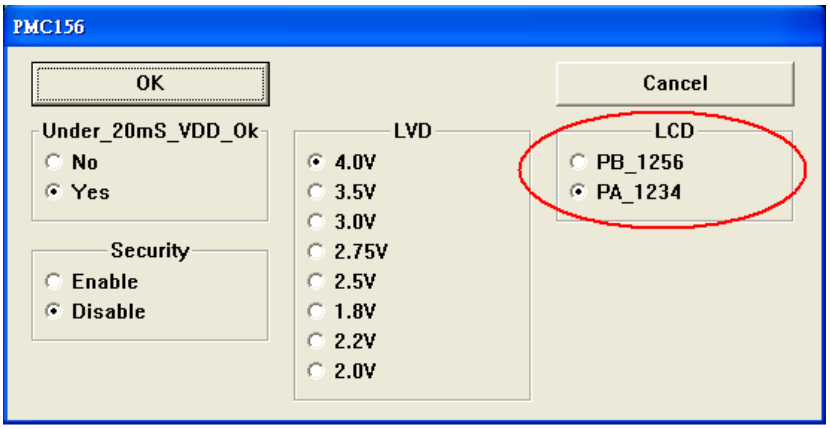
(1) Bit defined: Only addressed at 0x00 ~ 0x0F

(2) WORD defined : Only addressed at 0x00 ~ 0x1E

8.1.9. LCD COM pin application

(1) There are two groups of LCD COM IO selectable for PMC156/PMS156, they are PA_1234(PA1/COM1,PA2/COM2,PA3/COM3,PA4/COM4) and PB_1256(PB1/COM0A,PB2/COM1A,PB5/COM2A,PB6/COM3A).

(2) In the practical application, only one group is actually used. User can select one of the groups in Code Option where the LVR value is also selectable before program compiling. The code option is shown as below. Please notice that once the code option is confirmed, any change will be unacceptable in the next programming.



8.1.10. Program writing

PMC156/PMS156: Put the jumper over the CN40 location.

8.2. Using ICE

(1) PDK3S-I-001/002/003 emulators are designed to emulate at least 2-FPPA mode situation. They can not fully emulate the 1-FPPA mode situation. Even though PMC156/PMS156 series (1-FPPA) have been selected, the ICE still run at 2-FPPA mode; and hence the ICE runs only approximately half speed of the Real Chip under the same system clock setting. It is recommended to double the system clock when using ICE for better emulation. However, because of the executing cycle requirement of some instructions at 1-FPPA and 2-FPPA mode being different, there will be still some timing differences between ICE and Real Chip. Verifying your program timing and functions using Real Chip is a MUST.

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

(2) Total 8 pins with COM function in 2 groups are available for PMC156/PMS156. Each pin provides 1/2 VDD LCD driving voltage for LCD application. PDK3S-I-001/002/003 emulators are able to emulate the LCD COM function of group PA_1234 instead of those of group PB_1256. When using the PB_1256 in the application, both pull high and pull down resistors are required to be connected with COM IO for emulation.

(3) PMC156/PMS156 pin out is compatible with PDK82C12. User can find the corresponding pins at the ICE and connect them correctly to the target board using suitable bus cable or DuPont wires.

8.3. 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/technical-application.php