PMC150/PMS150 Series
8-bit OTP Type IO Controller
Data Sheet

Version 1.08 – Dec. 11, 2018

6F-6, No.1, Sec. 3, Gongdao 5th Rd., Hsinchu City 30069, Taiwan, R.O.C.
TEL: 886-3-572-8688 www.padauk.com.tw

IMPORTANT NOTICE

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

PADAUK Technology products are not warranted to be suitable for use in life-support applications or other critical applications. PADAUK Technology assumes no liability for such applications. Critical applications include, but are not limited to, those 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	2013/12/27	Add section 5.10.3 Notice for LVR reset
0.03	2014/02/12	Add chapter 8 Special Notes
0.04	2014/12/22	Amend PMS150 operating temperature to -40°C ~ 85°C
0.05	2015/06/17	Amend PMS150 operating temperature to -20°C ~ 70°C
0.06	2016/07/06	1. Add section 5.8.3: the description of wake-up 2. Add section 8.3 Warning
0.07	2017/06/13	1. Add section 8.2.7: IHRC description 2. Delete chapter 3: PA1 description 3. Add section 1.4: Package Information 4. Delete chapter 3: MSOP10 Pin Assignment and add SOP8 Pin Assignment
1.08	2018/12/11	1. Add company address & Tel No. 2. Amend Section 1.1, 1.2, 1.3 3. Add Section 1.4: PMC150-U06 & PMS150-U06 Package Information 4. Add Chapter 3: SOT23-6 Pin Assignment 5. Amend Section 4.1, 4.3 to 4.11 6. Add Section 4.5 Typical ILRC Frequency vs. Temperature 7. Add Section 4.12 Typical power down current (I_{PD}) and power save current (I_{PS}) 8. Add Section 5.2.1 Timing charts for reset conditions 9. Amend Section 5.4 Oscillator and clock 10. Amend Section 5.4.1, 5.4.3, 5.4.4 11. Amend Section 5.5 16-bit Timer 12. Amend Section 5.7 Interrupt 13. Amend Section 5.8.1, 5.8.2, 5.8.3 14. Amend Section 5.10.2, 5.10.3 15. Amend Section 6.4, 6.5, 6.6, 6.8, 6.10, 6.11, 6.12, 6.13, 6.14 16. Delete the Symbol “pc0” in Chapter 7 17. Amend Section 7.8 Summary of Instructions Execution Cycle and delete 9.2.8 18. Move Section 9.2.9 BIT definition to Section 7.10 19. Add Chapter 8 Code Options 20. Updated the link in Section 9.1 21. Amend Section 9.2.1, 9.2.5, 9.2.8 22. Amend Section 9.2.8 Program writing

1. Features

1.1. Special Features

- PMC150 series:
  - High EFT series
  - Operating temperature range: -40°C ~ 85°C
- PMS150 series:
  - General purpose series
  - Not supposed to use in AC RC step-down powered or high EFT requirement applications. PADAUK assumes no liability if such kind of applications can not pass the safety regulation tests.
  - Operating temperature range: -20°C ~ 70°C

1.2. System Features

- 1KW OTP program memory
- 60 Bytes data RAM
- One hardware 16-bit timer
- Support fast wake-up
- Internal High RC Oscillator (IHRC) frequency
- Band-gap circuit to provide 1.20V reference voltage
- 6 IO pins with 10mA capability and optional pull-high resistor
- Operating frequency range:

  DC ~ 8MHz@V_{DD}≧3.3V; DC ~ 4MHz@V_{DD}≧2.5V; DC ~ 2MHz@V_{DD}≧2.2V
- Operating voltage range: 2.2V ~ 5.5V
- Low power consumption

  $I_{operating}$ ~ 1.7mA@1MIPS, V_{DD}=5.0V $I_{operating}$ ~ 8uA@ILRC=21KHz, V_{DD}=3.3V

  $I_{powerdown}$ ~ 1uA@V_{DD}=5.0V $I_{powerdown}$ ~ 0.5uA@V_{DD}=3.3V
- Clock sources: internal high RC oscillator and internal low RC oscillator
- Every IO pin can be configured to enable wake-up function
- Eight levels of LVR reset ~ 4.1V, 3.6V, 3.1V, 2.8V, 2.5V, 2.2V, 2.0V, 1.8V
- One external interrupt pins

1.3. CPU Features

- One processing unit operating mode
- 79 Powerful instructions
- Most instructions are 1T execution cycle
- Programmable stack pointer and adjustable stack level
- Direct and indirect addressing modes for data access. Data memories are available for use as an index pointer of Indirect addressing mode
- IO space and memory space are independent

1.4. Package Information

- **PMC150 series**

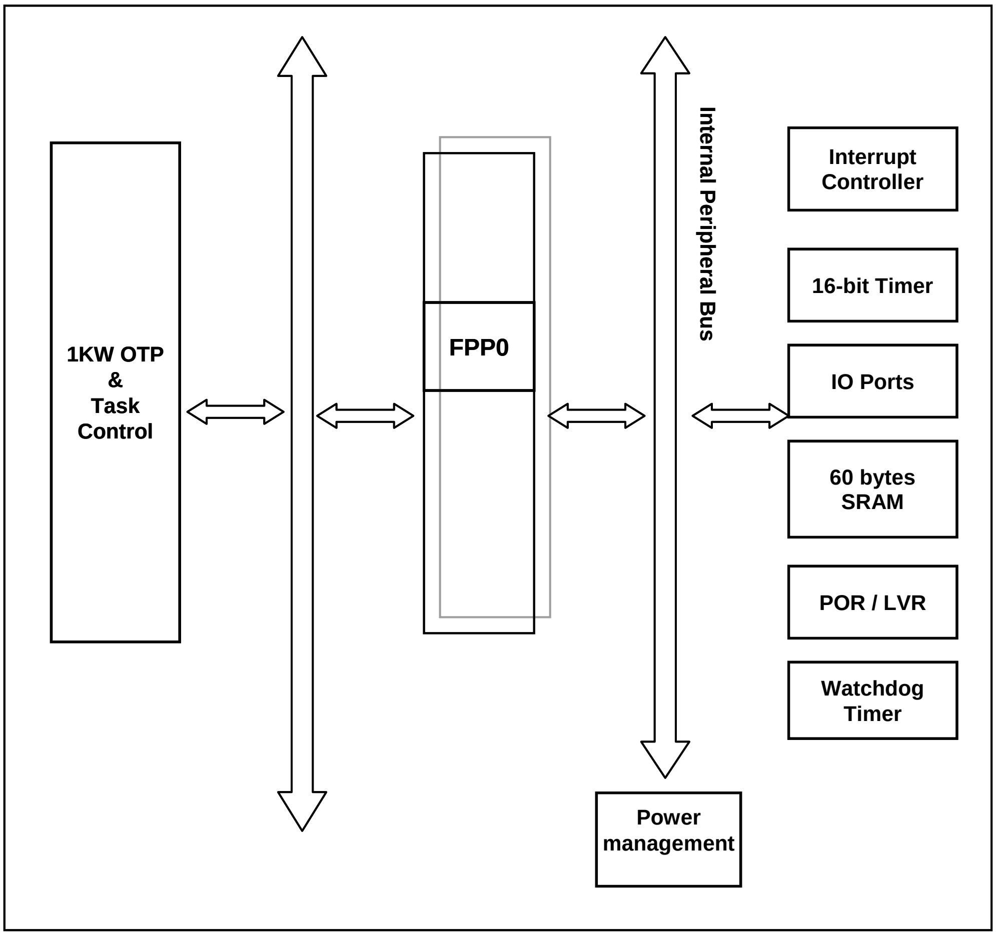
  PMC150 – S08: SOP8 (150mil)
  PMC150 – U06: SOT23-6

- **PMS150 series**

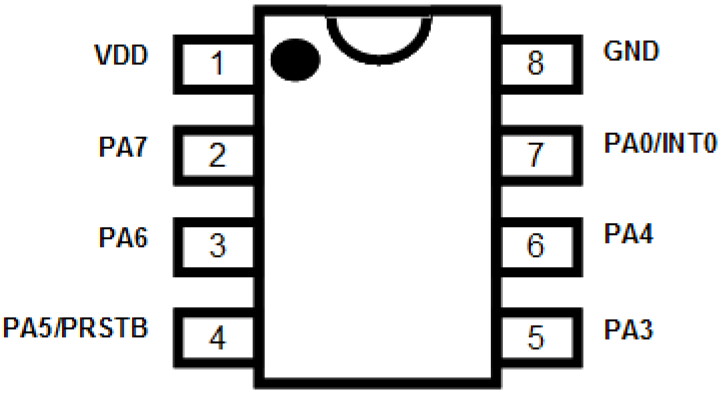
  PMS150 – S08: SOP8 (150mil)
  PMS150 – U06: SOT23-6

2. General Description and Block Diagram

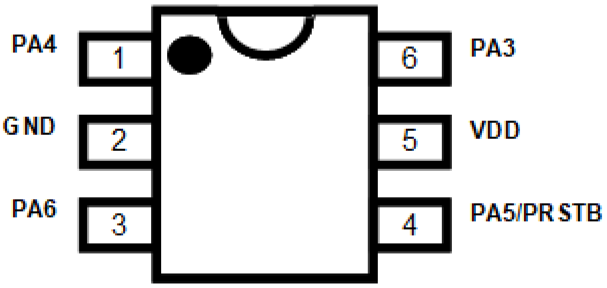
The PMC150/PMS150 is an IO-Type, fully static, OTP-based CMOS 8-bit micro controller; it employs RISC architecture and most the instructions are executed in one cycle except that few instructions are two cycles that handle indirect memory access. 1KW bits OTP program memory and 60 bytes data SRAM are inside, one hardware 16-bit timer is also provided in the PMC150/PMS150.



3. Pin Assignment and Functional Description



PMC150/PMS150-S08 (SOP8-150mil)



PMC150/PMS150-U06(SOT23-6)

Pin Name	Pin & Buffer Type	Description
PA7	IO ST / CMOS	The functions of this pin can be bit 7 of port A. It can be configured as digital input or two-state output, with pull-high resistor. 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	IO ST / CMOS	The functions of this pin can be bit 6 of port A. It can be configured as digital input or two-state output, with pull-high resistor. 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 digital input or open-drain output. Please notice that there is no pull-high 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	IO ST / CMOS	The functions of this pin can be bit 4 of port A. It can be configured as digital input or two-state output, with pull-high resistor. This pin can be used to wake up system during sleep mode; however, wake-up function from this pin is disabled when bit 4 of ***padier*** register is “0”.
PA3	IO ST / CMOS	The functions of this pin can be bit 3 of port A. It can be configured as digital input or two-state output, with pull-high resistor. This pin can be used to wake up system during sleep mode; however, wake-up function from this pin is disabled when bit 3 of ***padier*** register is “0” .
PA0/INT0	IO ST / CMOS	The functions of this pin can be: (1) Bit 0 of port A. It can be configured as digital input or two-state output, with pull-high 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”.
NC	-	No Connection
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 V_{DD} =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	* Subject to LVR tolerance
f_{SYS}	System clock (CLK)* = IHRC/2 IHRC/4 IHRC/8 ILRC	 0 0 0	 37K	 8M 4M 2M	Hz	Under_20ms_Vdd_ok** = Y/N V_{DD} ≧ 2.5V / V_{DD} ≧ 3.1V V_{DD} ≧ 2.2V / V_{DD} ≧ 2.5V V_{DD} ≧ 2.2V / V_{DD} ≧ 2.2V V_{DD}=5.0V
I_{OP}	Operating Current		1 6		mA uA	f_{SYS}=IHRC/16=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,V_{DD}=5.0V f_{SYS}= 0Hz,V_{DD}=3.3V
I_{PS}	Power Save Current (by ***stopexe*** command)		0.4		mA	V_{DD}=5.0V; Band-gap, LVR, IHRC, ILRC, Timer16 modules are ON.
V_{IL}	Input low voltage for IO lines	0		0.3V_{DD}	V	
V_{IH}	Input high voltage for IO lines	0.7 V_{DD}		V_{DD}	V	
I_{OL}	IO lines sink current	7	10	13	mA	V_{DD}=5.0V, V_{OL}=0.5V
I_{OH}	IO lines drive current	-5	-7	-9	mA	V_{DD}=5.0V, V_{OH}=4.5V
V_{IN}	Input voltage	-0.3		V_{DD} +0.3	V	
$I_{INJ (PIN)}$	Injected current on pin			1	mA	V_{DD} +0.3≧V_{IN}≧ -0.3
R_{PH}	Pull-high Resistance		62 100 210		KΩ	V_{DD}=5.0V V_{DD}=3.3V V_{DD}=2.2V
V_{LVR}	Low Voltage Detect 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*	16*	16.16*	MHz	@25°C
		15.20* 15.28*	16* 16*	16.80* 16.72*		V_{DD} =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	V_{DD}=5.0V, Ta=25°C
		24.0* 25.9*	37* 37*	50.0* 48.1*		V_{DD}=5.0V, -40°C <Ta<85°C* V_{DD}=5.0V, -20°C <Ta<70°C*
		18.3*	21*	24.5*		V_{DD}=3.3V, Ta=25°C
		14.0* 14.7*	21* 21*	29.0* 27.3*		V_{DD}=3.3V, -40°C <Ta<85°C* V_{DD}=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
			256			misc[1:0]=11
t_{SBP}	System boot-up period from power-on		29 48		ms	@V_{DD}=5V, ILRC~37KHz @V_{DD}=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	@V_{DD}=5V

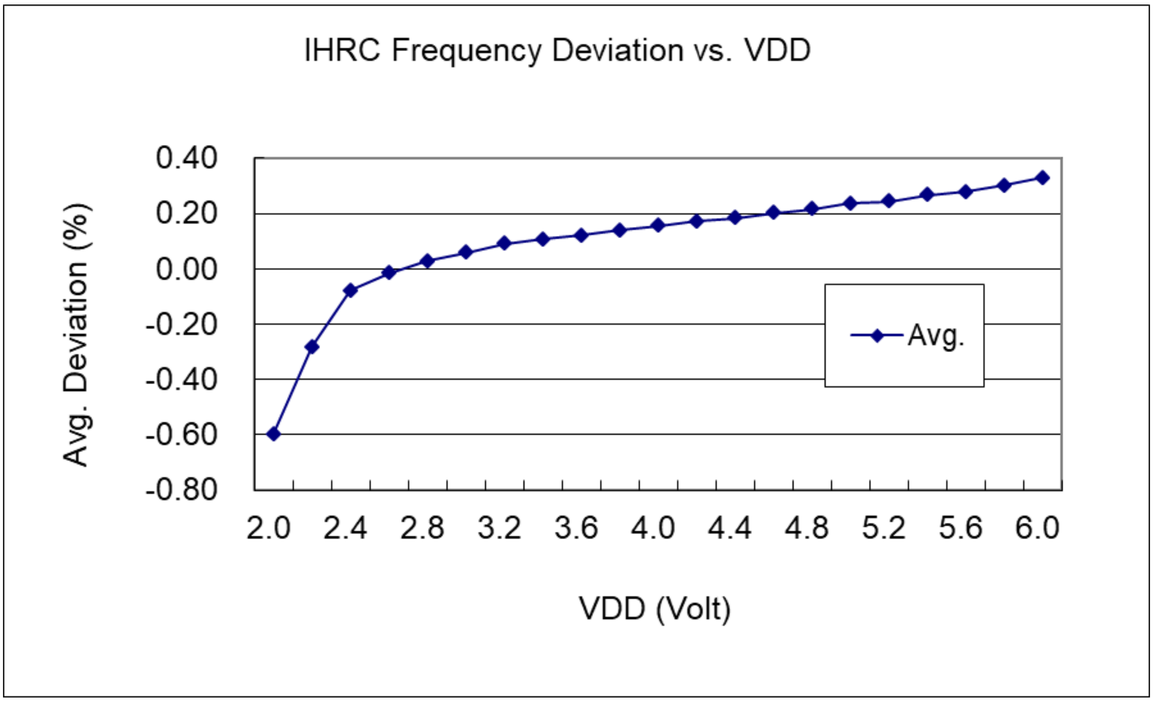
*These parameters are for design reference, not tested for every chip.

** Under_20ms_ V_{DD} _Ok is a checking condition for the V_{DD} rising from 0V to the stated voltage within 20ms.

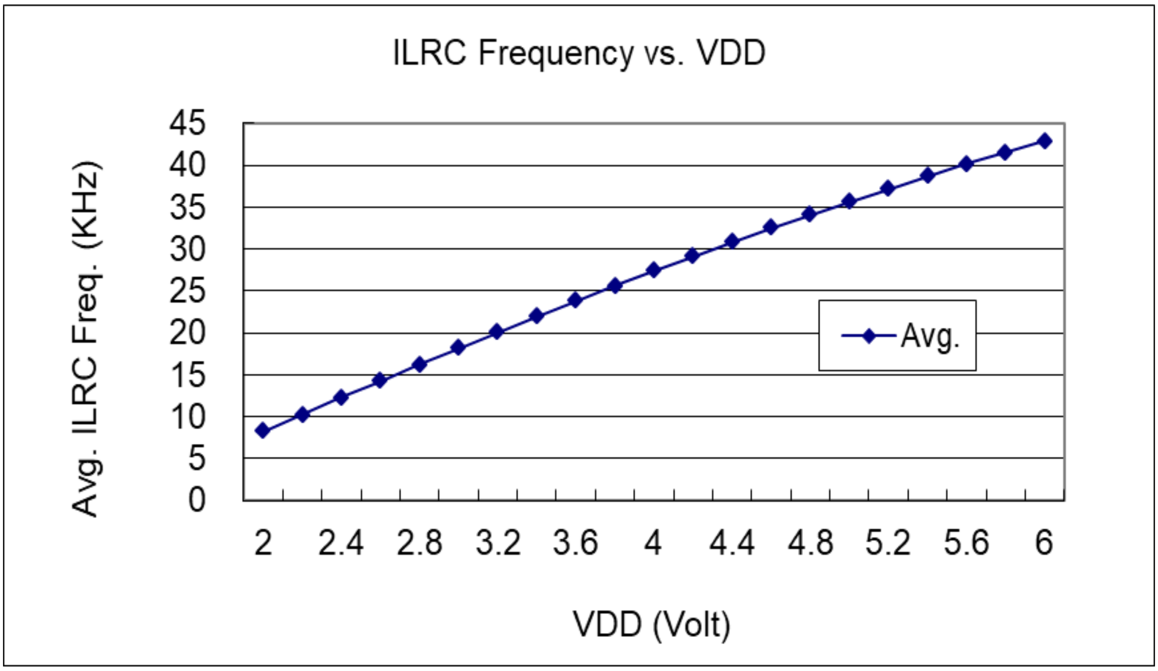
4.2. Absolute Maximum Ratings

- Supply Voltage ……………………………......2.2V ~ 5.5V
- Input Voltage ………………………………….-0.3V ~ V_{DD} + 0.3V
- Operating Temperature ……………………… PMC150 series:-40°C ~ 85°C;
  PMS150 series:-20°C ~ 70°C
- Storage Temperature …………………………-50°C ~ 125°C
- Junction Temperature ……………………….. 150°C

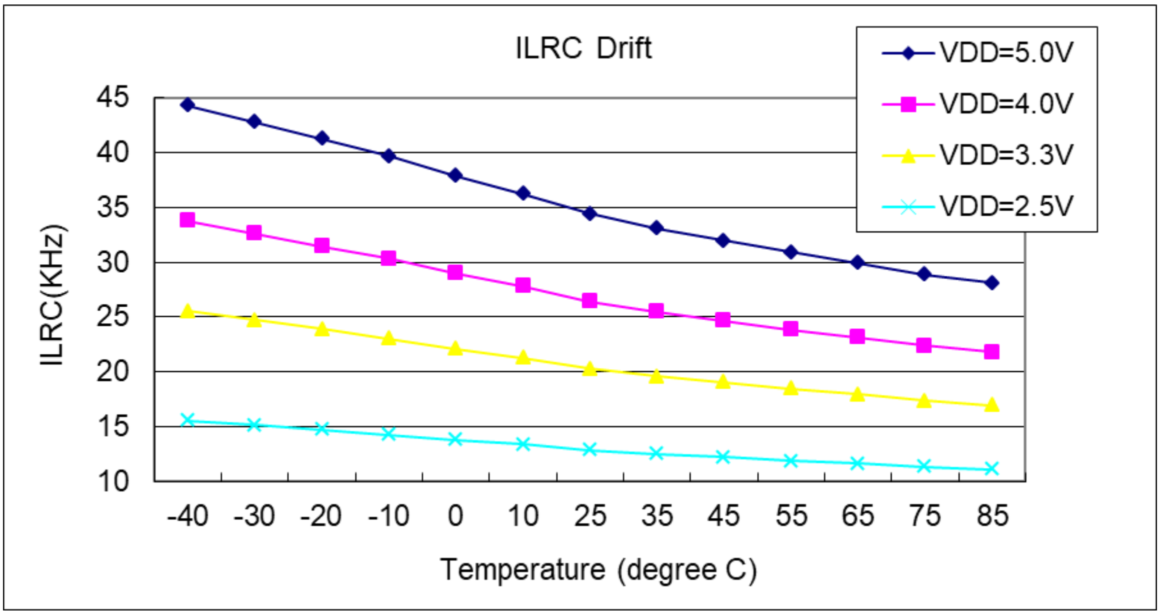
4.3. Typical IHRC Frequency vs. VDD (calibrated to 16MHz)



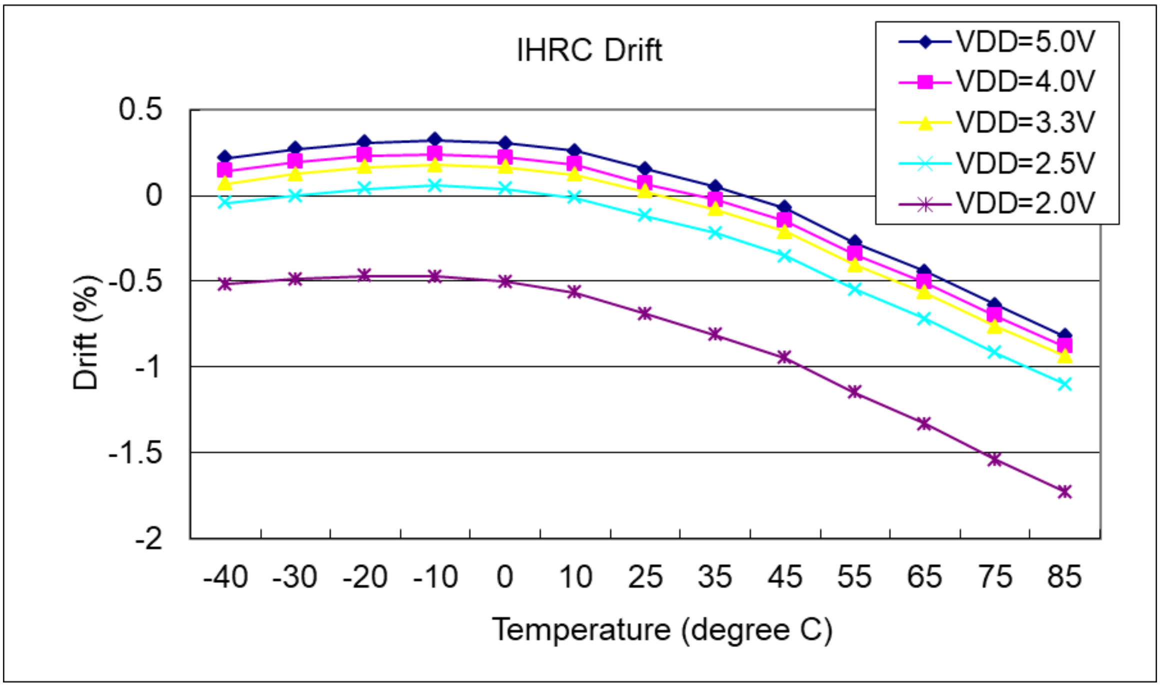
4.4. Typical ILRC Frequency vs. VDD



4.5. Typical ILRC Frequency vs. Temperature



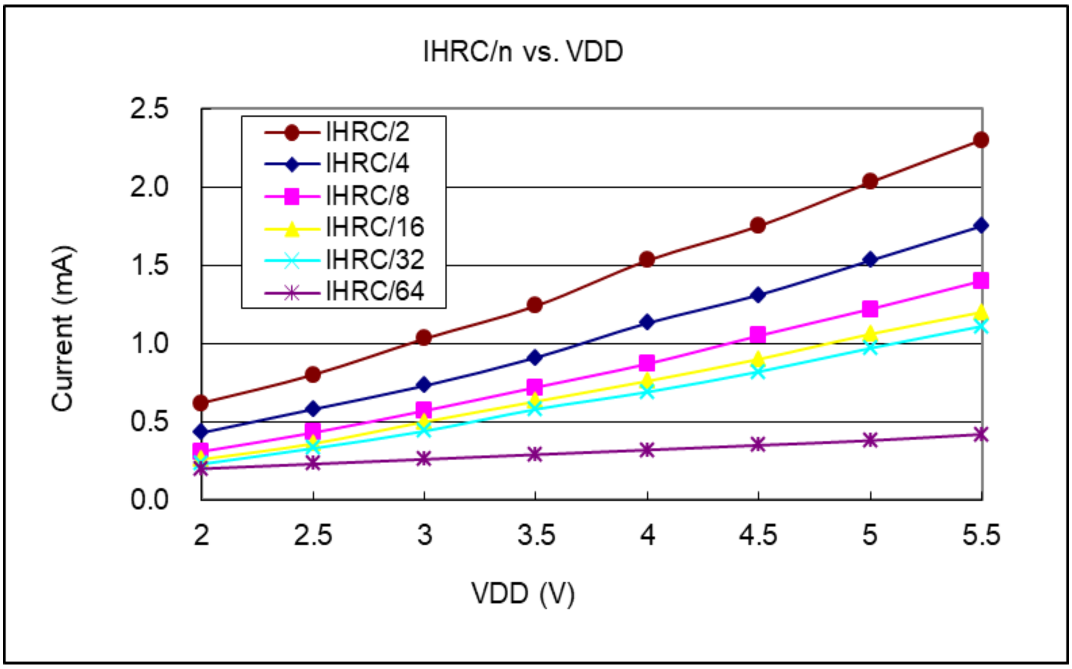
4.6. Typical IHRC Frequency vs. Temperature (calibrated to 16MHz)



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

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

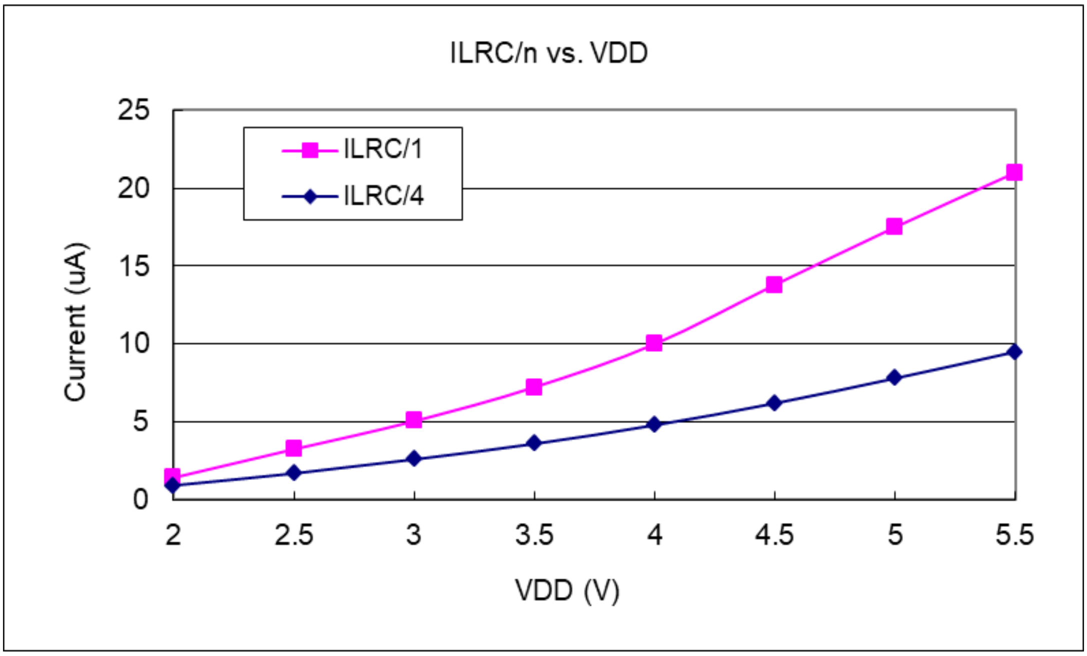
IO: PA0:0.5Hz output toggle and no loading; **others**: input and no floating



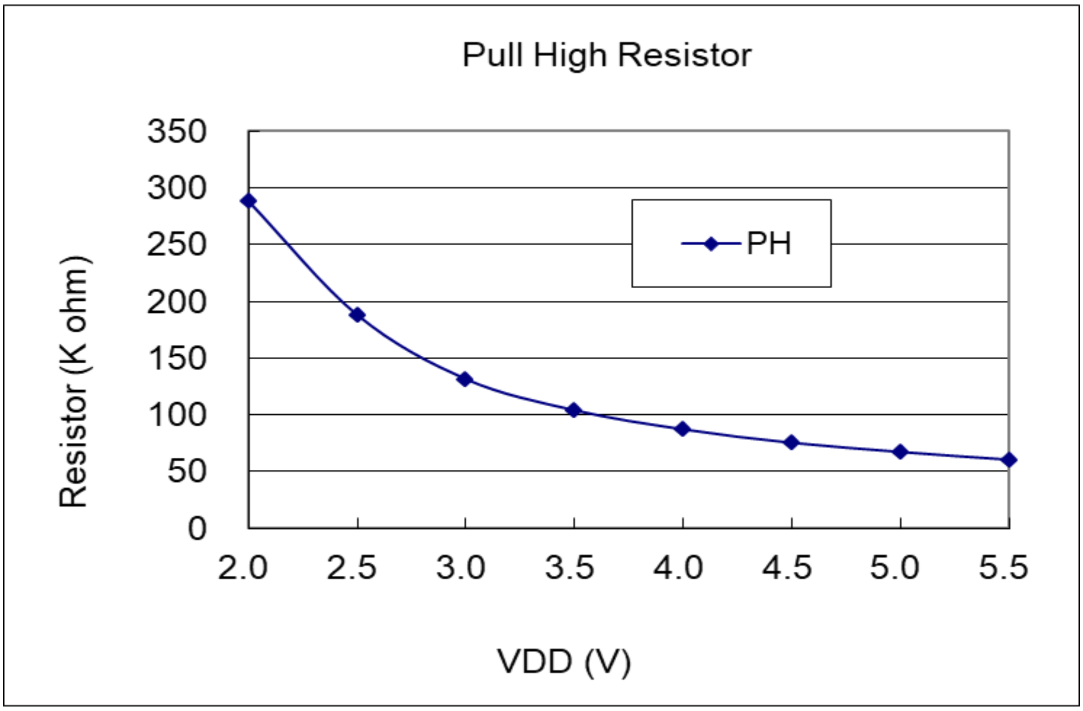
4.8. Typical Operating Current vs. VDD and CLK=ILRC/n

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

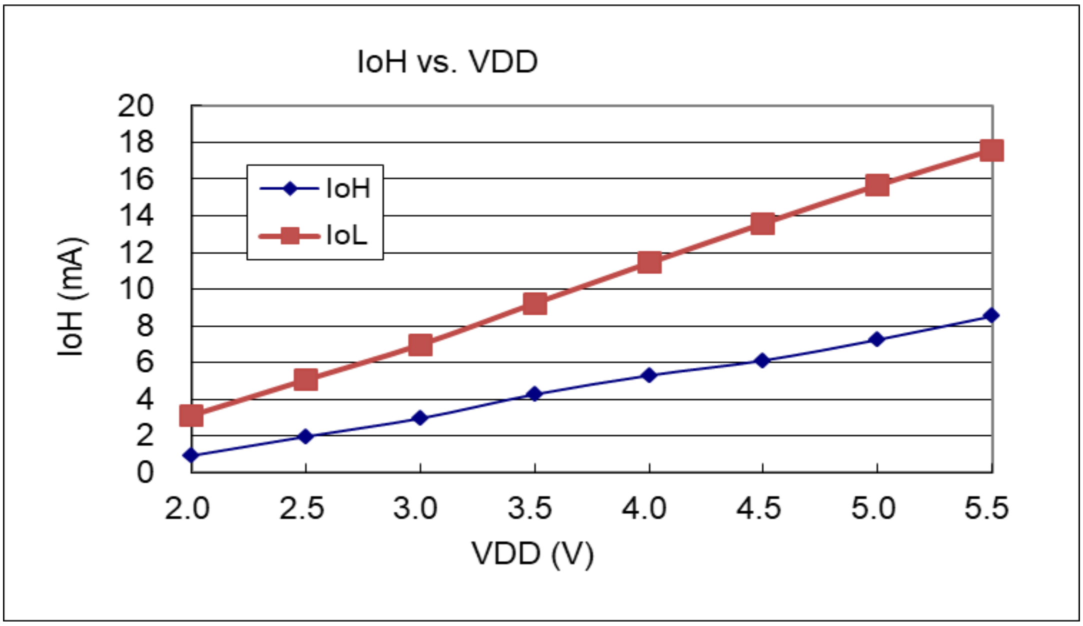
IO: PA0:0.5Hz output toggle and no loading; **others**: input and no floating



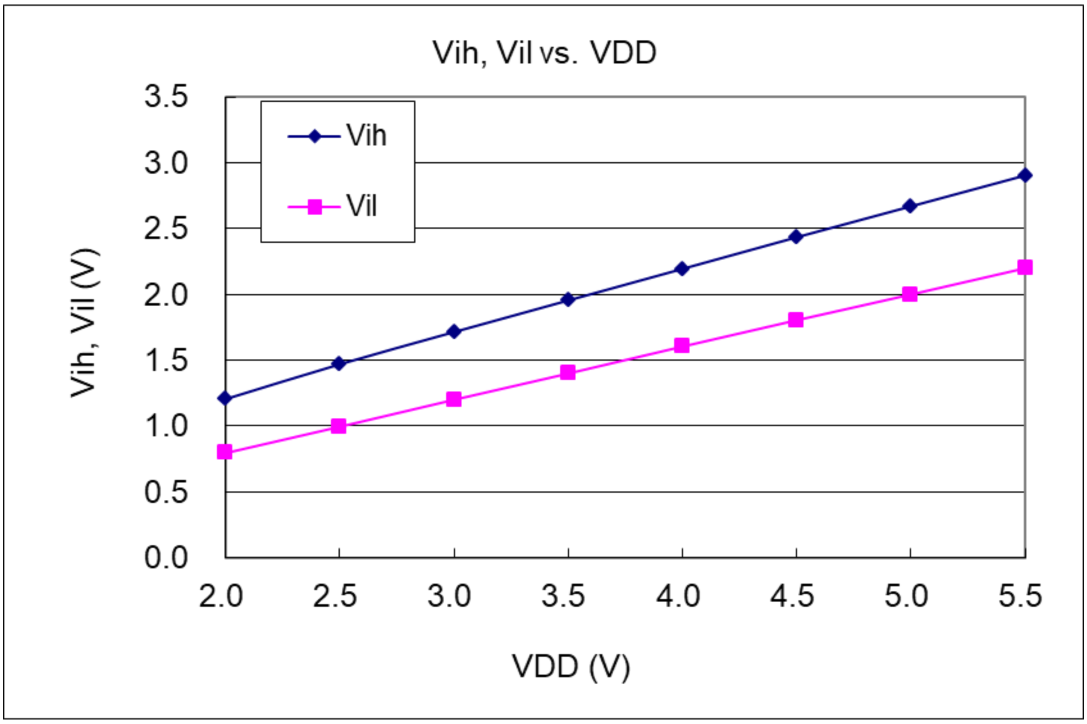
4.9. Typical IO pull high resistance



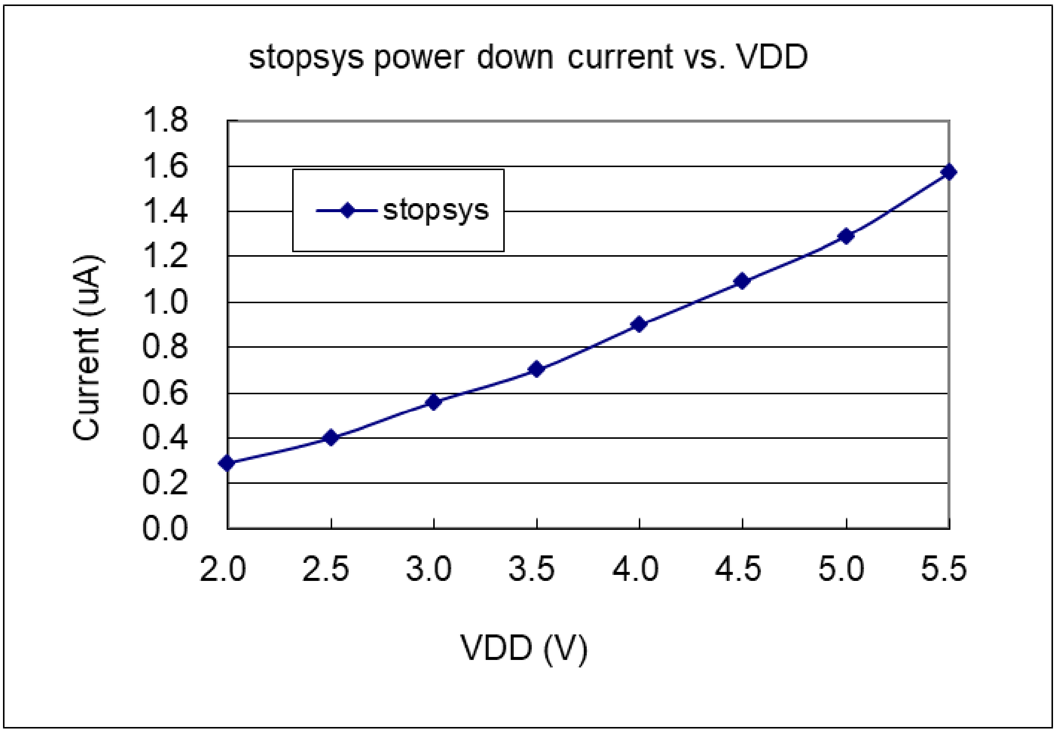
4.10. Typical IO driving current (I_{OH}) and sink current (I_{OL})

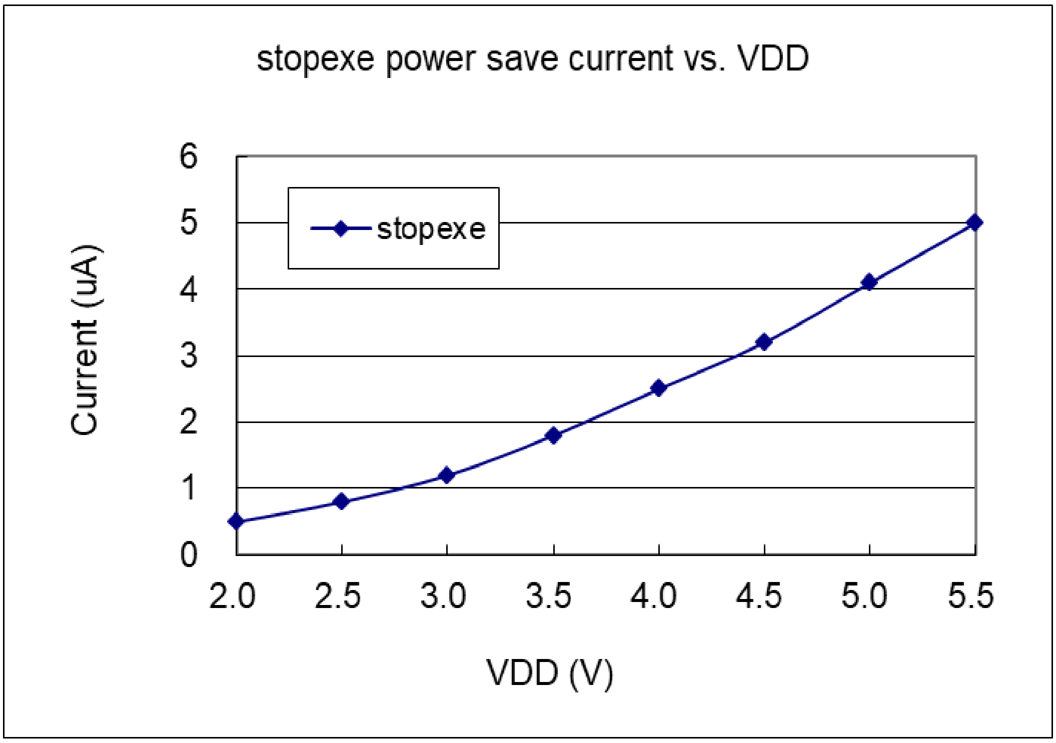


4.11. Typical IO input high / low threshold voltage (V_{IH}/V_{IL})



4.12. Typical power down current (I_{PD}) and power save current (I_{PS})





5. Functional Description

5.1. Program Memory – OTP

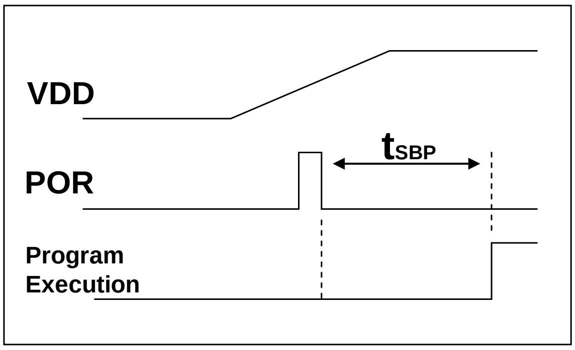
The OTP (One Time Programmable) program memory is used to store the program instructions to be executed. The OTP program memory may contains the data, tables and interrupt entry. After reset, the initial address for FPP0 is 0x000. The interrupt entry is 0x010 if used, the last eight addresses are reserved for system using, like checksum, serial number, etc. The OTP program memory for PMC150/PMS150 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 space.

Address	Function
0x000	FPP0 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

5.2. Boot Up

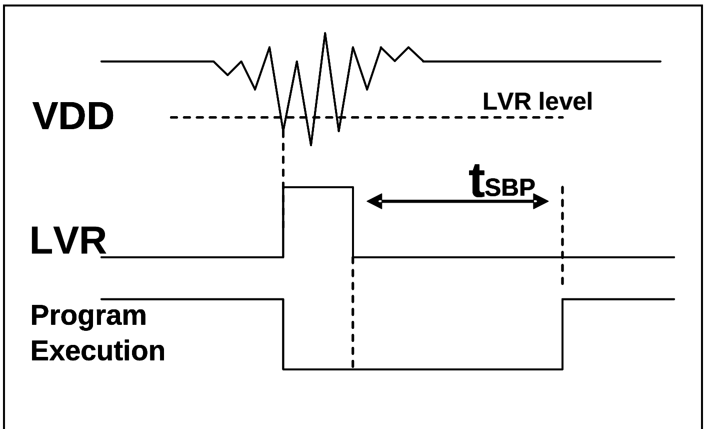
POR (Power-On-Reset) is used to reset PMC150/PMS150 when power up, however, the supply voltage may be not stable. To ensure the stability of supply voltage after power up, it will wait 1024 ILRC clock cycles before first instruction being executed, which is t_{SBP} and shown in the Fig. 1.



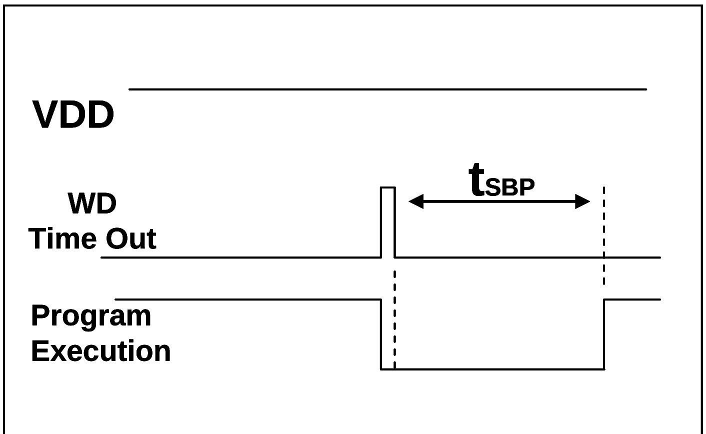
Boot up from Power-On Reset

Fig. 1 Power Up Sequence

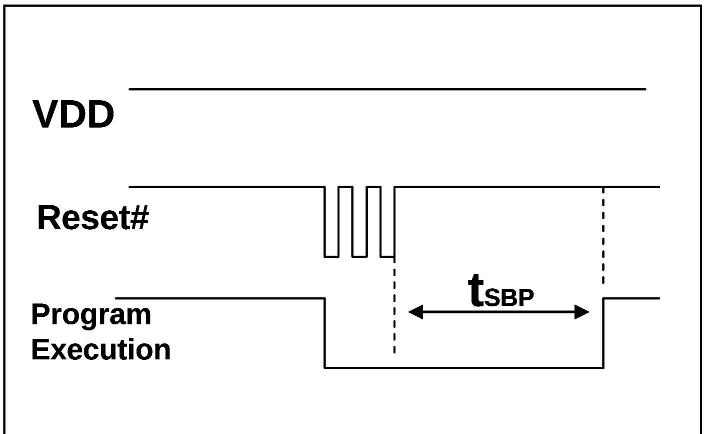
5.2.1. Timing charts for reset conditions



Boot up from LVR detection



Boot up from Watch Dog Time Out



Boot up from Reset Pad reset

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 60 bytes data memory of PMC150/PMS150 can be accessed by indirect access mechanism.

5.4. Oscillator and clock

There are two oscillator circuits provided by PMC150/PMS150: internal high RC oscillator (IHRC) and internal low RC oscillator (ILRC), and these two oscillators are enabled or disabled by registers clkmd.4 and clkmd.2 independently. User can choose one of these two oscillators as system clock source and use ***clkmd*** register to target the desired frequency as system clock to meet different application.

Oscillator Module	Enable/Disable
IHRC	**clkmd**.4
ILRC	**clkmd**.2

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. Please refer to the measurement chart for IHRC frequency verse V_{DD} and IHRC frequency verse temperature.

The frequency of ILRC is around 37KHz, 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, PMC150/PMS150 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=14 ~ 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 2:

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

Table 2: 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 PMC150/PMS150 for different option:

(1) .ADJUST_IC SYSCLK=IHRC/2, IHRC=16MHz, V_{DD}=5V

After boot up, CLKMD = 0x34:

- IHRC frequency is calibrated to 16MHz@V_{DD}=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, V_{DD}=3.3V

After boot, CLKMD = 0x14:

- IHRC frequency is calibrated to 16MHz@V_{DD}=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, V_{DD}=2.5V

After boot, CLKMD = 0x3C:

- IHRC frequency is calibrated to 16MHz@V_{DD}=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, V_{DD}=2.2V

After boot, CLKMD = 0x1C:

- IHRC frequency is calibrated to 16MHz@V_{DD}=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, V_{DD}=5V

After boot, CLKMD = 0x7C:

- IHRC frequency is calibrated to 16MHz@V_{DD}=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, V_{DD}=5V

After boot, CLKMD = 0XE4:

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

(7) **.**ADJUST_IC DISABLE

After boot, CLKMD is not changed (Do nothing):

- IHRC is not calibrated and IHRC module is disabled
- System CLK = ILRC
- Watchdog timer is enabled, ILRC is enabled, PA5 is in input mode

5.4.4. System Clock and LVR levels

The clock source of system clock comes from IHRC or ILRC, the hardware diagram of system clock in the PMC150/PMS150 is shown as Fig. 2.

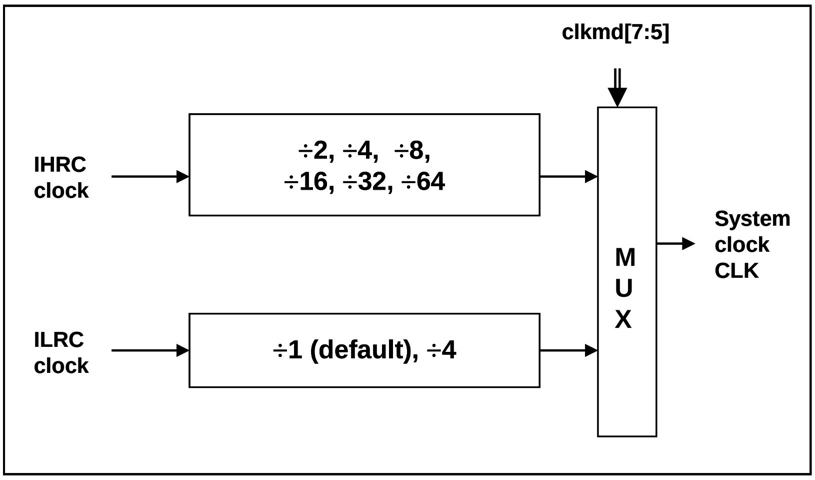


Fig. 2: Options of System Clock

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. Please refer to Section 4.1.

5.5. 16-bit Timer (Timer16)

PMC150/PMS150 provide a 16-bit hardware timer (Timer16) and its clock source may come from system clock (CLK), internal high RC oscillator (IHRC), internal low RC oscillator (ILRC), PA0 or PA4. Before sending clock to the 16-bit counter, a pre-scaling logic with divided-by-1, 4, 16 or 64 is selectable for wide range counting. The 16-bit counter performs up-counting operation only, the counter initial values can be stored from data memory by issuing the ***stt16*** instruction and the counting values can be loaded to data memory by issuing the ***ldt16*** instruction. The interrupt request from Timer16 will be triggered by the selected bit which comes from bit[15:8] of this 16-bit counter, rising edge or falling edge can be optional chosen by register ***integs.4***. The hardware diagram of Timer16 is shown as Fig. 3.

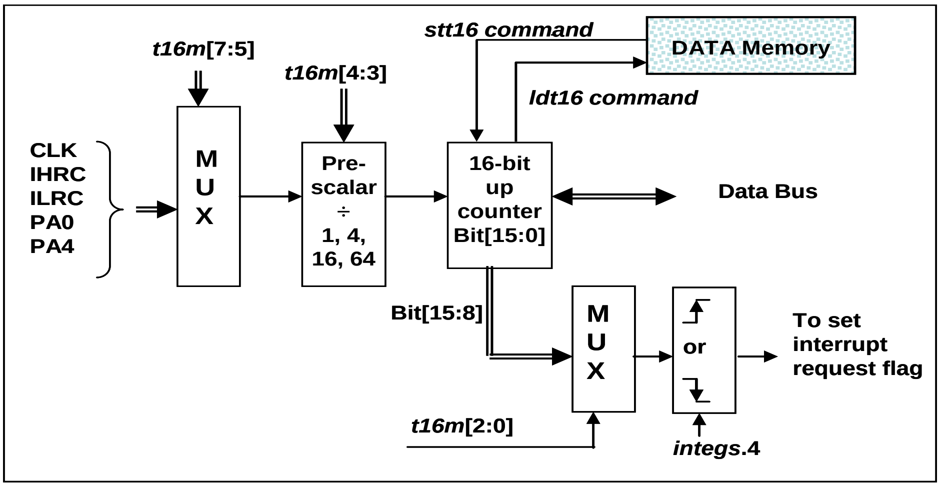


Fig. 3: Hardware diagram of Timer16

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, X, PA4_F, IHRC, X, 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 choose the proper parameters of T16M to meet system requirement, examples as below (For more examples, please refer to IDE software "Application Note → Introduction of IC → Introduction of Register → T16M"):

```
$    T16M     SYSCLK, /64, BIT15;
     // choose (SYSCLK/64) as clock source, every 2^16 clock to set INTRQ.2=1
     // if system clock SYSCLK = IHRC / 2 = 8 MHz
     // SYSCLK/64 = 8 MHz/64 = 8 uS, about every 524 mS to generate INTRQ.2=1

$    T16M     PA0, /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
```

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 four different timeout periods of watchdog timer can be chosen by setting the ***misc*** register, it is:

- 256 ILRC clock period when misc[1:0]=11
- 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, PMC150 will be reset to restart the program execution. The relative timing diagram of watchdog timer is shown as Fig. 4.

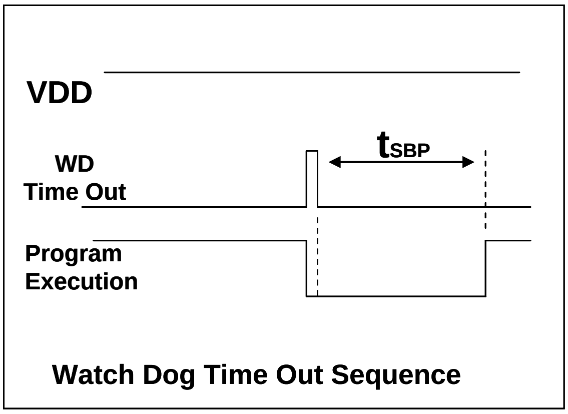


Fig. 4: Sequence of Watch Dog Time Out

5.7. Interrupt

There are two interrupt lines for PMC150/PMS150:

- External interrupt PA0
- 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. 5. 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, the stack position and level are arranged by the compiler in Mini-C project. When defining the stack level in ASM project, users should arrange their locations carefully to prevent address conflicts.

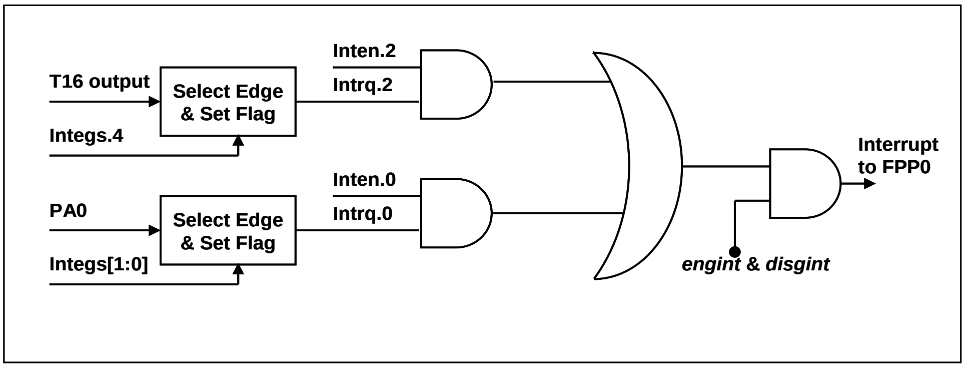


Fig. 5: Hardware diagram of Interrupt controller

Once the interrupt occurs, its operation will be:

- The program counter will be stored automatically to the stack memory specified by register ***sp.***
- New ***sp*** will be updated to ***sp+2.***
- Global interrupt will be disabled automatically.
- The next instruction will be fetched from address 0x010.

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

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

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

- The program counter will be restored automatically from the stack memory specified by register sp.
- New sp will be updated to sp-2.
- Global interrupt will be enabled automatically.
- The next instruction will be the original one before interrupt.

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

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

```
void Interrupt   (void)              // interrupt service routine
{
        PUSHAF                        // store ALU and FLAG register

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

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

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

5.8. 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. Table 3 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
STOPSYS	Stop	Stop
STOPEXE	No Change	No Change

Table 3: 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 be 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 set values when the clock source of Timer16 is IHRC or ILRC 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 shown below:

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

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 PMC150/PMS150 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: IO toggle in digital mode (PxDIER bit is 1)

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

```
CMKMD    =     0xF4;      //    Change clock from IHRC to ILRC, 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 PMC150/PMS150 can be resumed to normal operation by toggling IO pins, Timer16 interrupt is available for Power-Save mode ONLY. Table 4 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

Table 3: Differences in wake-up sources between Power-Save mode and Power-Down mode

When using the IO pins to wake-up the PMC150/PMS150, registers ***padier*** 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 IHRC oscillator stable time from IO toggling if STOPSYS was issued. The oscillator stable time is the time for IHRC oscillator from power-on.

Suspend mode	**Wake-up mode**	**System clock source**	**Wake-up time (t_{WUP}) from IO toggle**
STOPEXE suspend	fast wake-up	Any one	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.
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

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 switch 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***), control registers (***pac***) and pull-high registers (***paph***). All these pins have Schmitt-trigger input buffer and output driver with CMOS level. When it is set to output low, the pull-high 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. 6.

pa.0	***pac.0***	***paph.0***	**Description**
X	0	0	Input without pull-high resistor
X	0	1	Input with pull-high resistor
0	1	X	Output low without pull-high resistor
1	1	0	Output high without pull-high resistor
1	1	1	Output high with pull-high resistor

Table 5: PA0 Configuration Table

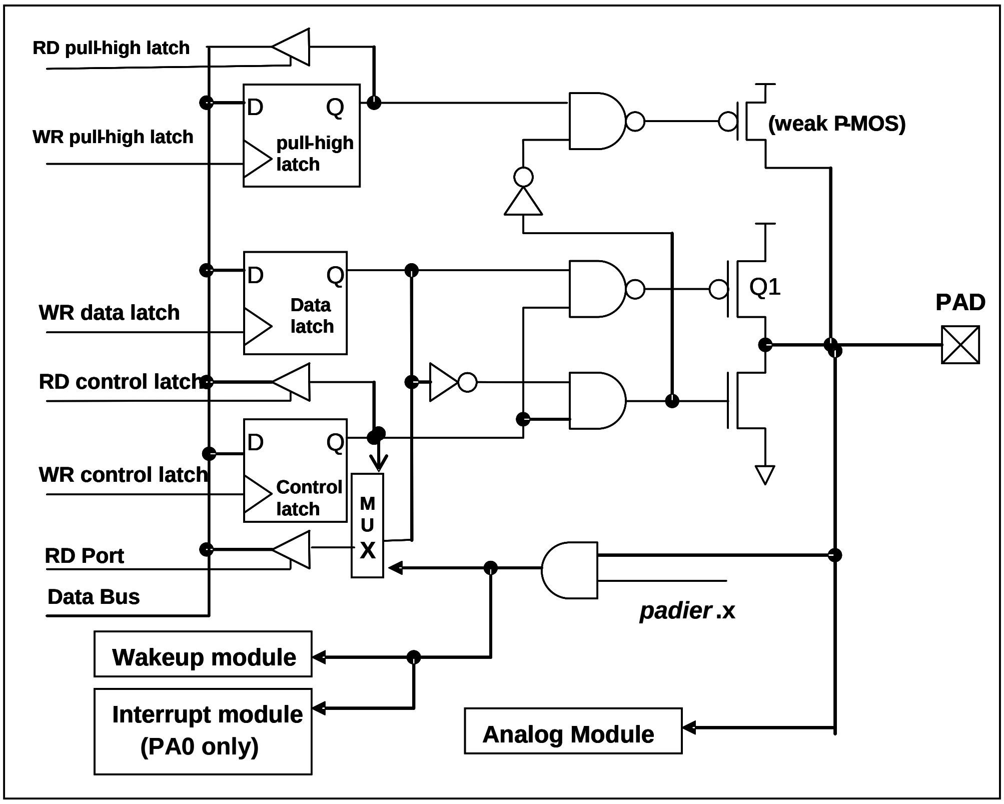


Fig. 6: Hardware diagram of IO buffer

Other than PA5, all the IO pins have the same structure; PA5 can be open-drain ONLY when setting to output mode (without Q1). When PMC150/PMS150 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*** to high**.** The same reason, ***padier***.0 should be set to high when PA0 is used as external interrupt pin.

5.10. Reset and LVR

5.10.1. Reset

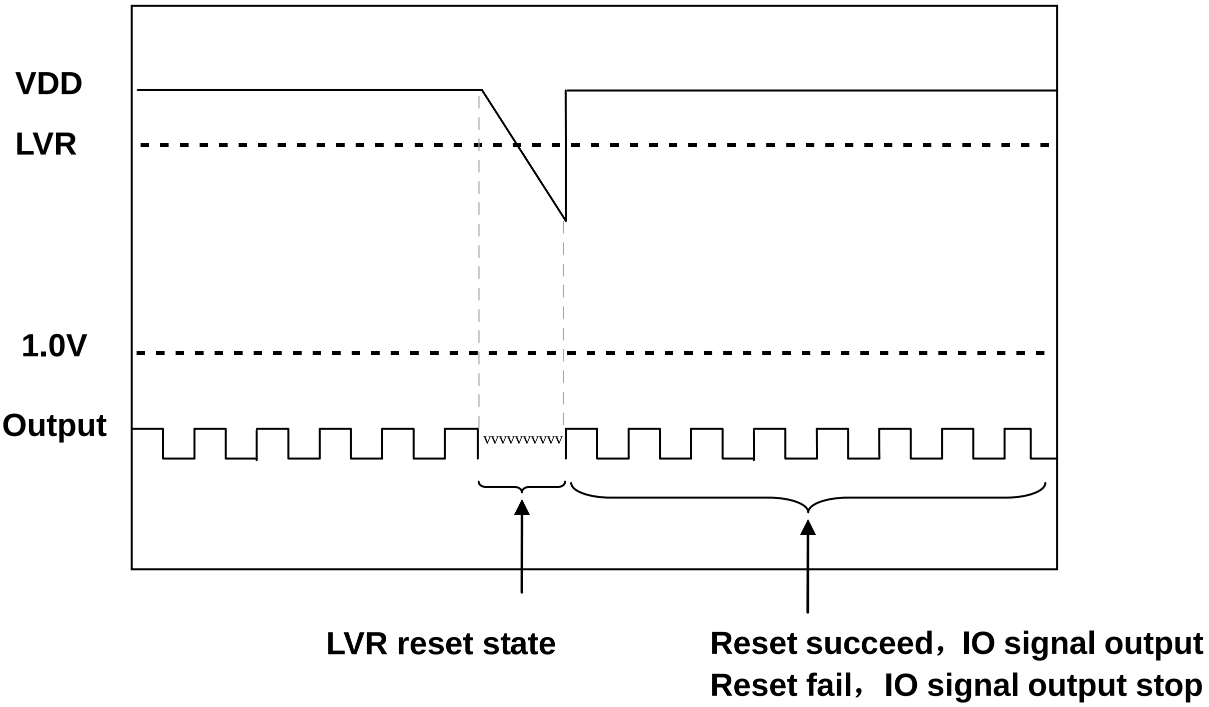
There are many causes to reset the PMC150/PMS150, once reset is asserted, all the registers in PMC150/PMS150 will be set to default values, system should be restarted once abnormal cases happen, or by jumping program counter to address ’h0. The data memory is in uncertain state when reset comes from 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 many different levels of LVR for reset. 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 LVR level but higher than 1.0V, if at this time the power VDD is pulled up again to be over LVR level (just see the diagram below), there may be some chances that cause the MCU malfunction or hanged.


VDD
LVR
1.0V
Output
vvvvvvvvvv
LVR reset state
Reset succeed，IO signal output
Reset fail，IO signal output stop

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)

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

Bit	Reset	R/W	Description	
7 - 5	111	R/W	System clock selection:	
			Type 0, clkmd[3]=0	Type 1, clkmd[3]=1
			000: IHRC/4 001: IHRC/2 01x: reserved 10x: reserved 110: ILRC/4 111: ILRC (default)	000: IHRC/16 001: IHRC/8 010: reserved 011: IHRC/32 100: IHRC/64 1xx: reserved.
4	1	R/W	IHRC oscillator Enable. 0 / 1: disable / enable	
3	0	R/W	Clock Type Select. This bit is used to select the clock type in bit [7:5]. 0 / 1: Type 0 / Type 1	
2	1	R/W	ILRC Enable. 0 / 1: disable / enable If ILRC is disabled, watchdog timer is also disabled.	
1	1	R/W	Watch Dog Enable. 0 / 1: disable / enable	
0	0	R/W	Pin PA5/PRST# function. 0 / 1: PA5 / PRST#	

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

Bit	Reset	R/W	Description
7 - 3	-	-	Reserved
2	0	R/W	Enable interrupt from Timer16 overflow. 0 / 1: disable / enable
1	-	-	Reserved
0	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	-	-	Reserved
2	-	R/W	Interrupt Request from Timer16, this bit is set by hardware and cleared by software. 0 / 1: No request / Request
1	-	-	Reserved
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) 010: reserved 011: PA4 falling edge (from external pin) 100: IHRC 101: reserved 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 - 1	-	-	Reserved. Please keep 0.
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
5 - 0	0	WO	Bit [5:0] 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	-	-	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 - 3	11111	WO	Enable PA7~PA3 wake up event. 1 / 0 : enable / disable. These bits can be set to low to disable wake up from PA7~PA3 toggling. Note: For ICE emulation, the function is disabled when this bit is “1” and “0” is enabled.
2 - 1	-	-	Reserved.
0	1	WO	Enable PA0 wake up event and interrupt request. 1 / 0 : enable / disable. This bit can be set to low to disable wake up from PA0 toggling and interrupt request from this pin. Note: For ICE emulation, the function is disabled when this bit is “1” and “0” is enabled.

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

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

6.12. 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.

6.13. 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-high resistor.

6.14. MISC Register (*misc*), IO address = 0x3b

Bit	Reset	R/W	Description
7 – 6	-	-	Reserved
5	0	WO	Enable fast Wake up. 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) if IHRC is used.
4	-	-	Reserved
3	0	-	Reserved.
2	0	WO	Disable LVR function. 0 / 1 : Enable / Disable
1 – 0	00	WO	Watch dog time out period 00: 2048 ILRC clock period 01: 4096 ILRC clock period 10: 16384 ILRC clock period 11: 256 ILRC clock period

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)
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* pa, a; Result: pa ← 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 ;         // move 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 ; // move 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* pa.5 ; Result: set bit 5 of port A 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 FPP0 Affected flags: 『N』Z 『N』C 『N』AC 『N』OV
disgint	Disable global interrupt enable Example: *disgint* ; Result: Interrupt request is blocked from FPP0 Affected flags: 『N』Z 『N』C 『N』AC 『N』OV
stopsys	System halt. Example: *stopsys*; Result: Stop the system clocks and halt the system Affected flags: 『N』Z 『N』C 『N』AC 『N』OV

stopexe	CPU halt. The oscillator module is still active to output clock, however, system clock is disabled to save power. Example: *stopexe*; Result: Stop the system clocks and keep oscillator modules active. Affected flags: 『N』Z 『N』C 『N』AC 『N』OV
reset	Reset the whole chip, its operation will be same as hardware reset. Example: *reset*; Result: Reset the whole chip. Affected flags: 『N』Z 『N』C 『N』AC 『N』OV
wdreset	Reset Watchdog timer. Example: *wdreset* ; Result: Reset Watchdog timer. Affected flags: 『N』Z 『N』C 『N』AC 『N』OV

7.8. Summary of Instructions Execution Cycle

2T		*goto, call, idxm,* pcadd, ret, reti
2T	Condition is fulfilled	*ceqsn,* cneqsn,*t0sn, t1sn, dzsn, izsn*
1T	Condition is not fulfilled	
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	-	-	-	-										

7.10. BIT definition

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

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

8. Code Options

Option	Selection	Description
Security	Enable	Security Enable
	Disable	Security Disable
LVR	4.0V	Select LVR = 4.0V
	3.5V	Select LVR = 3.5V
	3.0V	Select LVR = 3.0V
	2.75V	Select LVR = 2.75V
	2.5V	Select LVR = 2.5V
	2.2V	Select LVR = 2.2V
	2.0V	Select LVR = 2.0V
	1.8V	Select LVR = 1.8V
Under_20mS_VDD_OK	Yes	reach normal operating voltage quickly within 20 mS
	No	can't reach normal operating voltage quickly within 20 mS

9. Special Notes

This chapter is to remind user who use PMC150/PMS150 series IC in order to avoid frequent errors upon operation.

9.1. Warning

User must read all application notes of the IC by detail before using it. Please download the related application notes from the following link:

http://www.padauk.com.tw/tw/technical/index.aspx

9.2. Using IC

9.2.1. IO pin usage and setting

(1) IO pin as digital input

◆ When IO is set as digital input, the level of Vih and Vil would changes with the voltage and temperature. Please follow the minimum value of Vih and the maximum value of Vil.

◆ The value of internal pull high resistor would also changes with the voltage, temperature and pin voltage. It is not the fixed value.

(2) 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

◆ For those IO pins of PA that are not used, PADIER[1:2] should be set low in order to prevent them from leakage.

(3) PA5 is set to be output pin

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

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

◆ No internal pull-high resistor for PA5

◆ Configure PA5 as input

◆ Set CLKMD.0=1 to enable PA5 as PRST# input pin

(5) 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.

9.2.2. Interrupt

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

Step1: Set INTEN register, enable the interrupt control bit

Step2: Clear INTRQ register

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

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

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

* Use DISGINT in the main program to disable all interrupts

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

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

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

9.2.3. System clock switching

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.

◆ Example : 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
```

◆ **ERROR:** Switch ILRC to IHRC and turn off ILRC simultaneously

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

9.2.4. Power down mode, wakeup and watchdog

(1) Watchdog will be inactive once ILRC is disabled

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

(3) The clock source of Watchdog is ILRC if the fast wakeup is disabled; otherwise, the clock source of Watchdog will be the system clock and the reset time 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
```

9.2.5. TIMER time out

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

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

9.2.6. 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.

9.2.7. IHRC

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

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

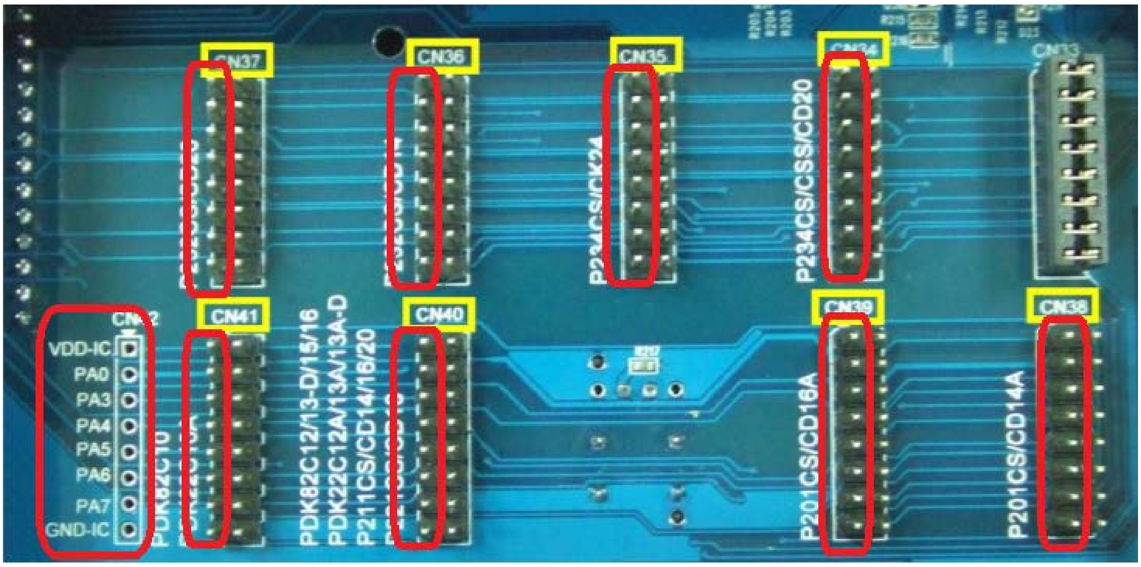
(3) It usually happens in COB package or Quick Turnover Programming (QTP). And PADAUK would not take any responsibility for this situation.

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

9.2.8. Program writing

There are 6 pins for using the writer to program: PA3, PA4, PA5, PA6, VDD and GND.

Please use PDK3S-P-002 for program real chip and just use the CN38 jumper (at the back for the writer) with putting the PMC150/PMS150-S08/DIP8 IC downward three spaces on the Textool . Other packages could be programmed by connecting the signals correspondingly. All the signals of the left side of the jumpers are the same and as the descriptions at the left bottom corner. They are VDD, PA0(not used), PA3, PA4, PA5, PA6, PA7(not used), and GND).



If user use PDK5S-P-003 or above to program, please follow the instructions for connecting jumpers.

- Special notes about voltage and current while Multi-Chip-Package(MCP) or On-Board Programming
  (1) PA5 (V_{PP}) may be higher than 11V.
  (2) V_{DD} may be higher than 6.5V, and its maximum current may reach about 20mA.
  (3) All other signal pins level (except GND) are the same as V_{DD}..
  User should confirm when using this product in MCP or On-Board Programming, the peripheral circuit or components will not be destroyed or limit the above voltages.

9.3. Using ICE

Please use PDK5S-I-S01/2(B) ICE to emulate PMC150/PMS150. Please note in the simulation:

(1) Fast Wake-up time is different from PDK5S-I-S01/2(B): 128 SYSCLK, PMC150/PMS150: refer to 5.8.3

(2) Watch dog time out period is different from PDK5S-I-S01/2(B):

WDT period	PMC150/PMS150	PDK5S-I-S01/2(B)
misc[1:0]=00	8K* T_{ILRC}	2048* T_{ILRC}
misc[1:0]=01	16K* T_{ILRC}	4096* T_{ILRC}
misc[1:0]=10	64K* T_{ILRC}	16384* T_{ILRC}
misc[1:0]=11	256K* T_{ILRC}	256* T_{ILRC}