PMS160B

6 Touch Keys OTP Controller

Datasheet

Version 0.01 – November 12, 2025

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.

Any programming software provided by PADAUK Technology to customers is of a service and reference nature and does not have any responsibility for software vulnerabilities. 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
0.00	2025/09/15	Preliminary version
0.01	2025/11/12	Add Chapter 9.2 Using ICE

Usage Warning

- There can be no overvoltage input (greater than the chip VDD voltage) at all IO pins of the chip, which will cause interference to the touch and cause abnormal touch.
- User must read all application notes of the IC by detail before using it.

  Please visit the official website to download and view the latest APN information associated with it.

  https://www.padauk.com.tw/en/product/show.aspx?num=259&kw=PMS160B

◆◆ PMS160B ◆◆

- Operating temperature : -40°C ~ 85°C

Feature | Documents | Software &Tools | Application Note

Content	Description	Download (CN)	Download (EN)
APN002	Over voltage protection		
APN003	Over voltage protection		
APN004	Semi-Automatic writing handler		
APN007	Setting up LVR level		
APN011	Semi-Automatic writing Handler improve writing stability		
APN019	E-PAD PCB layout guideline		
APN023	Self-Capacitance Touch PCB Design Guide		

1. Features

1.1. Special Features

- In applications with AC RC step-down power supply or high EFT requirements, the system circuit needs to be modified if necessary to improve the anti-interference capability
- Operating temperature range: -40°C ~ 85°C

1.2. System Features

- 1.5KW OTP program memory
- 128 Bytes data RAM
- Maximum 6 IO pins can be selected as TOUCH PAD individually, each featuring noise immunity
- One hardware 16-bit timer
- Two hardware 8-bit timers with PWM generation (Timer2/Timer3), Timer2/Timer3 also configured with NILRC oscillator. Its frequency is slower than ILRC and suitable for a more power-saving wake-up clock.
- One set triple 11bit SuLED (Super LED) PWM generators and timers (LPWMG0/LPWMG1/LPWMG2)
- One hardware comparator
- 6 IO pins with optional pull-high/pull-low resistor
- Bandgap circuit to provide 1.2V Bandgap voltage
- Clock sources: internal high RC oscillator and internal low RC oscillator
- 14 Levels of LVR reset: 4.5V, 4.0V, 3.75V, 3.5V, 3.3V, 3.15V, 3.0V, 2.7V, 2.5V, 2.4V, 2.3V, 2.2V, 2.1V, 2.0V
- Two selectable external interrupt pins
- Internal LDO to prevent touch noise
- Support low-power wake-up ‘stopsys’ by NILRC
- IOH: 18 mA , IOL: 25 mA

1.3. CPU Features

- Operating modes: One processing unit mode
- 82 powerful instructions
- Most instructions are 1T execution cycle
- Programmable stack pointer to provide adjustable stack level (Using 2 bytes SRAM for one stack level)
- Direct and indirect addressing modes for data and instructions
- All data memories are available for use as an index pointer
- Separated IO and memory space

1.4. Ordering/ Package Information

- PMS160B-S08A: SOP8 (150mil)
- PMS160B-S08B: SOP8 (150mil)
- PMS160B-2N08A: DFN8(2*2mm)
- PMS160B-2N08B: DFN8(2*2mm)
- PMS160B-2N06A: DFN6(2*2mm)
- PMS160B-U06A: SOT23-6 (60mil)

- Please refer to the official website file for package size information：”Package information”.

2. General Description and Block Diagram

The PMS160B is a fully static, OTP-based 8-bit CMOS MCU; 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.

A maximum 6 keys touch controller is built inside PMS160B. Besides, PMS160B also includes 1.5KW OTP program memory, 128 bytes data SRAM, one hardware 16-bit timer, one hardware 8-bit Timer3, one hardware 8-bit Timer2 and one new triple 11-bit timer with SuLED PWM generation (LPWMG0/1/2).

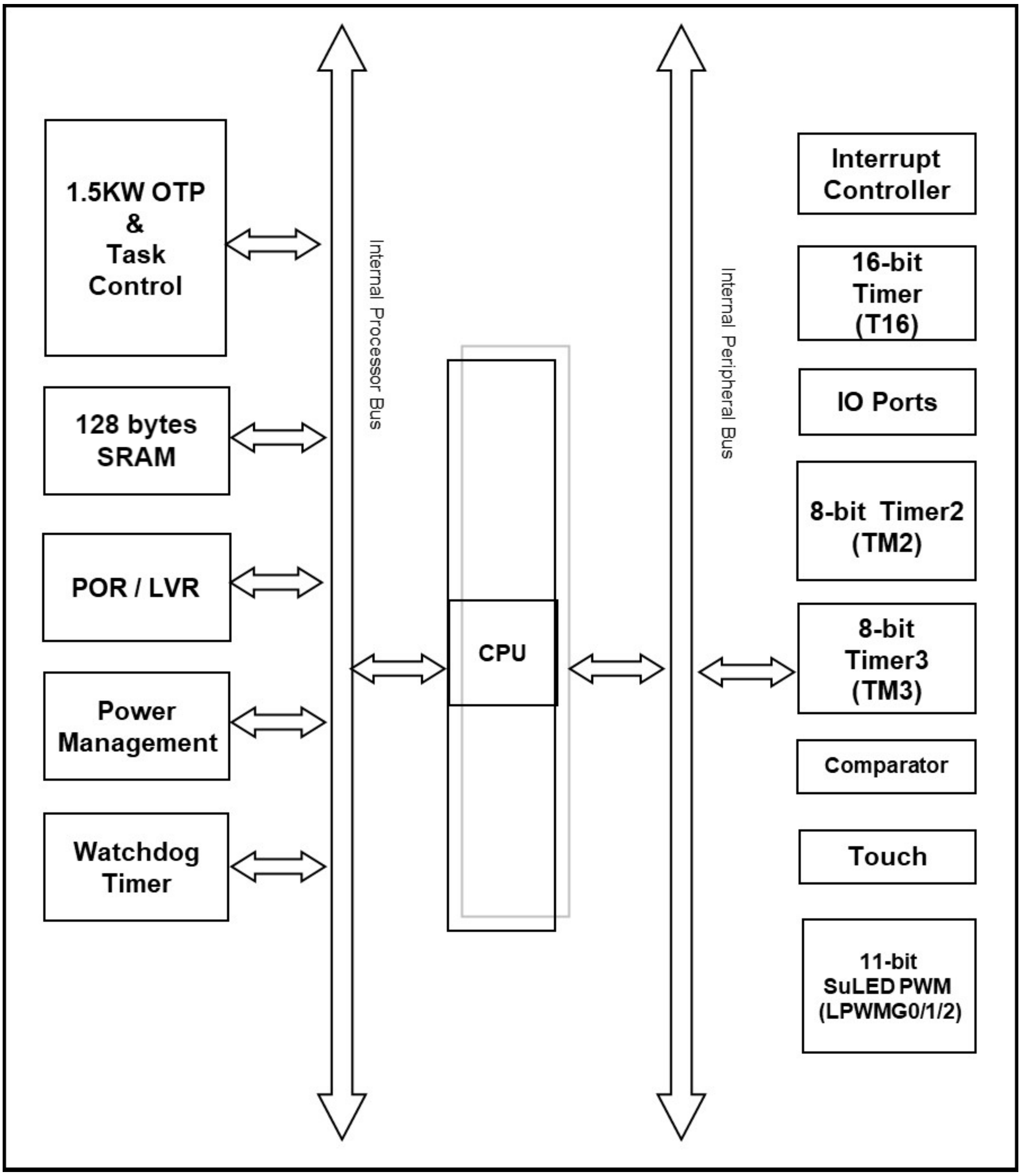
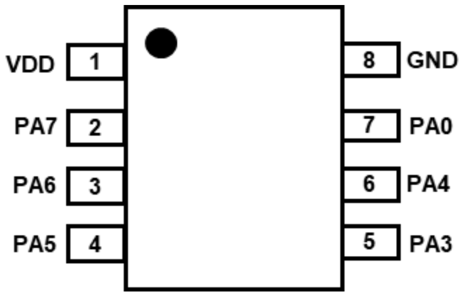
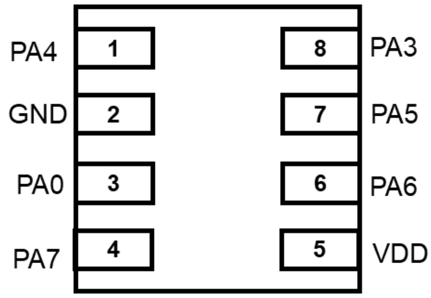


Fig. 1: PMS160B Block Diagram

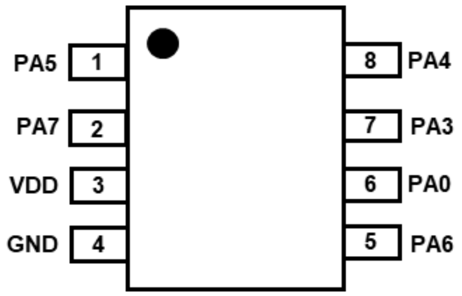
3. Pin Definition and Functional Description



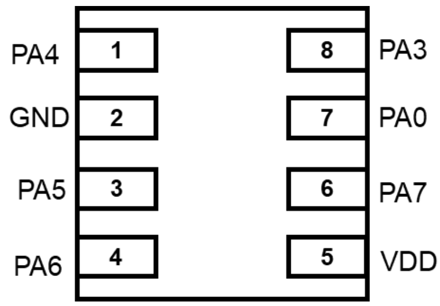
PMS160B-S08A: SOP8 (150mil)



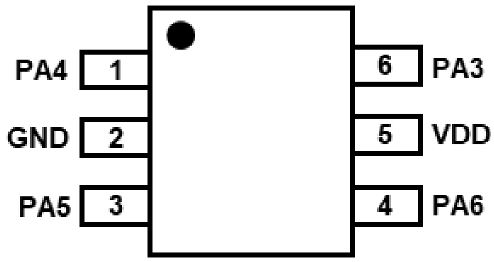
PMS160B-2N08A: DFN8 (2*2mm)



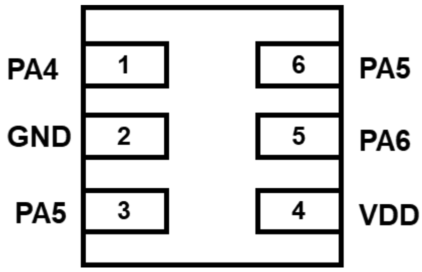
PMS160B-S08B: SOP8 (150mil)



PMS160B-2N08B: DFN8 (2*2mm)



PMS160B-U06A: SOT23-6 (60mil)



PMS160B-2N06A: DFN6 (2*2mm)

Pin Name	Pin & Buffer Type	Description
PA6 / TK4 / CIN-	IO ST / CMOS / Analog	This pin can be used as: (1) Bit 6 of port A. It can be configured as digital input or two-state output, with pull-high/pull-low resistor independently by software. (2) Touch Key 4 (3) Minus input source of comparator. When this pin is configured as analog input, please use bit 6 of register ***padier*** to disable the digital input to prevent leakage current.
PA5 / TK3 / INT0 / PRSTB / VPP	IO ST / CMOS	This pin can be used as: (1) Bit 5 of port A. It can be configured as digital input or two-state output, with pull-high/pull-low resistor independently by software. (2) Touch Key 3 (3) Optional external interrupt line 0. Both rising edge and falling edge are accepted to request interrupt service. (4) External reset pin (5) VPP for OTP programming 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 / TK2 / PWM1 / LPWM1 / CIN+ / CIN-	IO ST / CMOS / Analog	This pin can be used as: (1) Bit 4 of port A. It can be configured as digital input or two-state output, with pull-high/pull-low resistor independently by software. (2) Touch Key 2 (3) PWM output of Timer2 (4) Channel 1 output of LPWM (5) Positive input source of comparator. (6) Minus input source of comparator. When this pin is configured as analog input, please use bit 4 of register ***padier*** to disable the digital input to prevent leakage current.

Pin Name	Pin & Buffer Type	Description
PA3 / TK1 / PWM1 / LPWM0 / CIN-	IO ST / CMOS / Analog	This pin can be used as: (1) Bit 3 of port A. It can be configured as digital input or two-state output, with pull-high/pull-low resistor independently by software. (2) Touch Key 1 (3) PWM output of Timer2 (4) Channel 0 output of LPWM (5) Minus input source of comparator. When this pin is configured as analog input, please use bit 3 of register ***padier*** to disable the digital input to prevent leakage current.
PA0 / TK0 / INT0 / LPWM2 / CMPO	IO ST / CMOS / Analog	This pin can be used as: (1) Bit 0 of port A. It can be configured as digital input or two-state output, with pull-high/pull-low resistor independently by software. (2) Touch Key 0 (3) Optional external interrupt line 0. Both rising edge and falling edge are accepted to request interrupt service. (4) Channel 2 output of LPWM (5) Output result of comparator. When this pin is configured as analog input, please use bit 0 of register ***padier*** to disable the digital input to current leakage current.
PA7 / TK5 / LPWM2	IO ST / CMOS	This pin can be used as: (1) Bit 7 of port A. It can be configured as digital input or two-state output, with pull-high/pull-low resistor independently by software. (2) Touch Key 5 (3) Channel 2 output of LPWM When this pin is configured as analog input, the input function of this pin is disabled to prevent leakage current regardless of the setting of the bit 7 of register ***padier.***
VDD	VDD	VDD: Digital positive power
GND	GND	GND: Digital negative power
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	2.4#		5.5	V	#Subject to LVR tolerance VDD ≧ 2.4V when Touch conversion
LVR%	Low Voltage Reset Tolerance	-5		5	%	
f_{SYS}	System clock (CLK)* = IHRC/2 IHRC/4 IHRC/8 ILRC	 0 0 0	 46K	 8M 4M 2M	Hz	 V_{DD} ≧ 3.0V，No Touch V_{DD} ≧ 2.5V，No Touch V_{DD} ≧ 2.0V，No Touch V_{DD} = 5V
V_{POR}	Power on reset		1.8		V	
I_{OP}	Operating Current		0.5 40		mA uA	f_{SYS}=IHRC/16=1MIPS@5.0V f_{SYS}=ILRC=45KHz@5.0V
I_{PD}	Power Down Current (by *stopsys* command)		0.6 0.1		uA	V_{DD} = 5V V_{DD} = 3.3V
I_{PS}	Power Save Current (by *stopexe* command) *Disable IHRC		2.3		uA	V_{DD} = 5V
V_{IL}	Input low voltage for IO lines	0		0.1 V_{DD}	V	
V_{IH}	Input high voltage for IO lines	0.7 V_{DD}		V_{DD}	V	
I_{OL}	IO lines sink current (Normal)		25		mA	V_{DD} = 5.0V, V_{OL} = 0.5V
I_{OH}	IO lines drive current (Normal)		18		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		uA	V_{DD} +0.3 ≧ V_{IN} ≧ -0.3
R_{PH}	Pull-high Resistance		75		KΩ	V_{DD} = 5.0V
R_{PL}	Pull-low Resistance		68		KΩ	V_{DD} = 5.0V
f_{IHRC}	Frequency of IHRC after calibration *	15.76*	16*	16.24*	MHz	25°C, VDD = 2.2V~5.5V
		15.20*	16*	16.80*		VDD = 2.2V~5.5V, -40°C <Ta<85°C*
		14.60*	16*	17.40*		VDD = 2.0V~5.5V, -40°C <Ta<85°C
f_{ILRC}	Frequency of ILRC *		45		KHz	VDD = 5V
f_{NILRC}	Frequency of NILRC *		18		KHz	VDD = 5.0V
t_{INT}	Interrupt pulse width	30			ns	VDD = 5.0V
t_{WDT}	Watchdog timeout period		8192		ILRC clock period	misc[1:0]=00 (default)
			16384			misc[1:0]=01
			65536			misc[1:0]=10
			262144			misc[1:0]=11

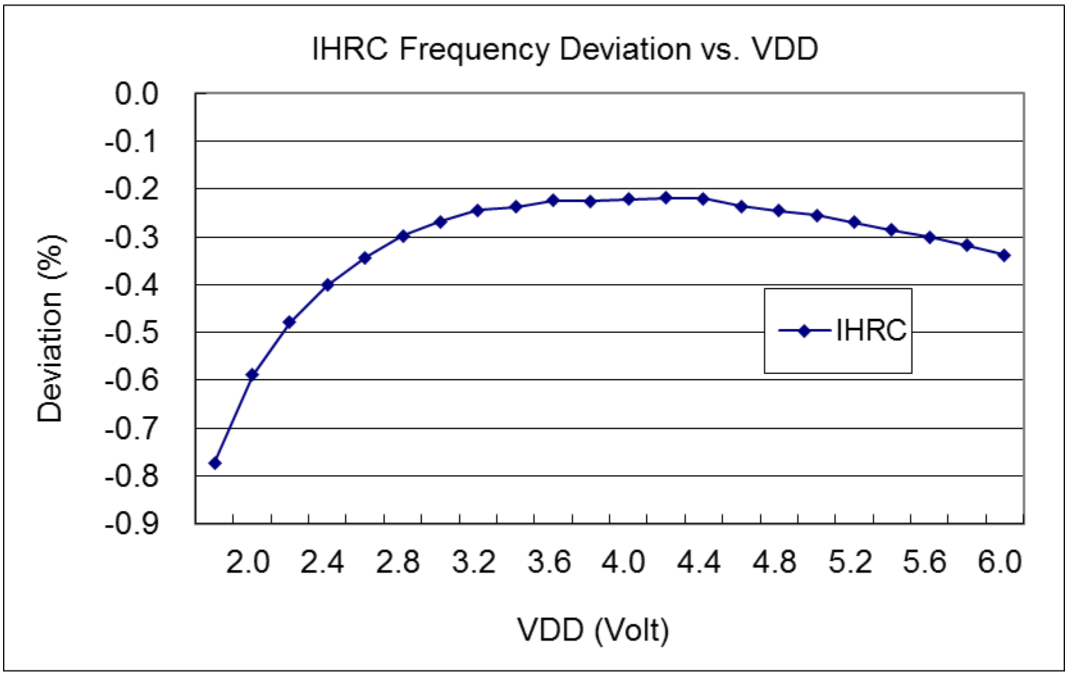
Symbol	Description	Min.	Typ	Max.	Unit	Conditions
t_{WUP}	Wake-up time period for fast wake-up		8		T_{ILRC}	Where T_{ILRC} is the time period of ILRC
	Wake-up time period for normal wake-up		16			
t_{SBP}	System boot-up period from power-on for Normal boot-up		43		ms	@ V_{DD} = 5V
t_{RST}	External reset pulse width	120			us	@ V_{DD} = 5V

*These parameters are for design reference, not tested for every chip.
*The characteristic diagrams are the actual measured values. Considering the influence of production drift and other factors, the data in the table are within the safety range of the actual measured values.

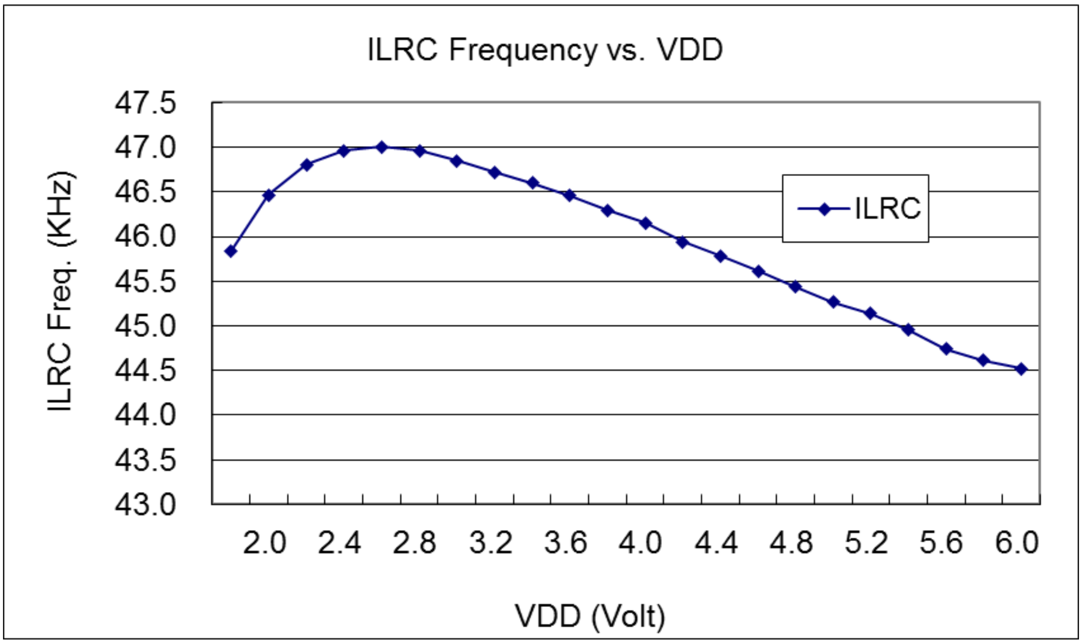
4.2. Absolute Maximum Ratings

Parameter	Maximum Rating	Notes
Supply Voltage	2.0V ~ 5.5V (Maximum Rating: 5.5V)	If VDD is over the maximum rating, it may lead to a permanent damage of IC
Input Voltage	-0.3V ~ VDD + 0.3V	
Operating Temperature	-20°C ~ 70°C	
Storage Temperature	-50°C ~ 125°C	
Junction Temperature	150°C	

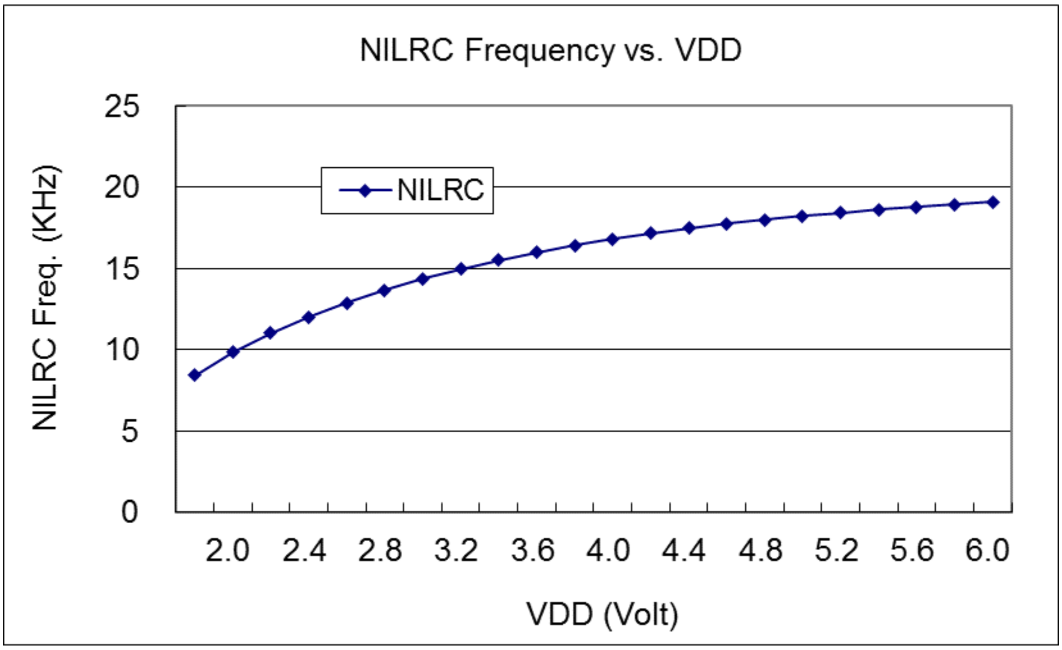
4.3. Typical IHRC Frequency vs. VDD (calibrated to 16MHz)



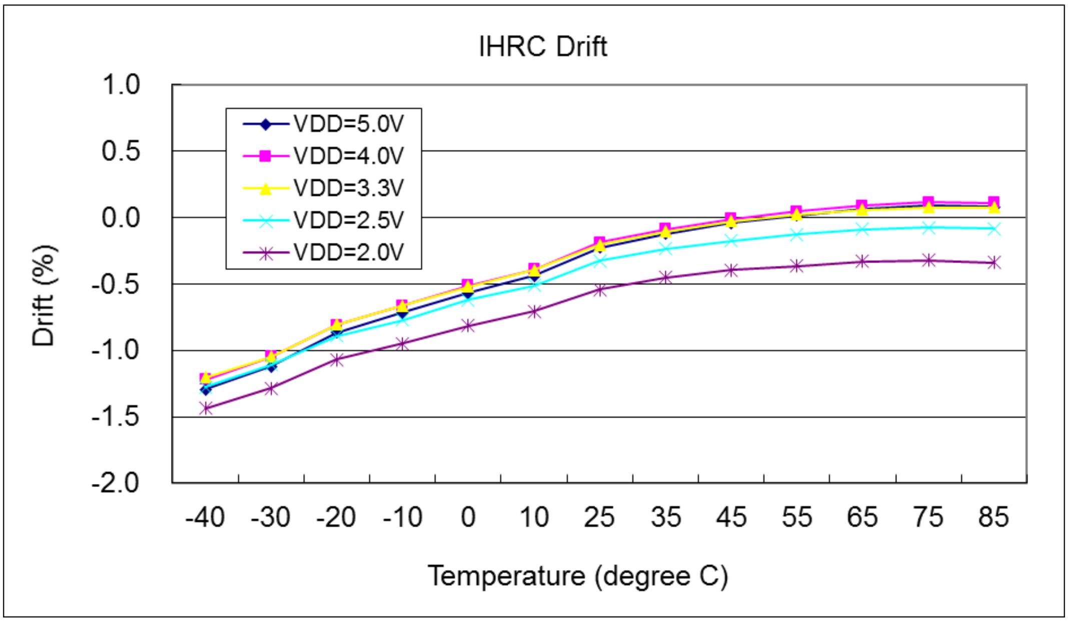
4.4. Typical ILRC Frequency vs. VDD



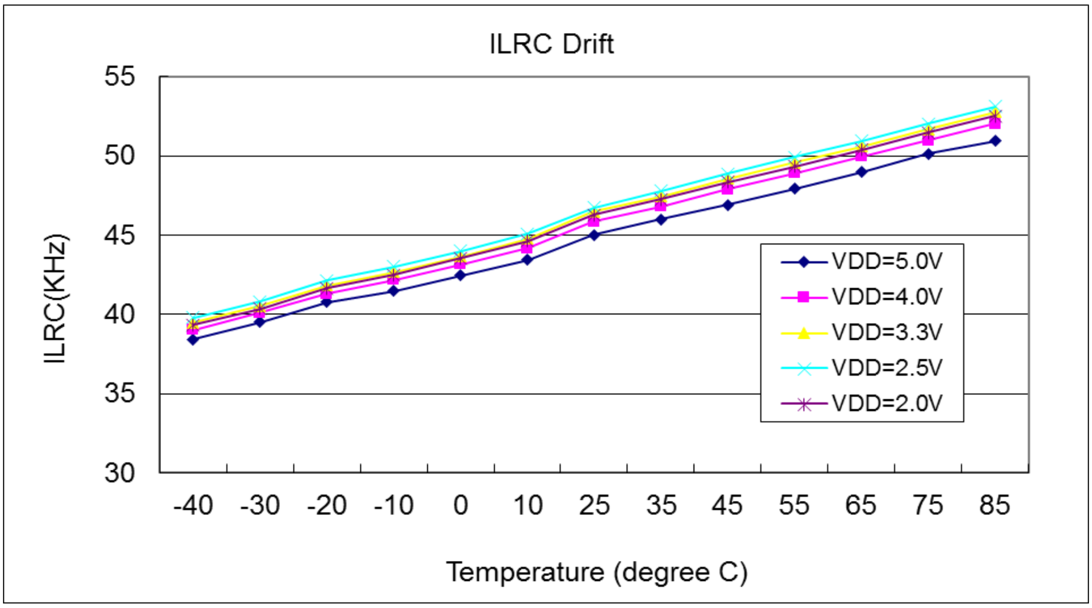
4.5. Typical NILRC Frequency vs. VDD



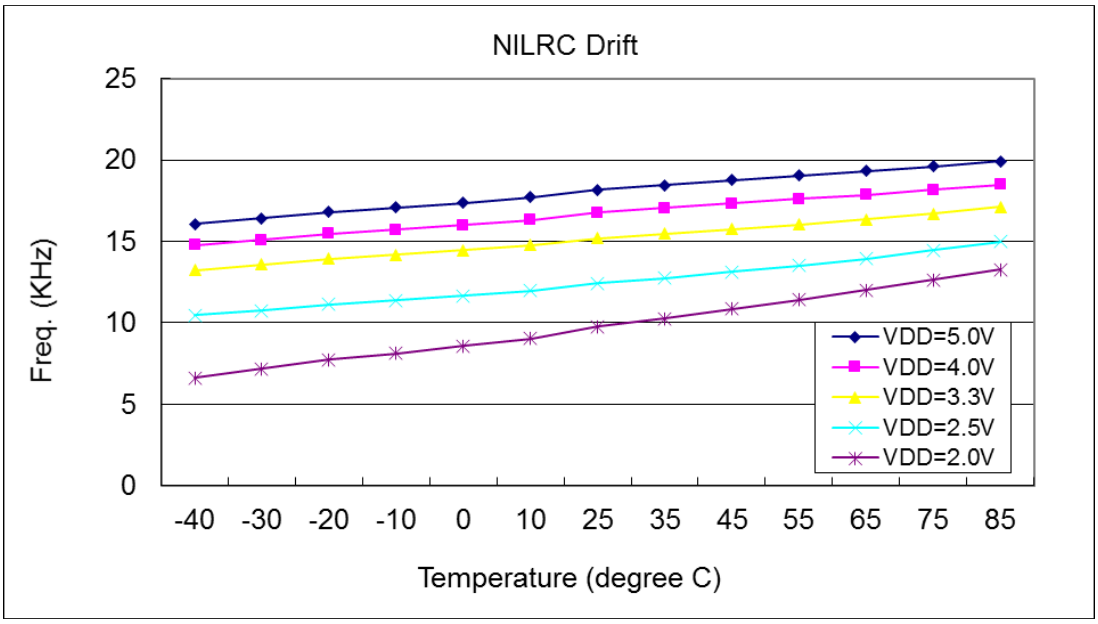
4.6. Typical IHRC Frequency vs. Temperature (calibrated to 16MHz)



4.7. Typical ILRC Frequency vs. Temperature

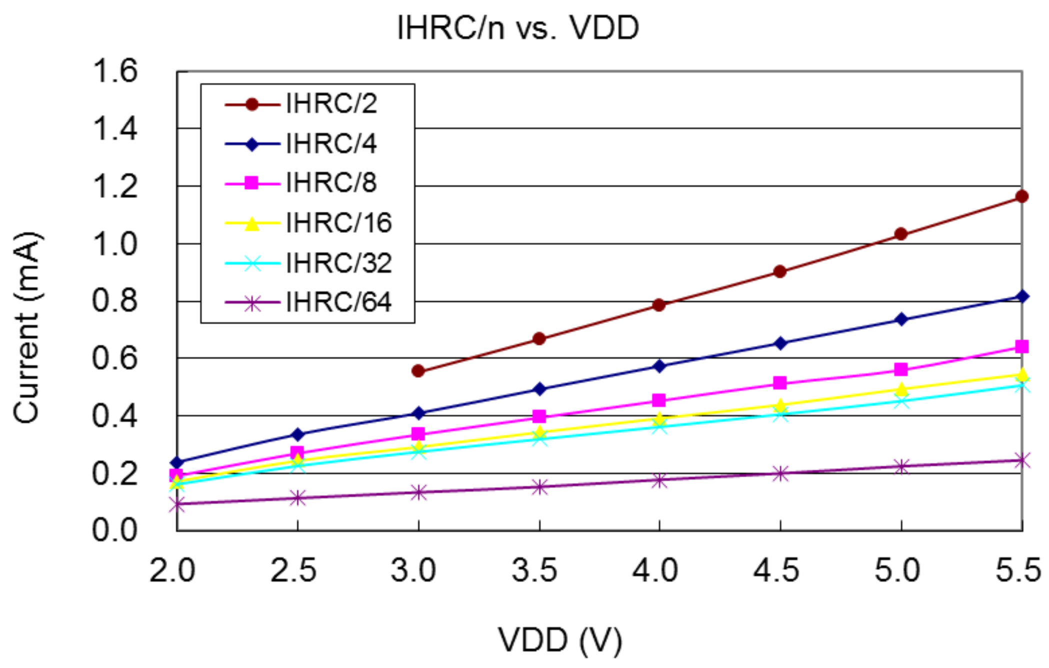


4.8. Typical NILRC Frequency vs. Temperature



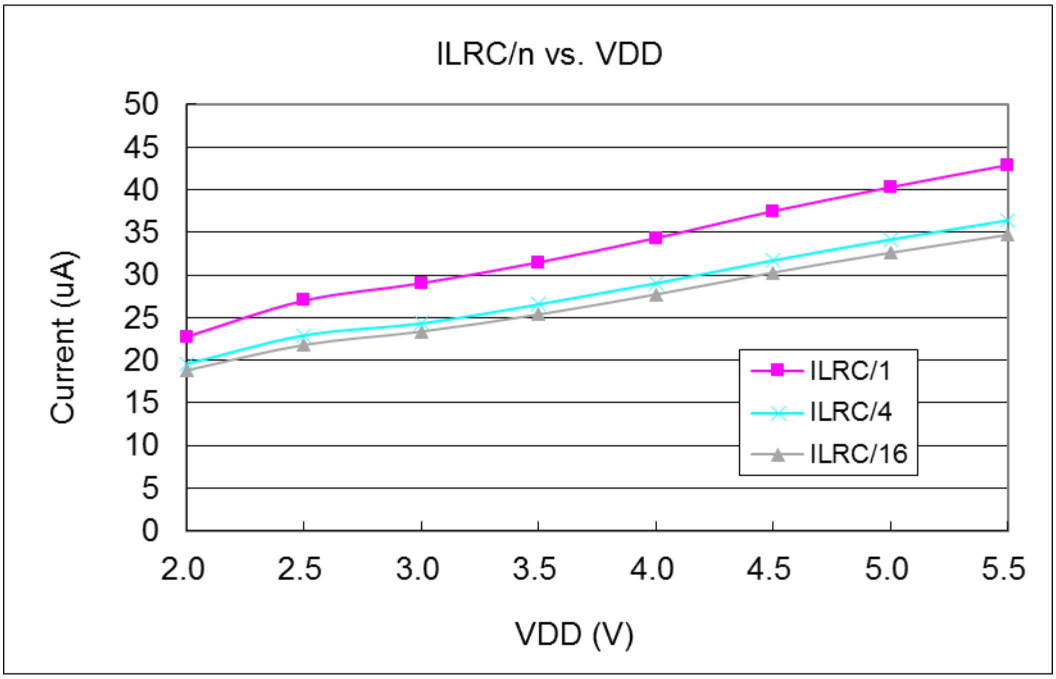
4.9. Typical Operating Current vs. VDD and CLK=IHRC/n

- Conditions:
  tog pa0(1s), **ON**: Bandgap, LVR, IHRC
  no t16m, no interrupt, no floating IO pins, disable ILRC, **Touch**: Disable

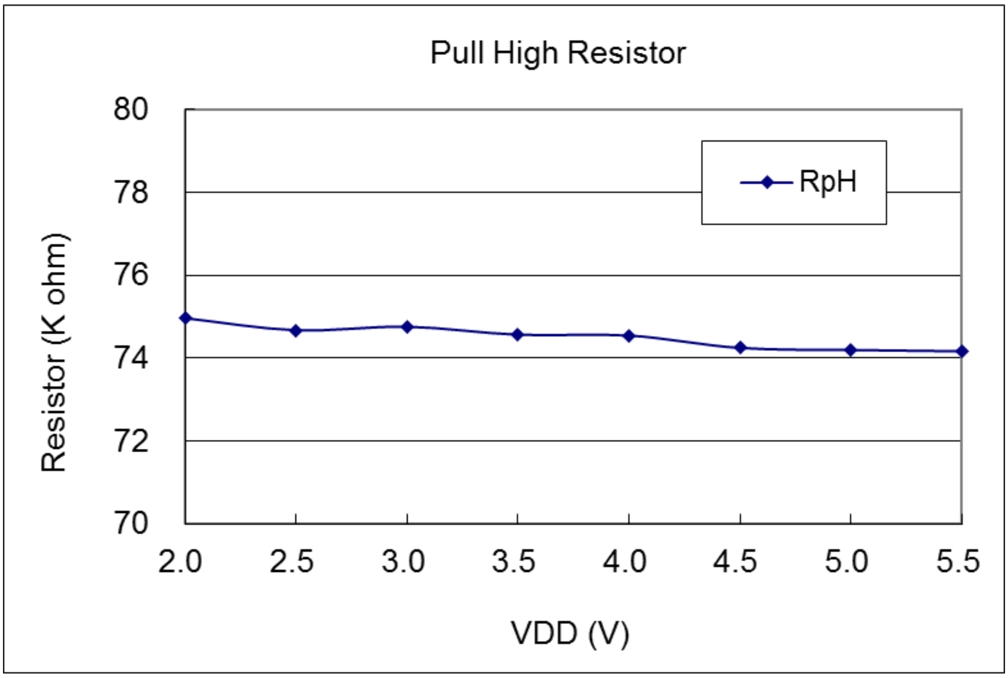


4.10. Typical Operating Current vs. VDD and CLK=ILRC/n

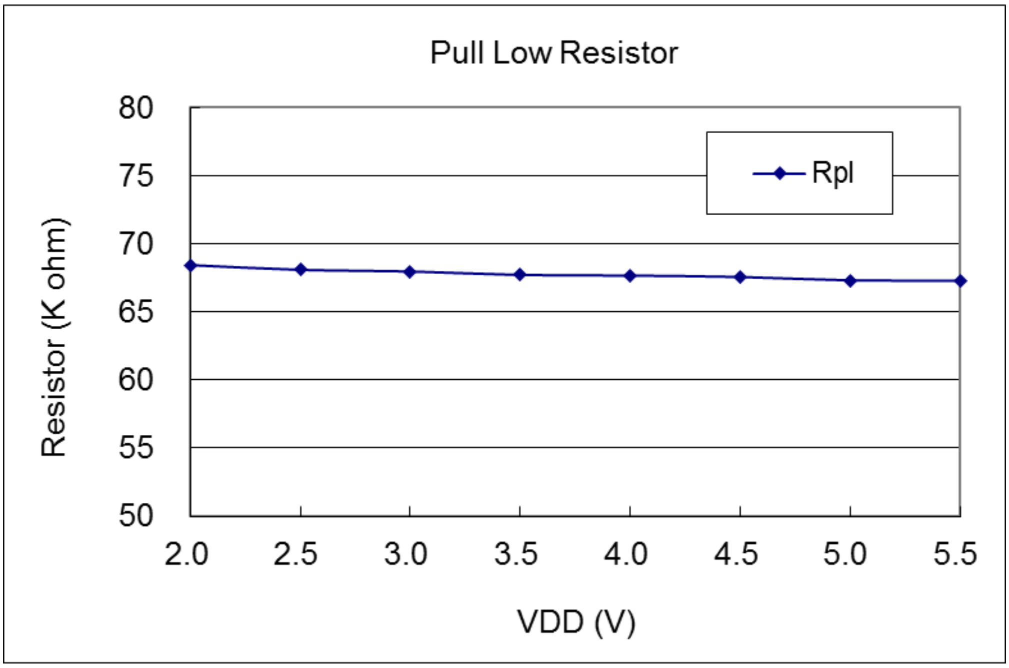
- Conditions:
  tog pa0(1s), **ON**: Bandgap, LVR, IHRC
  no t16m, no interrupt, no floating IO pins, disable ILRC, **Touch**: Disable



4.11. Typical IO pull high resistance

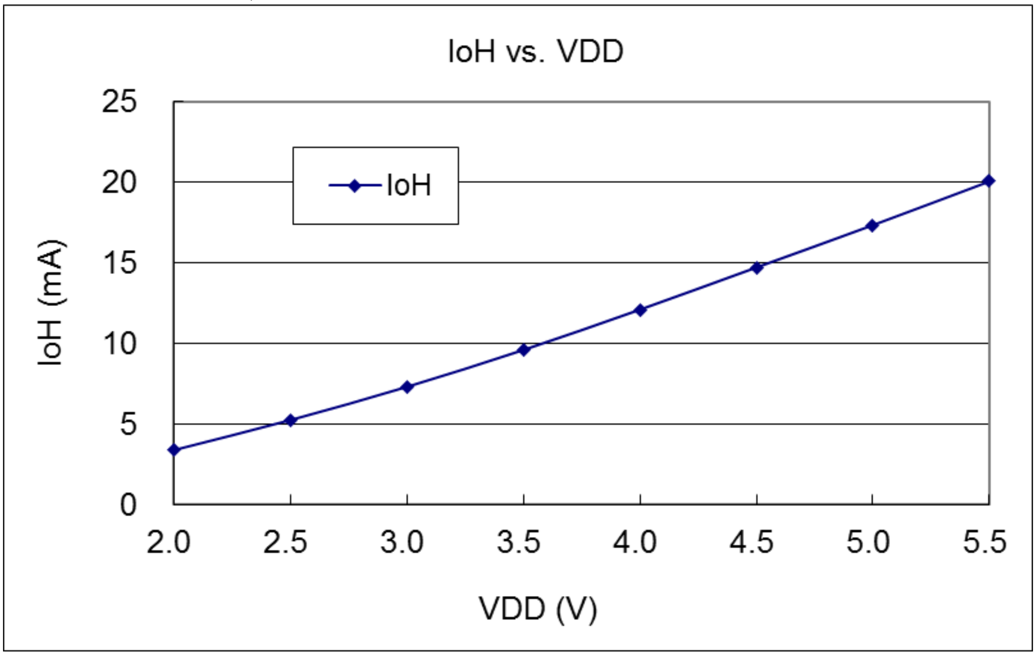


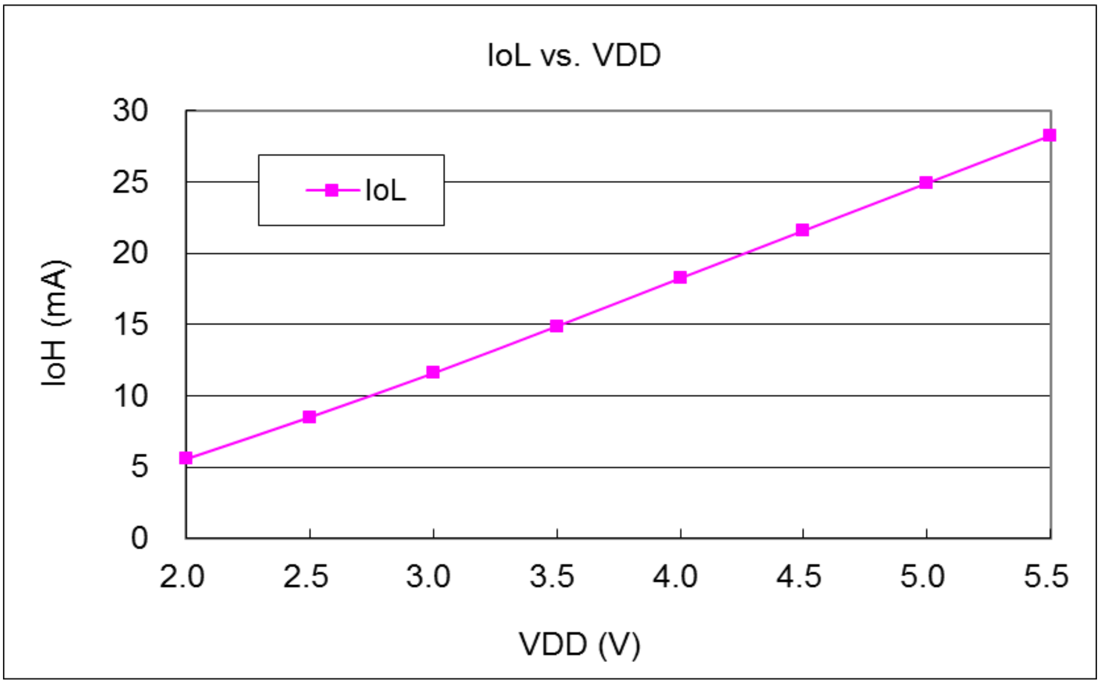
4.12. Typical IO pull low resistance



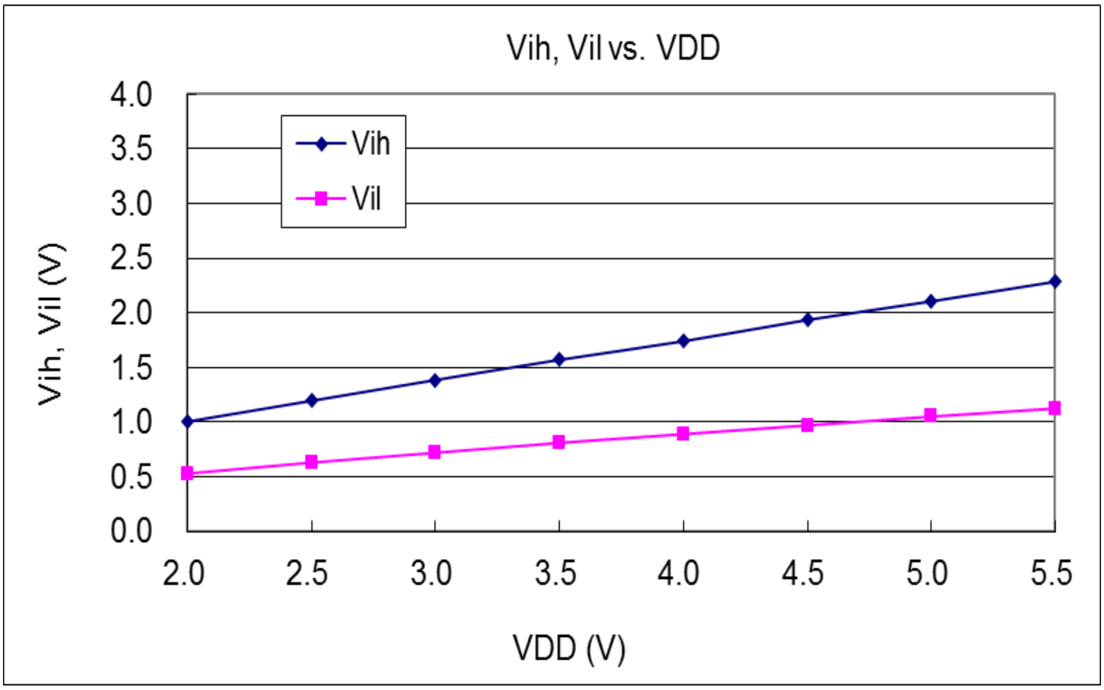
4.13. Typical IO driving current (I_{OH}) and sink current (I_{OL})

（VOH=0.9*VDD, VOL=0.1*VDD）

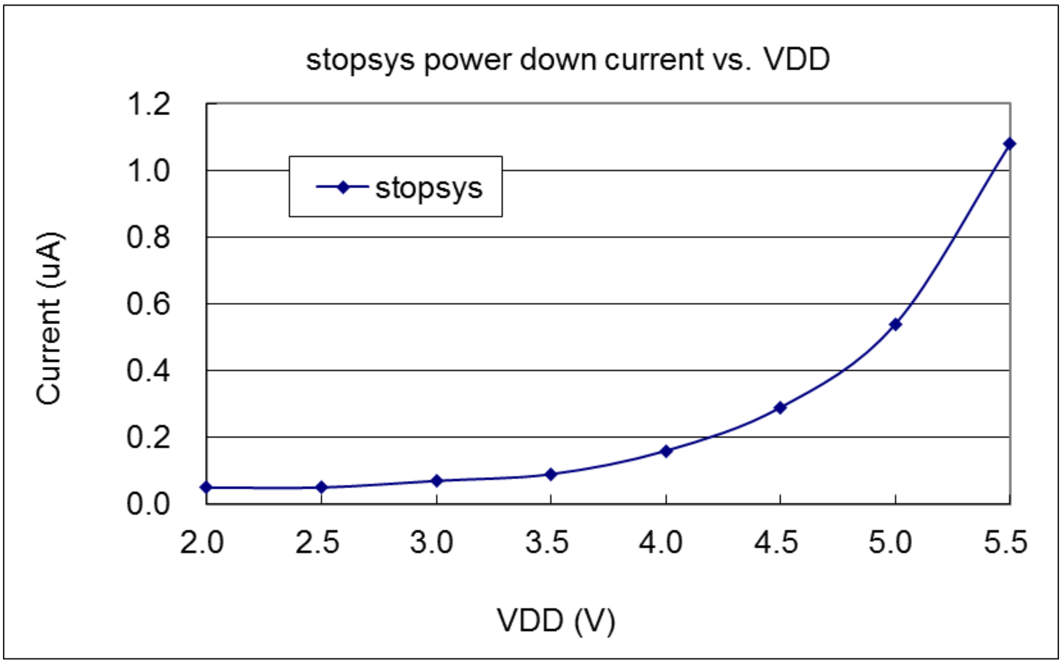


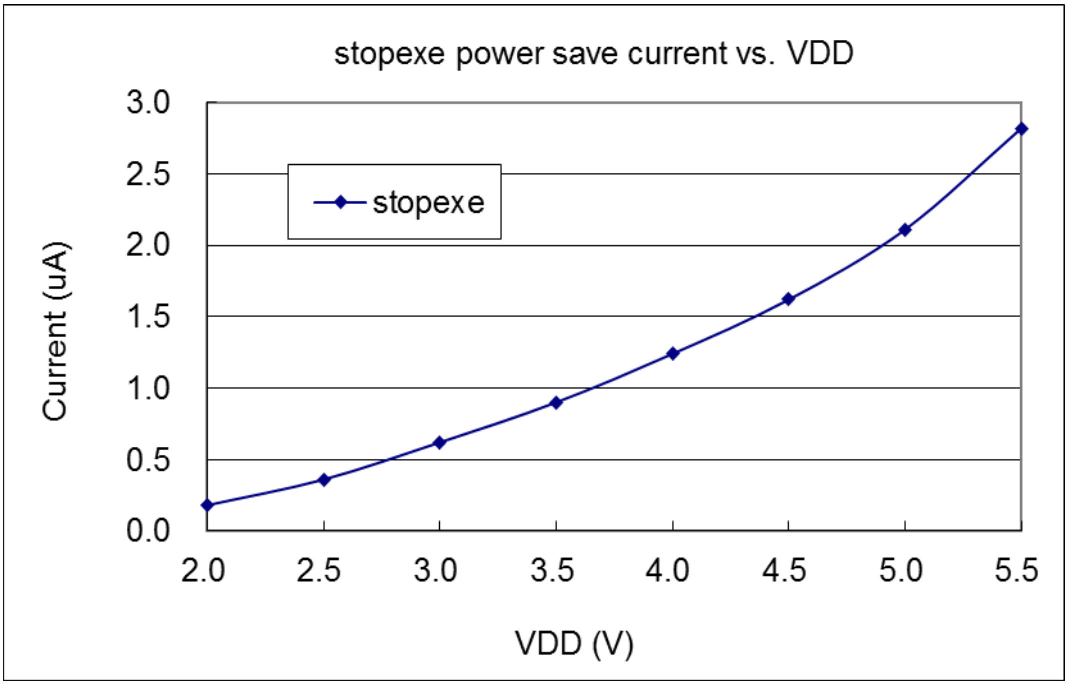


4.14. Typical IO input high/ low threshold voltage (V_{IH}/ V_{IL})



4.15. 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 contain the data, tables and interrupt entry. After reset, the initial address for FPP0 is 0x000. The interrupt entry is 0x010 if used. The OTP program memory for PMS160B is a 1.5KW that is partitioned as Table 1. The OTP memory from address 0x5F0 to 0x5FF is for system using, address space from 0x001 to 0x00F and from 0x011 to 0x5EF is user program space.

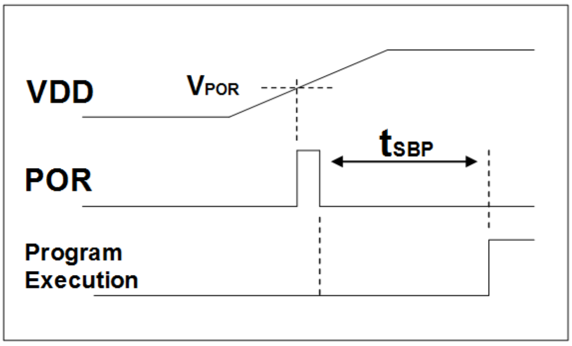
Address	Function
0x000	FPP0 reset – goto instruction
0x001	User program
•	•
•	•
0x00F	User program
0x010	Interrupt entry address
0x011	User program
•	•
0x5EF	User program
0x5F0	System Using
•	•
0x5FF	System Using

Table 1: Program Memory Organization

5.2. Boot Up

POR (Power-On-Reset) is used to reset PMS160B when power up. Customer must ensure the stability of supply voltage after power up no matter which boot up mode is used, the power up sequence is shown in the Fig. 2 and t_{SBP} is the boot up time.

Please noted, during Power-On-Reset, the V_{DD} must go higher than V_{POR} to boot-up the MCU.



Boot up from Power-On Reset

Fig. 2: 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 128 bytes data memory of PMS160B can be accessed by indirect access mechanism.

5.4. Oscillator and clock

There are three oscillator circuits provided by PMS160B: internal high RC oscillator (IHRC), internal low RC oscillator (ILRC/NILRC). IHRC and ILRC oscillators are enabled or disabled by registers clkmd.4 and clkmd.2 independently, and NILRC oscillator remains on by default. User can choose one of these two oscillators (IHRC or ILRC) 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
NILRC	**tm3c.0**

Table 2: Oscillator Module

5.4.1. Internal High RC oscillator and Internal Low RC oscillator

After boot-up, IHRC, ILRC and NILRC oscillators are enabled. The frequency of IHRC can be calibrated to eliminate process variation by ***ihrcr*** register; normally it is calibrated to 16MHz. Please refer to the measurement chart for IHRC frequency verse V_{DD} and IHRC frequency verse temperature.

The frequency of ILRC will vary by process, supply voltage and temperature, please refer to DC specification and do not use for accurate timing application.

NILRC oscillator is slower than ILRC, which can be used to much more power-save wake clock. NILRC and ILRC can estimate frequency by IHRC, but NILRC's error is greater, so it needs to estimate frequency in advance before using NILRC. If related demo is needed, please contact with FAE.

5.4.2. IHRC calibration

The IHRC frequency may be different chip by chip due to manufacturing variation, PMS160B 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, V_{DD}=(**p3**)V

Where, **p1**=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=2.3 ~ 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 / 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 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 PMS160B for different option:

(1) **.**ADJUST_IC SYSCLK=IHRC/4, IHRC=16MHz, V_{DD}=3.3V
After boot up, 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

(2) .ADJUST_ICSYSCLK=IHRC/8, IHRC=16MHz, V_{DD}=2.5V
After boot up, 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

(3) .ADJUST_ICSYSCLK=IHRC/16, IHRC=16MHz, V_{DD}=2.3V
After boot up, CLKMD = 0x1C:
- IHRC frequency is calibrated to 16MHz@V_{DD}=2.3V and IHRC module is enabled
- System CLK = IHRC/16 = 1MHz
- Watchdog timer is disabled, ILRC is enabled, PA5 is in input mode

(4) .ADJUST_ICSYSCLK=IHRC/32, IHRC=16MHz, V_{DD}=5V
After boot up, 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

(5) .ADJUST_ICSYSCLK=ILRC, IHRC=16MHz, VDD=5V
After boot up, 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

(6) .ADJUST_IC DISABLE
After boot up, CLKMD is not changed (Do nothing):
- IHRC is not calibrated and IHRC module is to be enabled or disabled by Code Option Boot-up Time.
- System CLK = ILRC or IHRC/64 (by Code Option Boot-up_Time)
- Watchdog timer is enabled, ILRC is enabled, PA5 is 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 PMS160B is shown as Fig. 3.

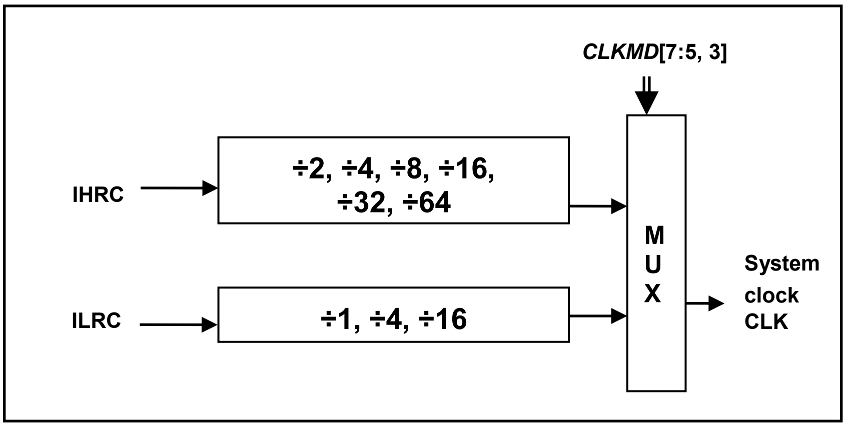


Fig. 3: Options of System Clock

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

5.4.5. 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 PMS160B can be switched among IHRC and ILRC 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.

<u>Case 1:</u> Switching system clock from ILRC to IHRC/8

…			//	*system clock is ILRC*
CLKMD.4	**=**	***1;***	//	***turn on IHRC first to improve anti-interference ability***
CLKMD	**=**	***0x3C；***	//	*switch to IHRC/8，ILRC **CAN NOT** be disabled here*
CLKMD.2	**=**	***0；***	//	***if need,** ILRC **CAN** be disabled at this time*
…				

<u>Case 2:</u> Switching system clock from IHRC/8 to ILRC

…			//	*system clock is IHRC/8*
CLKMD	**=**	***0xF4；***	//	*switch to ILRC, IHRC **CAN NOT** be disabled here*
CLKMD.4	**=**	***0；***	//	*IHRC **CAN** be disabled at this time*
…				

<u>Case 3:</u> Switching system clock from IHRC/8 to IHRC/32

…			//	*system clock is IHRC/8, ILRC is enabled here*
CLKMD	**=**	***0X7C；***	//	*switch to IHRC/32*
…				

<u>Case 4:</u> 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/8 and turn off ILRC oscillator at the same time*

5.5. Comparator

One hardware comparator is built inside the PMS160B; Fig.4 shows its hardware diagram. It can compare signals between two pins or with either internal reference voltage $V_{internal\ R}$ or internal bandgap reference voltage. The two signals to be compared, one is the plus input and the other one is the minus input. For the minus input of comparator can be PA3, PA4, Internal bandgap 1.20 volt, PA6 or $V_{internal\ R}$ selected by bit [3:1] of gpcc register, and the plus input of comparator can be PA4 or $V_{internal\ R}$ selected by bit 0 of gpcc register.

The comparator result can be selected through gpcs.7 to forcibly output to PA0 whatever input or output state. It can be a direct output or sampled by Timer2 clock (TM2_CLK) which comes from Timer2 module. The output polarity can be also inverted by setting gpcc.4 register. The comparator output can be used to request interrupt service or read through gpcc.6.

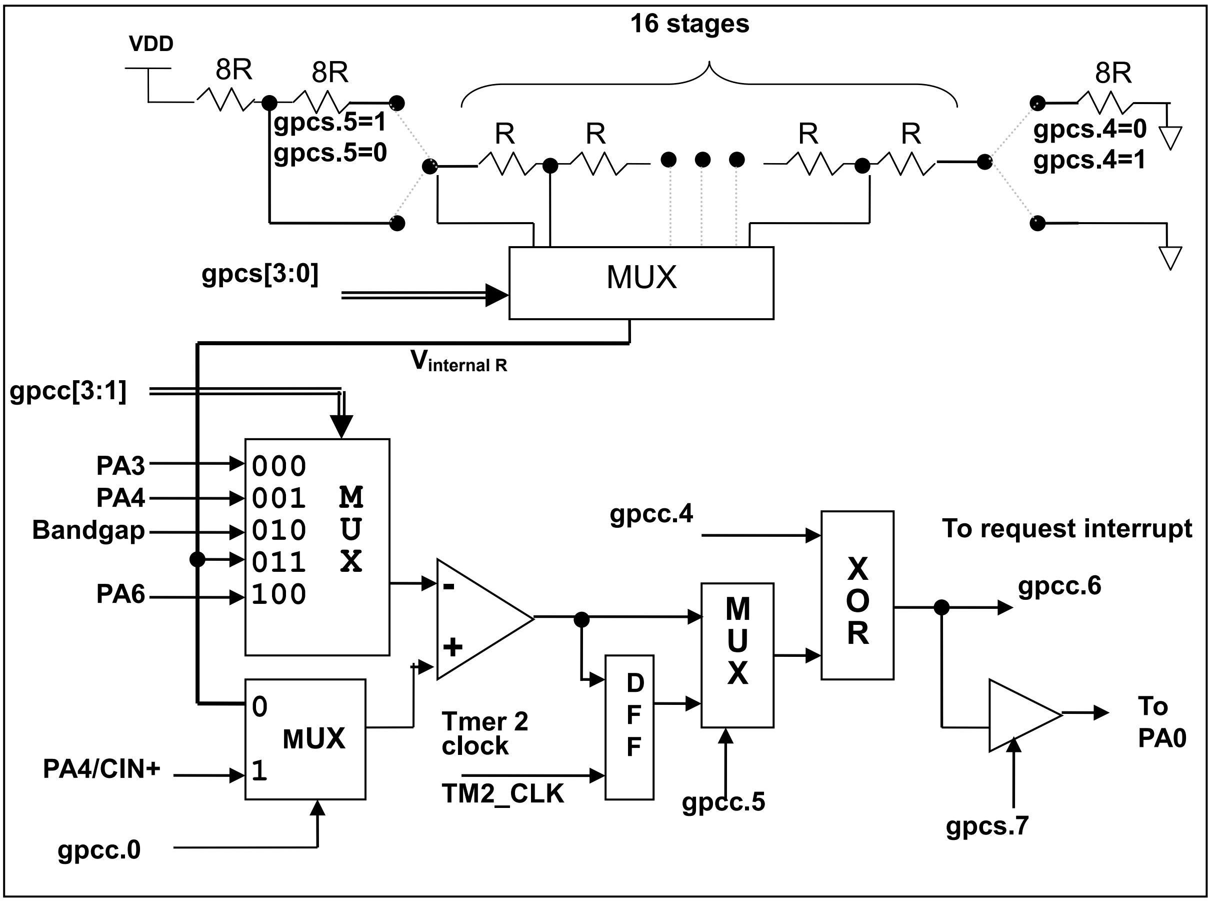
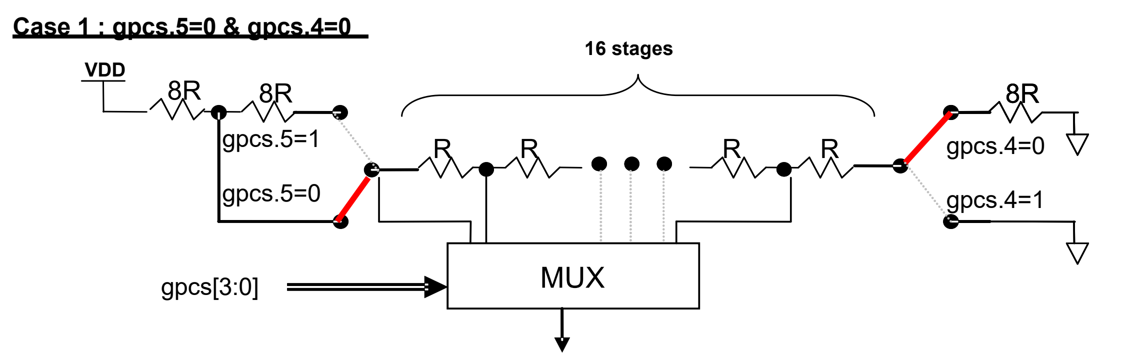


Fig.4: Hardware diagram of comparator

5.5.1. Internal reference voltage ($V_{internal\ R}$)

The internal reference voltage $V_{internal\ R}$ is built by series resistance to provide different level of reference voltage, bit 4 and bit 5 of ***gpcs*** register are used to select the maximum and minimum values of $V_{internal\ R}$ and bit [3:0] of ***gpcs*** register are used to select one of the voltage levels which is deivided-by-16 from the defined maximum level to minimum level. Fig.5 to Fig.8 shows four conditions to have different reference voltage $V_{internal\ R}$. By setting the ***gpcs*** register, the internal reference voltage $V_{internal\ R}$ can be ranged from (1/32)*V_{DD} to (3/4)*V_{DD}.

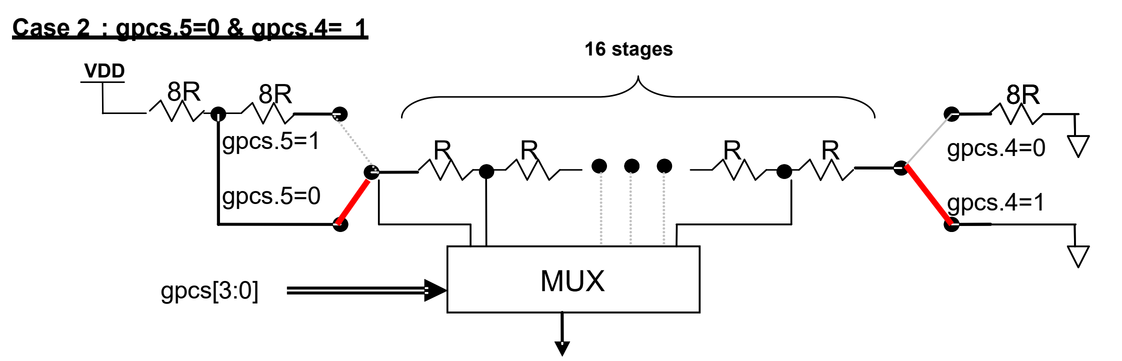


$V_{internal\ R}$ = (3/4) VDD ~ (1/4) VDD + (1/32) VDD

@ gpcs[3:0] = 1111 ~ gpcs[3:0] = 0000

$$V_{internal\ R} = \frac{1}{4} * VDD + \frac{(n+1)}{32} * VDD, \text{ n = gpcs[3:0] in decimal}$$

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



$V_{internal\ R}$ = (2/3) VDD ~ (1/24) VDD

@ gpcs[3:0] = 1111 ~ gpcs[3:0] = 0000

$$V_{internal\ R} = \frac{(n+1)}{24} * VDD, \text{ n = gpcs[3:0] in decimal}$$

Fig.6: $V_{internal\ R}$ hardware connection if gpcs.5=0 and gpcs.4=1

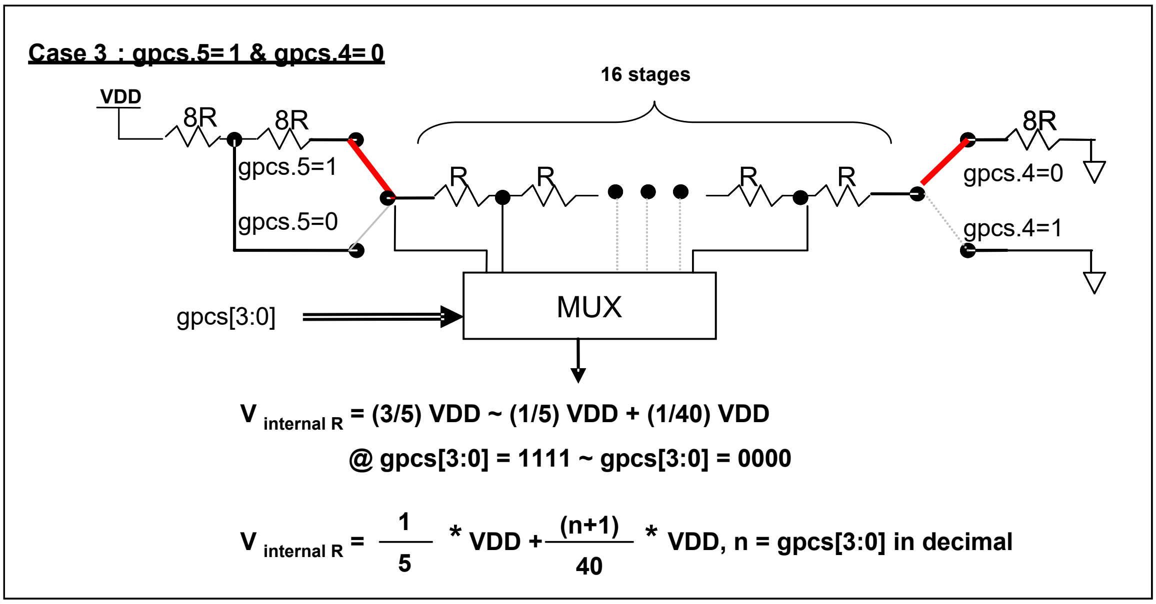


Fig.7: $V_{\text{internal R}}$ hardware connection if gpcs.5=1 and gpcs.4=0

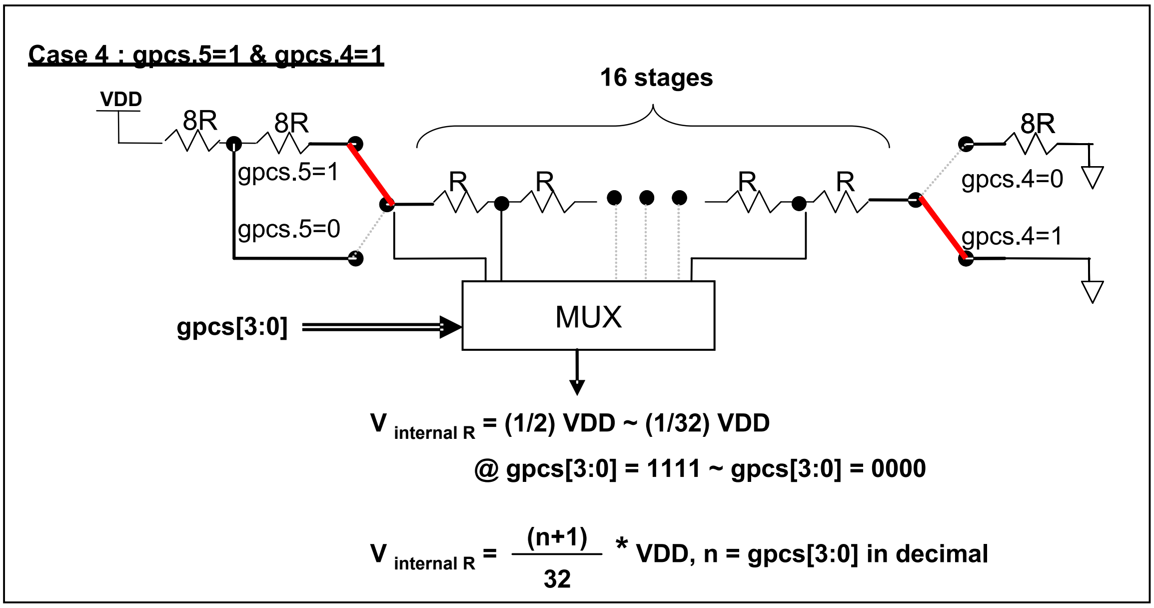


Fig.8: $V_{\text{internal R}}$ hardware connection if gpcs.5=1 and gpcs.4=1

5.5.2. Using the comparator

Case 1:

Choosing PA3 as minus input and $V_{internal\ R}$ with (18/32)*V_{DD} voltage level as plus input, $V_{internal\ R}$ is configured as the above Figure "gpcs[5:4] = 2b'00" and gpcs [3:0] = 4b'1001 (n=9) to have $V_{internal\ R}$ = (1/4)*V_{DD} + [(9+1)/32]*V_{DD} = [(9+9)/32]*V_{DD} = (18/32)*V_{DD}.

```
gpcs   = 0b1_0_00_1001;      // Vinternal R = VDD*(18/32)
gpcc   = 0b1_0_0_0_000_0;    // enable comp, - input: PA3, + input: Vinternal R
padier = 0bxxxx_0_xxx;       // disable PA3 digital input to prevent leakage current
```

or

```
$ GPCS  VDD*18/32;
$ GPCC  Enable, N_PA3, P_R;      // - input: N_xx， + input: P_R(Vinternal R)
PADIER = 0bxxxx_0_xxx;
```

Case 2:

Choosing $V_{internal\ R}$ as minus input with (22/40)*VDD voltage level and PA4 as plus input, the comparator result will be inversed and then output to PA0. $V_{internal\ R}$ is configured as the above Figure "gpcs[5:4] = 2b'10" and gpcs [3:0] = 4b'1101 (n=13) to have $V_{internal\ R}$ = (1/5)*V_{DD} + [(13+1)/40]*V_{DD} = [(13+9)/40]*V_{DD} = (22/40)*V_{DD}.

```
gpcs   = 0b1_0_1_0_1101;     // output to PA0,Vinternal R = VDD*(22/40)
gpcc   = 0b1_0_0_1_011_1;    // Inverse output, - input: Vinternal R, + input: PA4
padier = 0bxxx_0_xxxx;       // disable PA4 digital input to prevent leakage current
```

or

```
$ GPCS  Output, VDD*22/40;
$ GPCC  Enable, Inverse, N_R, P_PA4;  // - input: N_R(Vinternal R)， + input: P_xx
PADIER = 0bxxx_0_xxxx;
```

5.5.3. Using the comparator and Bandgap 1.20V

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

For using Case 1: V_{DD} = [32 / (N+9)] * 1.20 volt ;
For using Case 2: V_{DD} = [24 / (N+1)] * 1.20 volt ;
For using Case 3: V_{DD} = [40 / (N+9)] * 1.20 volt ;
For using Case 4: V_{DD} = [32 / (N+1)] * 1.20 volt ;

More information and sample code, please refer to IDE utility.

Case 1:

```
$ GPCS   VDD*12/40;                // 4.0V * 12/40 = 1.2V
$ GPCC   Enable, BANDGAP, P_R;     // - input: BANDGAP, + input: P_R(Vinternal R)
....
if  (GPC_Out)                      // or GPCC.6
{                                  // when VDD > 4V
}
else
{                                  // when VDD < 4V
}
```

5.6. 16-bit Timer (Timer16)

PMS160B 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. 9.

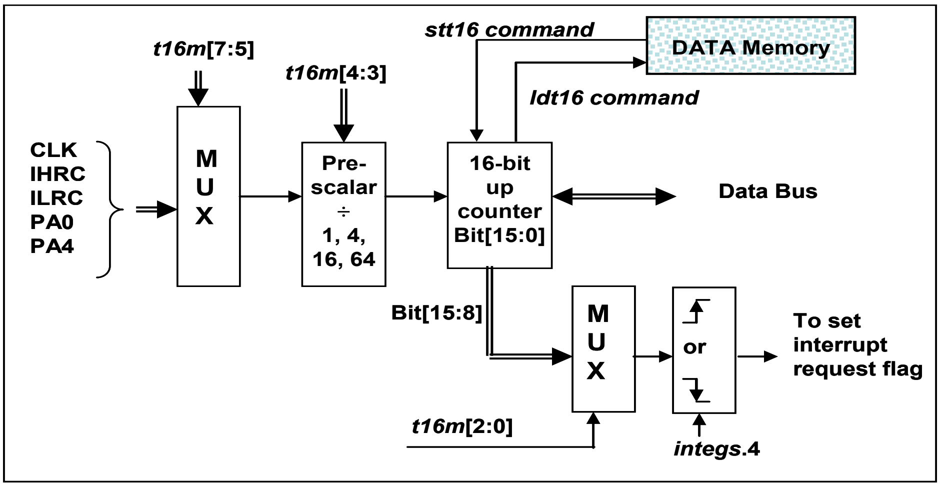


Fig. 9: Hardware diagram of Timer16

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:

$ 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

5.7. Watchdog Timer

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

- 8k ILRC clocks period if register misc[1:0]=00 (default)
- 16k ILRC clocks period if register misc[1:0]=01
- 64k ILRC clocks period if register misc[1:0]=10
- 256k ILRC clocks period if register misc[1:0]=11

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. Besides, the watchdog period will also be shorter than expected after Reset or Wakeup events. It is suggested to clear WDT by wdreset command after these events to ensure enough clock periods before WDT timeout.

When WDT is timeout, PMS160B will be reset to restart the program execution. The relative timing diagram of watchdog timer is shown as Fig.10.

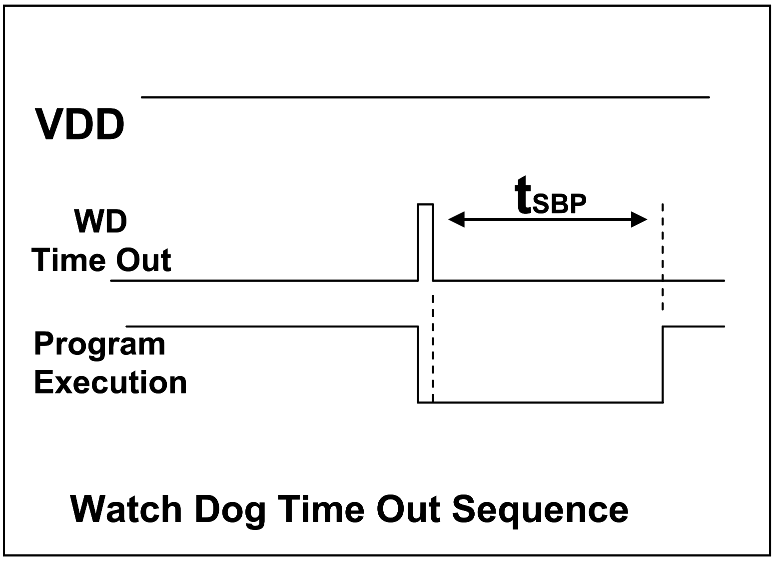


Fig. 10: Sequence of Watch Dog Time Out

5.8. Interrupt

There are 6 interrupt lines for PMS160B:

- External interrupt PA0 / PA5
- Timer16 interrupt
- Timer2 interrupt
- Timer3 interrupt
- GPC interrupt
- LPWM interrupt
- Touch 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. 11. 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.

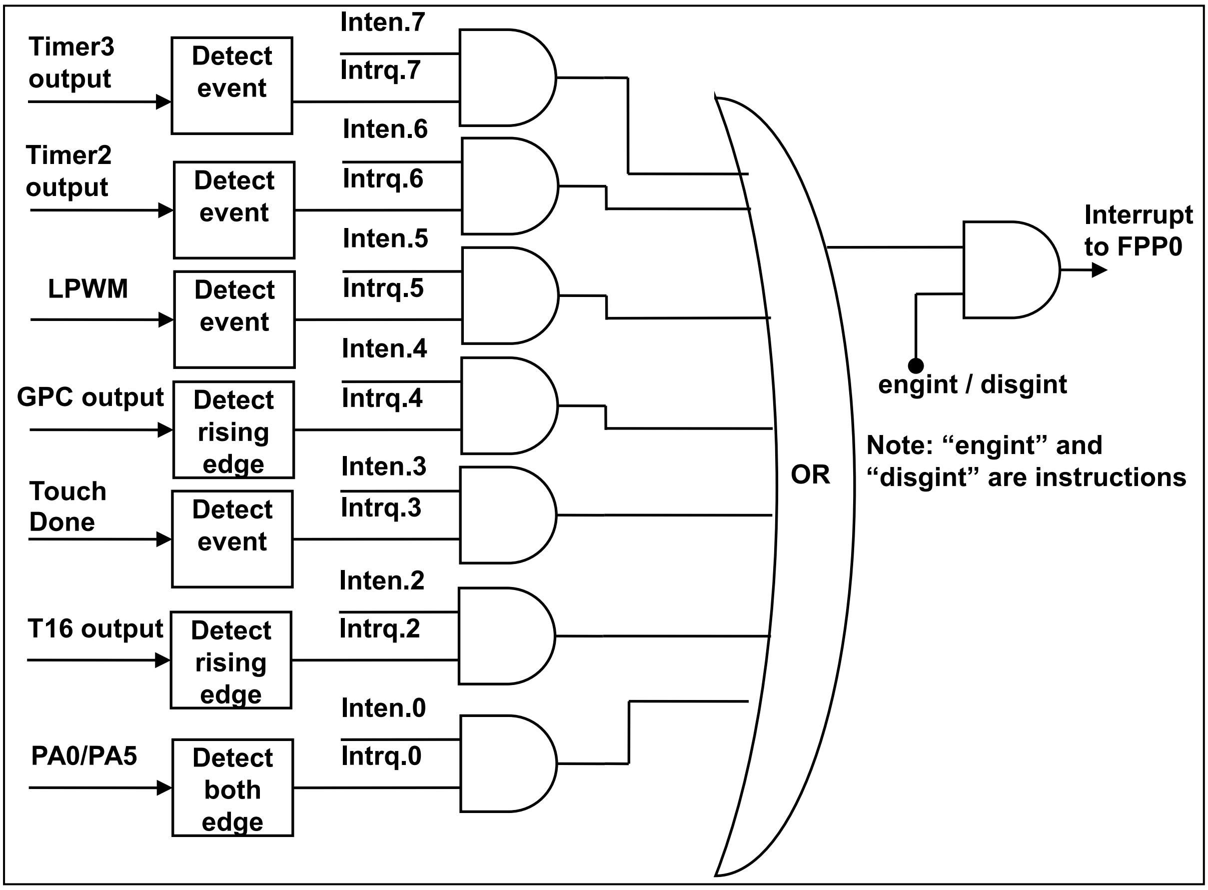


Fig. 11: Hardware diagram of Interrupt controller

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 one level 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
}
```

5.9. 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 4 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	NILRC
STOPSYS	Stop	Stop	No Change
STOPEXE	No Change	No Change	No Change

Table 4: Differences in oscillator modules between STOPSYS and STOPEXE

5.9.1. 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, TM2C/TM3C wake up with NILRC clock source which needs to set MISC2.0=1 to enable the NILRC or wake-up by comparator when setting GPCC.7=1 and GPCS.6=1 to enable the comparator wake-up function at the same time. Wake-up from input pins can be considered as a continuation of normal execution, the detail information for Power-Save mode shows 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
- Timer counter: Stop counting if system clock is selected by clock source or the corresponding oscillator module is disabled; otherwise, it keeps counting. (The Timer contains TM16, TM2, TM3, LPWMG0/1/2)
- Wake-up sources:
  a. IO toggle wake-up: IO toggling in digital input mode (*PxC* bit is 1 and *PxDIER* bit is 1).
  b. Timer Counter wake-up: If the clock source of Timer is not the SYSCLK, the system will be awakened when the Timer counter reaches the set value. It is awakened on both the rising and falling edges.
  c. TM2C/TM3C wake up with NILRC clock source: it needs setting MISC2.0=1 to enable the NILRC, at the same time, the clock source of Timer2/Timer3 selects NILRC.
  d. Comparator wake-up: It need setting *GPCC*.7=1 and *GPCS*.6=1 to enable the comparator wake-up function at the same time. Please note: the internal 1.20V bandgap reference voltage is not suitable for the comparator wake-up function.

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

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

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

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

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

5.9.2. 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. It is recommended to set GPCC.7=0 to disable the comparator before the command “*stopsys*”. The following shows the internal status of PMS160B in detail when “***stopsys***” command is issued:

- All the oscillator modules are turned off
- OTP memory is turned off
- The contents of SRAM and registers remain unchanged
- Wake-up sources:
    a. IO toggle in digital mode (PxDIER bit is 1)
    b. TM2C/TM3C wake up with NILRC clock source: it needs setting MISC2.0=1 to enable the NILRC at the same time, the clock source of Timer2/Timer3 selects NILRC.

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:

```
CLKMD   =   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   =   0x3C;     //   Change clock from ILRC to IHRC/8
```

5.9.3. Wake-up

After entering the Power-Down or Power-Save modes, the PMS160B can be resumed to normal operation by toggling IO pins, Timer wake-up is available for Power-Save mode ONLY. Table 5 shows the differences in wake-up sources between STOPSYS and STOPEXE.

Differences in wake-up sources between STOPSYS and STOPEXE				
	IO Toggle	TM2C/TM3C wake up with NILRC clock source	Timer16 wake-up	Comparator wake-up
STOPSYS	Yes	Yes	No	NO
STOPEXE	Yes	Yes	Yes	Yes

Table 5: Differences in wake-up sources between Power-Save mode and Power-Down mode

When using the IO pins to wake-up the PMS160B, registers ***padier*** *and* ***pbdier*** should be properly set to enable the wake-up function for every corresponding pin. The time for normal wake-up is about 16 ILRC clocks counting from wake-up event; fast wake-up can be selected to reduce the wake-up time by *misc*.5 register, and the time for fast wake-up is 8 ILRC clocks from IO toggling.

Suspend mode	Wake-up mode	Wake-up time (t_{WUP}) from IO toggle
STOPEXE suspend or STOPSYS suspend	fast wake-up	8 * T_{ILRC}, Where T_{ILRC} is the time period of ILRC
STOPEXE suspend or STOPSYS suspend	normal wake-up	16 * T_{ILRC}, Where T_{ILRC} is the clock period of ILRC

Table 6: Differences in wake-up time between fast/normal wake-up

5.10. IO Pins

All the IO pins have the same structure. When PMS160B 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.

For pins selected for analog function, the corresponding bit in register PADIER must be set to low to prevent leakage current.

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 7 shows the configuration table of bit 0 of port A. The hardware diagram of IO buffer is also shown as Fig. 12.

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

Table 7: PA0 Configuration Table

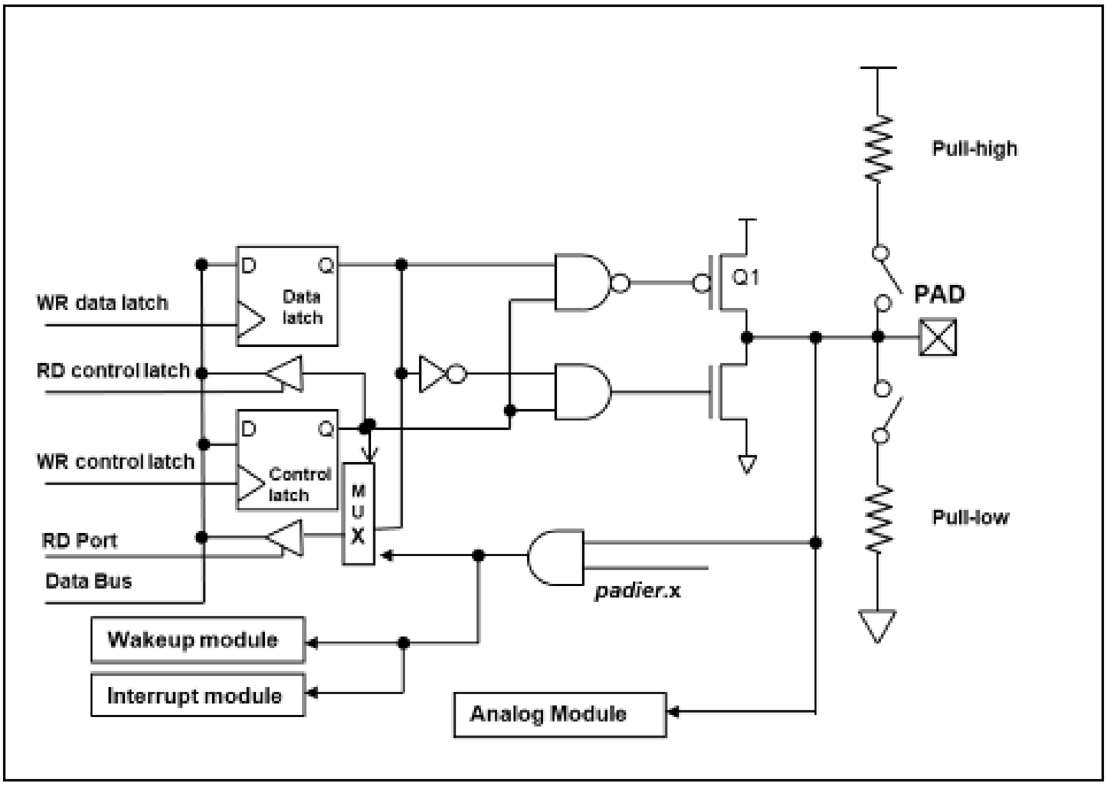


Fig. 12: Hardware diagram of IO buffer

5.11. Reset

There are many causes to reset the PMS160B, once reset is asserted, all the registers in PMS160B will be set to default values, system should be restarted once abnormal cases happen, or by jumping program counter to address 0x00.

After a power-on reset or LVR reset occurs, if VDD is greater than VDR (data storage voltage), the value of the data memory will be retained, but if the SRAM is cleared after re-power, the data cannot be retained; if VDD is less than VDR, the data the value of the memory will be turned into an unknown state that is in an indeterminate state.

If a reset occurs, and there is an instruction or syntax to clear SRAM in the program, the previous data will be cleared during program initialization and cannot be retained.
The content will be kept when reset comes from PRSTB pin or WDT timeout.

5.12. 8-bit Timer (Timer2)

The 8-bit hardware timer Timer2 is implemented in the PMS160B, the clock sources of Timer2 may come from system clock, internal high RC oscillator (IHRC), internal low RC oscillator (ILRC/NILRC), comparator, PA0 and PA4, bit [7:4] of register tm2c are used to select the clock of Timer2. A clock pre-scaling module is provided with divided-by-1, 4, 16, and 64 options, controlled by bit [6:5] of tm2s register; one scaling module with divided-by-1~32 is also provided and controlled by bit [4:0] of tm2s register. In conjunction of pre-scaling function and scaling function, the frequency of Timer2 clock (TM2_CLK) can be wide range and flexible.

The Timer2 counter performs 8-bit up-counting operation only; the counter values can be set or read back by tm2ct register. The 8-bit counter will be clear to zero automatically when its values reach for upper bound register, the upper bound register is used to define the period of timer. There are one operating modes for Timer2: period mode; period mode is used to generate periodical output waveform or interrupt event. Fig. 14 shows the timing diagram of Timer2 for period mode.

Bit [7:4] of register TM2C/TM3C selects NILRC as clock source, which can support lower-power wake-up “stopexe” and “stopsys”. NILRC is a slower clock than ILRC, and it is used to make a wake-up clock source with lower power consumption. NILRC and ILRC estimate frequency through IHRC, however NILRC’s frequency drifts a lot. It needs to estimate frequency before it can be used. If users need related demo, please contact FAE.

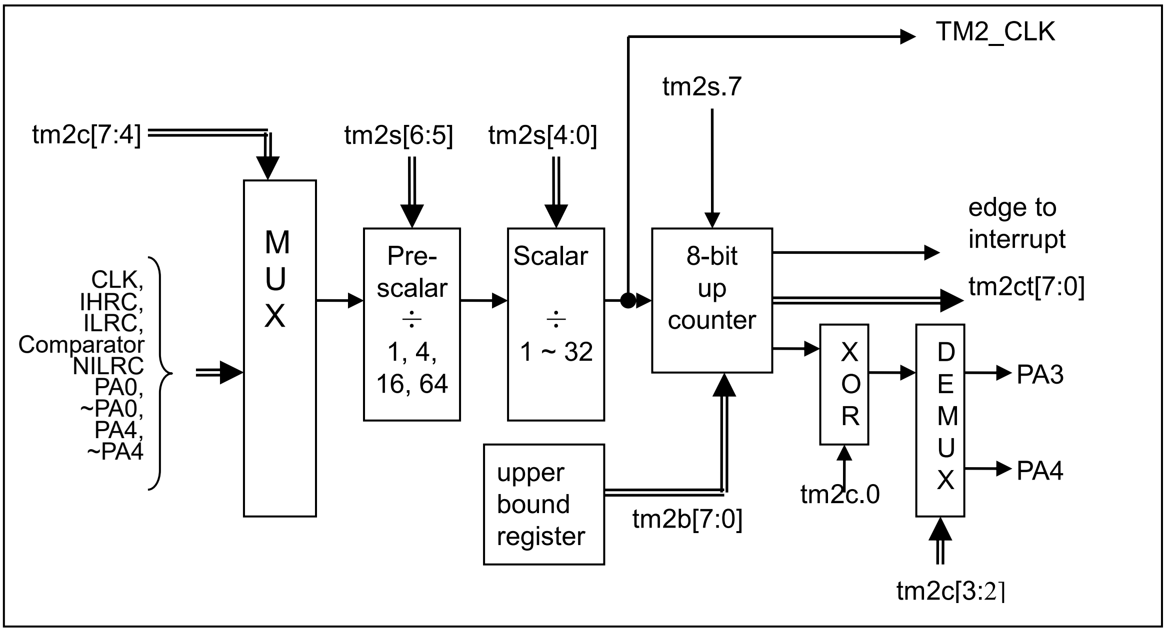


Fig. 13: Timer2 hardware diagram

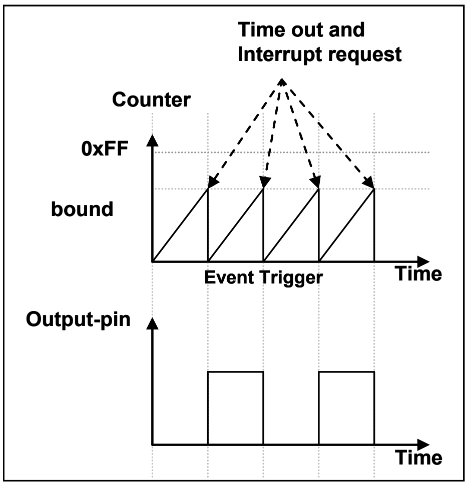


Fig. 14: Timing diagram of Timer2 in period mode

5.12.1. Using the Timer2 to generate periodical waveform

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

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

Where, Y = tm2c[7:4] : frequency of selected clock source
K = tm2b[7:0] : bound register in decimal
S1 = tm2s[6:5] : pre-scalar (S1= 1, 4, 16, 64)
S2 = tm2s[4:0] : scalar register in decimal (S2= 0 ~ 31)

Example 1:

tm2c = 0b0001_1000, Y=8MHz
tm2b = 0b0111_1111, K=127
tm2s = 0b0_00_00000, S1=1, S2=0
➔ frequency of output = 8MHz ÷ [2 × (127+1) × 1 × (0+1)] = 31.25kHz

Example 2:

tm2c = 0b0001_1000, Y=8MHz
tm2b = 0b0111_1111, K=127
tm2s[7:0] = 0b0_11_11111, S1=64 , S2 = 31
➔ frequency = 8MHz ÷ (2 × (127+1) × 64 × (31+1)) =15.25Hz

Example 3:

tm2c = 0b0001_1000, Y=8MHz
tm2b = 0b0000_1111, K=15
tm2s = 0b0_00_00000, S1=1, S2=0
➔ frequency = 8MHz ÷ (2 × (15+1) × 1 × (0+1)) = 250kHz

Example 4:

tm2c = 0b0001_1000, Y=8MHz
tm2b = 0b0000_0001, K=1
tm2s = 0b0_00_00000, S1=1, S2=0
➔ frequency = 8MHz ÷ (2 × (1+1) × 1 × (0+1)) =2MHz

The sample program for using the Timer2 to generate periodical waveform to PA3 is shown as below:

```
void    FPPA0 (void)
{
        . ADJUST_IC     SYSCLK=IHRC/2, IHRC=16MHz, VDD=5V
        …
        tm2ct = 0x00;
        tm2b = 0x7f;
        tm2s = 0b0_00_00001;            //      8-bit PWM, pre-scalar = 1, scalar = 2
        tm2c = 0b0001_10_0_0;           //      system clock, output=PA3, period mode
        while(1)
        {
                nop;
        }
}
```

5.13. 8-bit Timer (Timer3)

The 8-bit hardware timer Timer3 is implemented in the PMS160B, the clock sources of Timer3 may come from system clock, internal high RC oscillator (IHRC), internal low RC oscillator (ILRC/NILRC), and comparator, bit [6:4] of register tm3c are used to select the clock of Timer3. A clock pre-scaling module is provided with divided-by-1, 4, 16, and 64 options, controlled by bit [6:5] of tm3s register; one scaling module with divided-by-1, 2, 4 is also provided and controlled by bit [1:0] of tm3s register. In conjunction of pre-scaling function and scaling function, the frequency of Timer3 clock (TM3_CLK) can be wide range and flexible.

The Timer3 counter performs 8-bit up-counting operation only; the counter values can be set or read back by tm3ct register. The 8-bit counter will be clear to zero automatically when its values reach for upper bound register, the upper bound register is used to define the period of timer. There are one operating modes for Timer3: period mode; period mode is used to generate periodical output waveform or interrupt event. Fig. 16 shows the timing diagram of Timer3 for period mode.

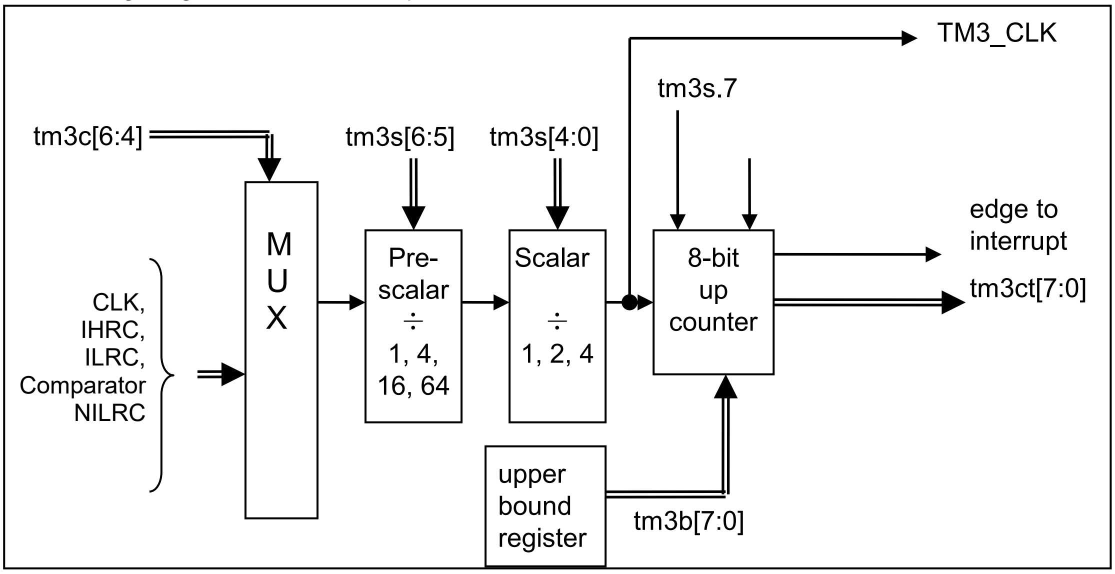
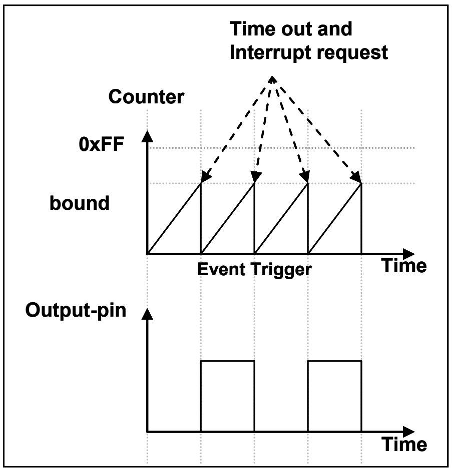


Fig. 15: Timer3 hardware diagram



Mode 0 – Period Mode

Fig. 16: Timing diagram of Timer3 in period mode

5.14. 11-bit SuLED LPWM Generation (LPWMG0/1/2)

Three 11-bit SuLED (Super LED) hardware LPWM generators are built in the PMS160B. Their individual outputs are listed as below:

- LPWMG0 – PA3
- LPWMG1 – PA4
- LPWMG2 – PA0, PA7

5.14.1. LPWM Waveform

A LPWM output waveform (Fig. 17) has a time-base (T_{Period} = Time of Period) and a time with output high level (Duty Cycle). The frequency of the LPWM output is the inverse of the period ($f_{LPWM} = 1/T_{Period}$),

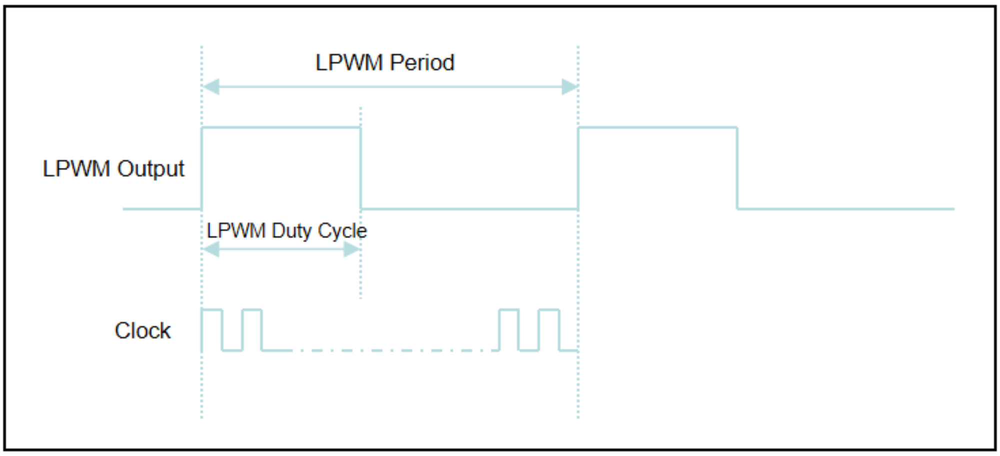


Fig. 17: PWM Output Waveform

5.14.2. Hardware Diagram

Fig. 18 shows the hardware diagram of 11-bit LPWM generation. These triple of LPWM generators use UP-Counter together and clock source selection switch to generate the time-base. So, the starting point(rising) of the LPWM cycle is synchronized, and the clock source can be IHRC or system clock. The LPWM output pin is selected by register *LPWMGxC.* The period of LPWM waveform is defined in the LPWM upper bond high and low registers, the duty cycle of LPWM waveform is defined in the LPWM duty high and low registers.

The two attached logic gates OR and XOR in the LPWMG0 channel are used to generate complementary non-overlapping switches with dead zones to control waveforms.

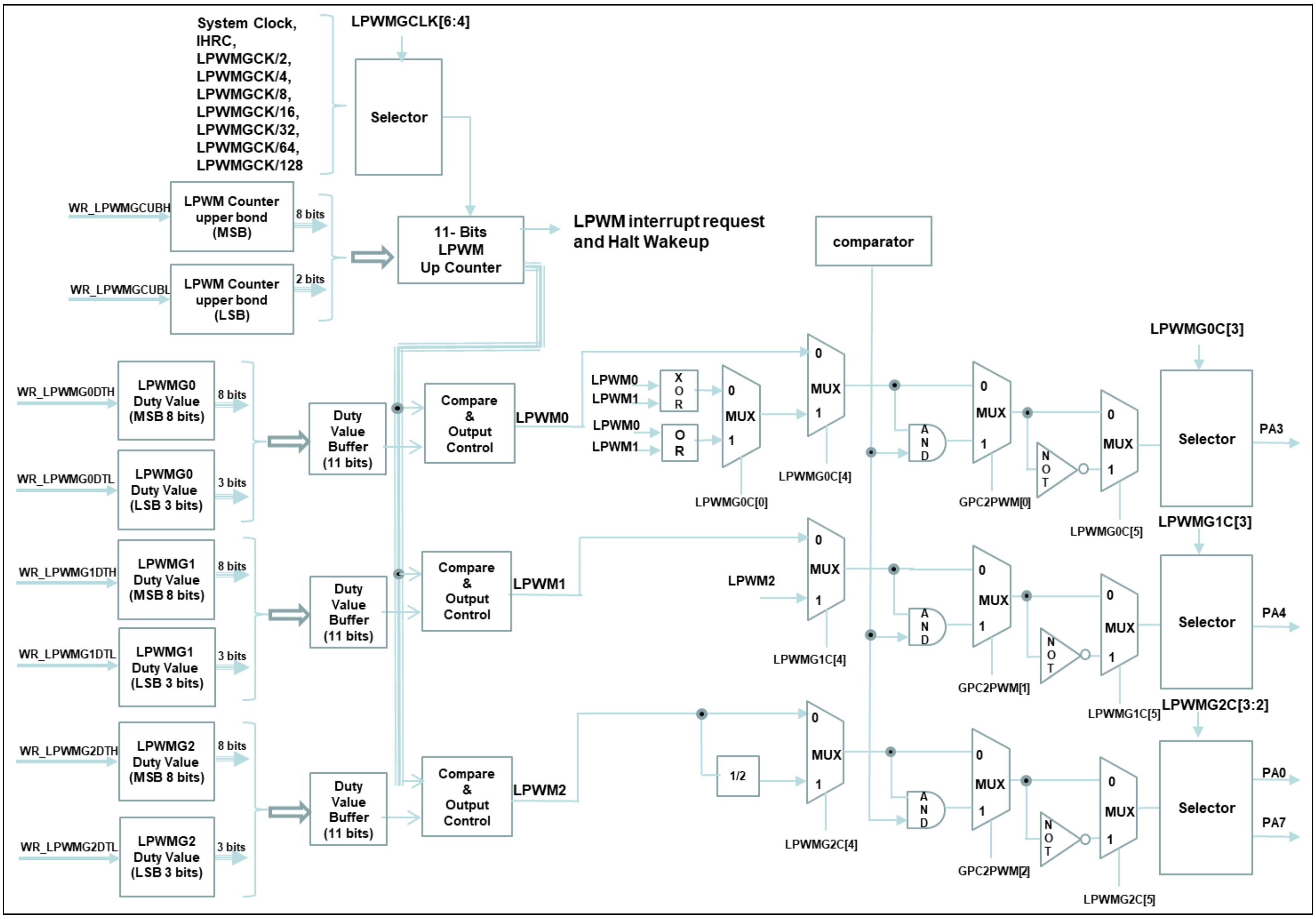


Fig. 18: Hardware Diagram of three 11-bit LPWM Generators

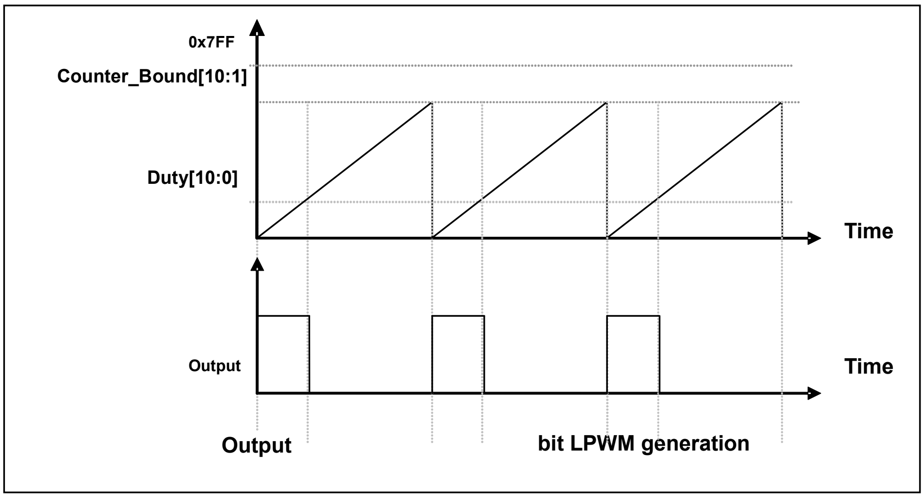


Fig. 19: Output Timing Diagram of 11-bit LPWM Generator

The bit [2:0] of GPC2PWM register can select to control PWM output function of LPWMG0/1/2 by comparator results. And users can select to enable or disable as require. After enabled, the LPWM output stops while the comparator output is 1 and then the LPWM output turns on while the comparator output goes back to 0, as shown in Fig. 20.

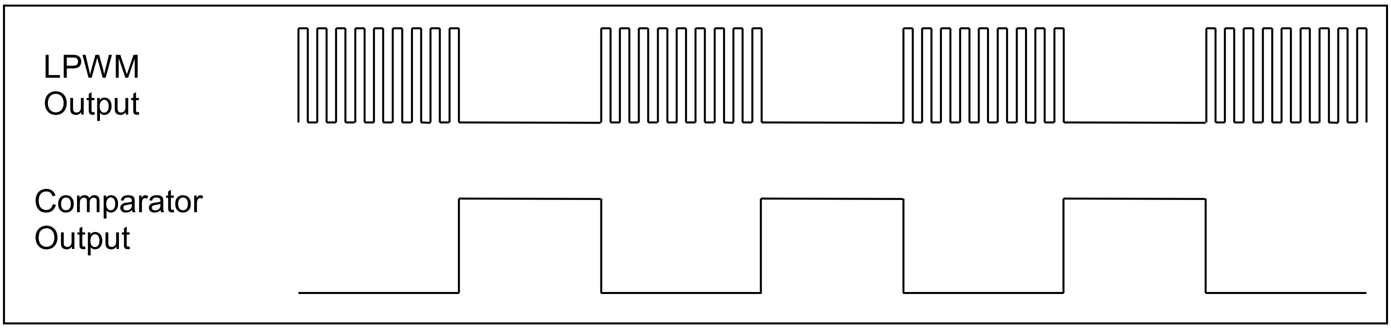


Fig.20：Comparator controls the output of LPWM waveform

5.14.3. Equations for 11-bit LPWM Generator

LPWM Frequency F_{LPWM} = $F_{clock\ source}$ ÷ [P × (CB10_1 + 1)]

LPWM Duty(in time) = (1 / F_{LPWM}) × (DB10_1 + DB0 × 0.5 + 0.5) ÷ (CB10_1 + 1)

LPWM Duty(in percentage)= (DB10_1 + DB0 × 0.5 + 0.5) ÷ (CB10_1 + 1) × 100%

Where,

P=*LPWMGCLK*[6:4]; pre-scalar **P**=1,2,4,8,16,32,64,128

DB10_1 = Duty_Bound[10:1] = {*LPWMGxDTH* [7:0],*LPWMGxDTL*[7:6]},（x=0/1/2） duty bound

DB0 = Duty_Bound[0] = *LPWMGxDTL*[5] （x=0/1/2）

CB10_1 = Counter_Bound[10:1] = {*LPWMGCUBH*[7:0],*LPWMGCUBL*[7:6]}, counter bound

5.14.4. Examples of LPWM Waveforms with Complementary Dead Zones

Based on the specific 11 bit SuLED LPWM architecture of PMS160B, here we employ LPWM2 output and LPWM0 inverse output after LPWM0 xor LPWM1 to generate two LPWM waveforms with complementary dead zones.

Example program is as follows:

(Note: This function **does not** support simulation.)

```
    #define   dead_zone        10          //   dead time = 10% * (1/LPWM_Frequency) us
    #define   LPWM_Pulse       50          //   set 50% as LPWM duty cycle

    #define   LPWM_Pulse_1     35          //   set 35% as LPWM duty cycle
    #define   LPWM_Pulse_2     60          //   set 60% as LPWM duty cycle
    #define   switch_time      400*2       //   adjusting switch time
    //Note：To avoid noise, switch_time must be a multiple of LPWM period. In this example LPWM period =
    //400us,
    // so switch_time = 400*2 us.
void    FPPA0 (void)
{
.ADJUST_IC SYSCLK=IHRC/16, IHRC=16MHz, VDD=5V;
//******* Generating fixed duty cycle waveform ************************
//------- Set the counter upper bound and duty cycle --------------------
LPWMG0DTL     =    0x00;
LPWMG0DTH     =    LPWM_Pulse + dead_zone;

LPWMG1DTL     =    0x00;
LPWMG1DTH     =    dead_zone;       // After LPWMG0 xor LPWMG, LPWM duty
                                    //cycle=LPWM_Pulse%

LPWMG2DTL      =   0x00;
LPWMG2DTH     =    LPWM_Pulse + dead_zone*2;

LPWMGCUBL     =    0x00;
LPWMGCUBH     =    100;
//---- Configure clock and pre-scalar ----------------
$ LPWMGCLK      Enable, /1, sysclk;
//------- Output control    -----------------------------
```

```
$ LPWMG0C  Inverse,LPWM_Gen,PA3,gen_xor; //   After LPWMG0 xor LPWMG,
                                          //   output the inversed waveform through PA3
$ LPWMG1C   LPWMG1,disable;               //   disable LPWMG1 output
$ LPWMG2C     PA0;                        //   output LPWMG2 waveform through PA0

while(1)
{

   //******** Switching duty cycle ********************************
      // To avoid the possible instant disappearance of dead zone, user should comply with the following
      // instruction sequence.
      // When increase the duty cycle: 50%/60%  →  35%
   LPWMG0DTL  =   0x00;
   LPWMG0DTH  =   LPWM_Pulse_1 + dead_zone;
   LPWMG2DTL  =   0x00;
   LPWMG2DTH  =   LPWM_Pulse_1 + dead_zone*2;
   .delay     switch_time

      //When decrease the duty cycle: 35% → 60%
      LPWMG2DTL   =   0x00;
      LPWMG2DTH   =   LPWM_Pulse_2 + dead_zone*2;
   LPWMG0DTL  =   0x00;
   LPWMG0DTH  =   LPWM_Pulse_2 + dead_zone;
   .delay     switch_time
 }
}
```

The following figures show the waveforms at different condition.

1. The PWM waveform in a fixed-duty cycle:

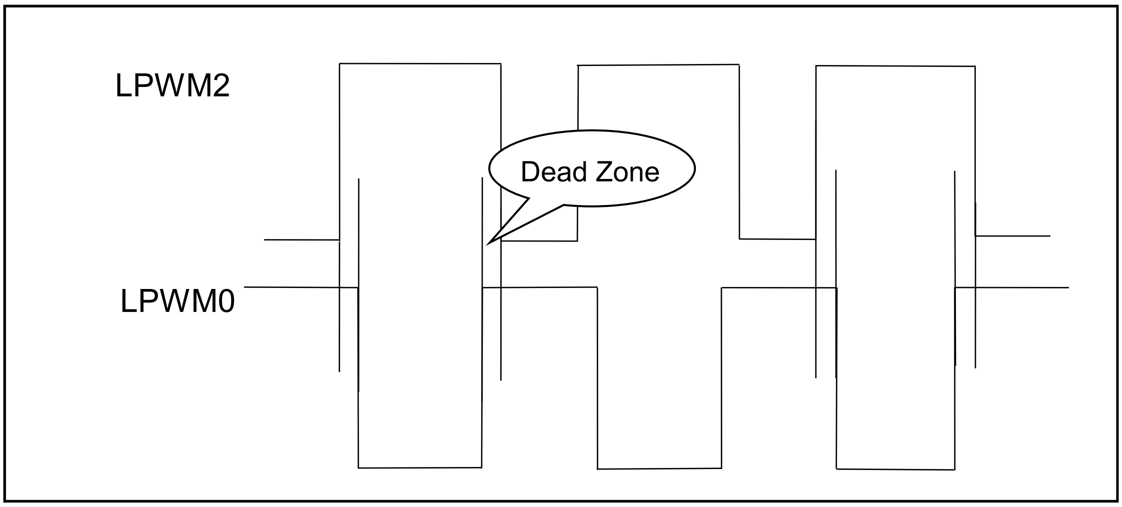


Fig. 21: Complementary LPWM waveform with dead zones

2. PWM waveform when switching two duty cycles:

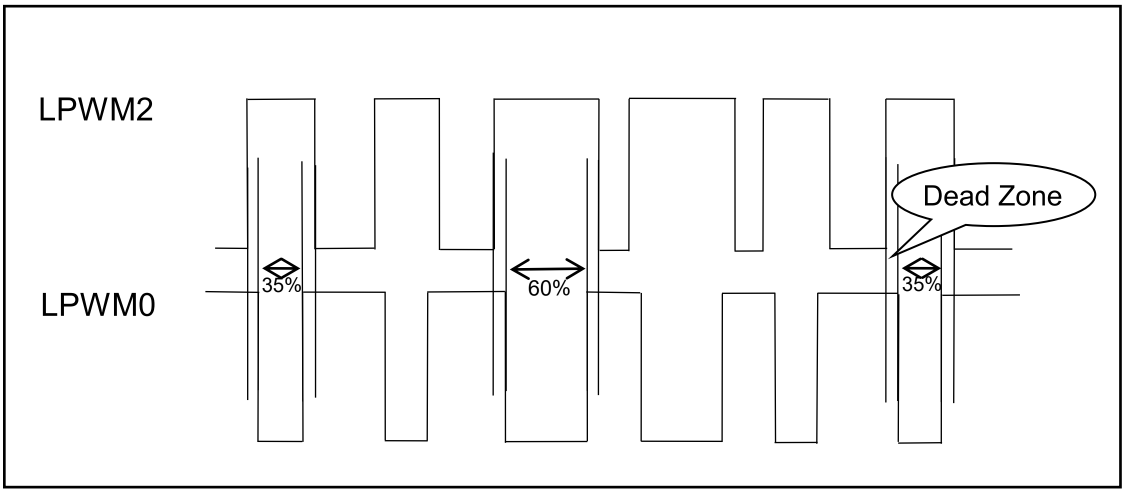


Fig. 22: Complementary LPWM waveform with dead zones

User can find that above example only provides dead zone where LPWM are both in high. If need dead zone where LPWM are both in low, you can realize it by resetting each control register's Inverse like:

$ LPWMG0C LPWM_Gen,PA3,gen_xor;

$ LPWMG2C Inverse, PA0;

5.15. Touch Function

A touch detecting circuit is included in PMS160B. Its functional block diagram is shown as Fig.23.

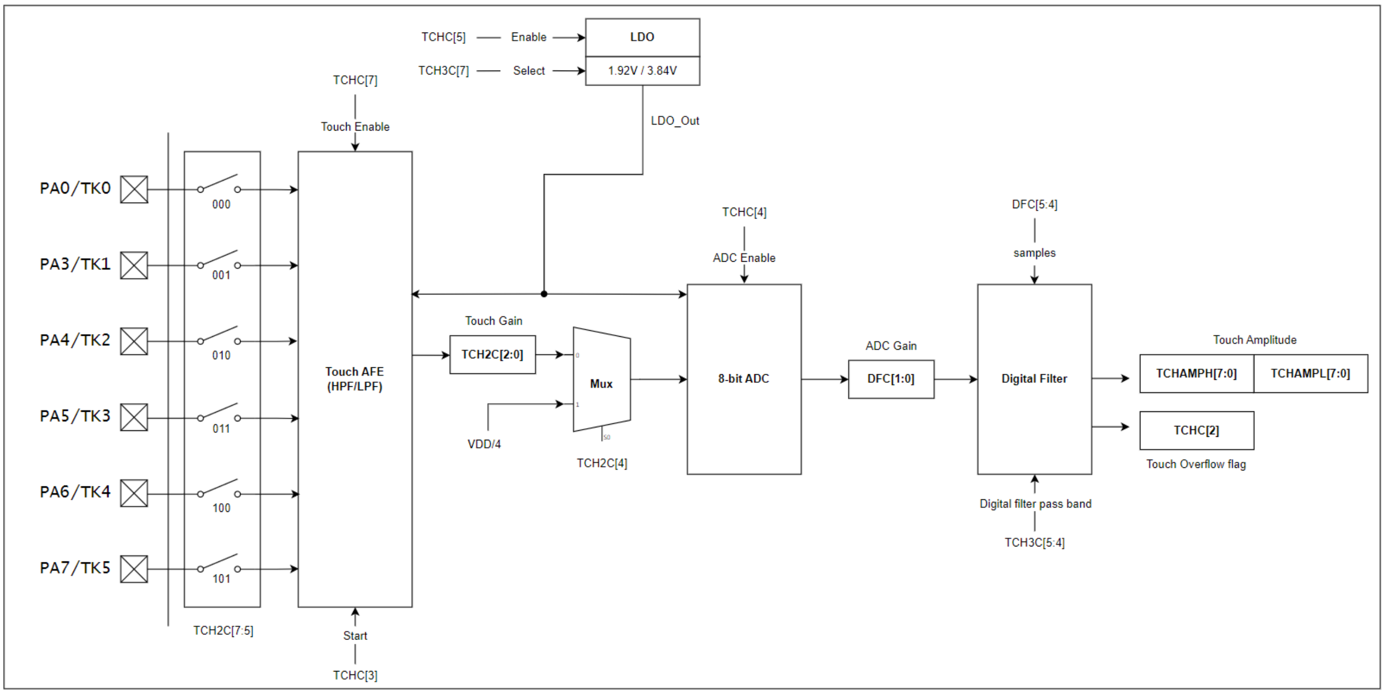


Fig. 23: Functional block diagram of the touch detecting circuit

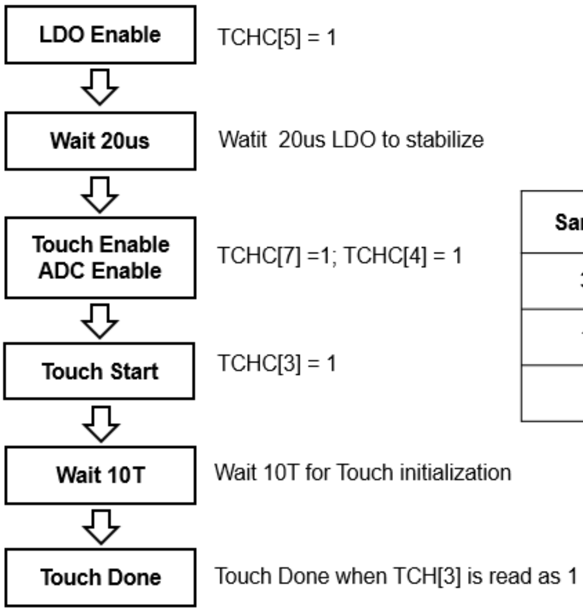
The touch detecting circuits in the PMS160B apply the method of self-capacitance detection. This touch mode includes a Touch AFE, a hardware LDO voltage reference, an 8-bit ADC, and a Digital filter. The circuit operates at a selectable digital filter pass band of 20 kHz, 40 kHz, 80 kHz, or 100 kHz, and the touch performance can be verified through amplitude analysis. There are two touch modes available in the PMS160B: Touch Mode and VDD/4 Mode, which are described in more detail below.

Touch Mode is used for detecting PAD signals, which are first processed by the Touch AFE. The LDO can select between two reference voltages: 1.92V and 3.84V. The 1.92V setting is intended for battery-powered operation, while the 3.84V setting is used when operating with an adapter.

The Touch AFE includes a high-pass filter (HPF) and a low-pass filter (LPF) to eliminate unwanted noise. The filtered analog signal is amplified via TCH2C[2:0], and then converted into a digital signal by the ADC. This resulting digital signal is further amplified using DFC[1:0]. PMS160B uses the Digital filter technique to enhance touch detection performance. If a computation overflow occurs, the Touch Overflow Flag (TCHC.2) is set to ‘1’, indicating that the corresponding touch result is invalid and should be discarded.

By reading the values of TCHAMPH and TCHAMPL, users can obtain the touch amplitude. When a finger touches the pad, the values of TCHAMPH and TCHAMPL will change accordingly. This change can be used to determine whether a touch event has occurred.

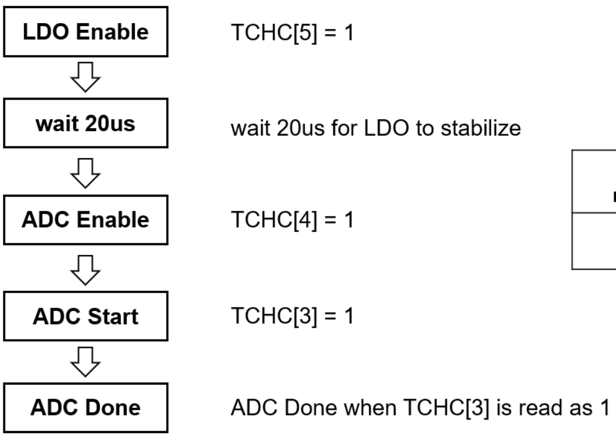
Fig. 24 shows the sequence diagram of Touch Mode. First, enable the LDO and wait 20 us for it to stabilize. Then, enable Touch and ADC. When the start signal (TCHC.3) is set, touch detection begins and requires a 10T delay for initialization. The runtime of the touch operation depends on the sampling rate. For example, if the Digital filter is set to sample 300 times, the touch operation will take approximately 1 ms. To determine whether the touch process is complete, read TCHC.3 and check if it is equal to 1.



Samples	Touch runtime
300	1ms
150	0.5ms
75	0.25ms

Fig. 24: Sequence diagram of Touch Mode

VDD/4 Mode, also known as ADC Mode, is used to detect the battery voltage. Fig. 25 shows the sequence diagram of VDD/4 Mode. First, enable the LDO and wait 20 us for it to stabilize. Then, enable the ADC. When the start signal (TCHC.3) is set, ADC conversion begins. The runtime of the ADC conversion is approximately 3.3 us. After the conversion is complete, the ADC result can be obtained by reading the values of TCHAMPL.



ADC runtime
3.3 us

Fig. 25: Sequence diagram of VDD/4 Mode

5.15.1. Sample of touch detection program

The following examples show how to use Touch detection with PA0、PA3~PA7.

First, defining the selected pins:

```
PAC    =  0B_0000_0XX0;   //  PA0、PA3~PA7 as input
PAPH   =  0B_0000_0XX0;   //  PA0、PA3~PA7 without pull-high
PADIER =  0B_0000_0XX0;   //  PA0、PA3~PA7 digital input is disabled
```

Next, setting **TCH2C** register, example as below:

```
$   TCH2C PA0, Touch, Gain_x1;      // set PA0 as Touch input
                                    // set Mode = Touch
                                    // set Touch Gain = x1
$   TCH2C PA7, Touch, Gain_x7p5;    // set PA7 as Touch input
                                    // set Mode = Touch
                                    // set Touch Gain = x7.5
```

Next, setting **TCH3C** register, example as below:

```
$   TCH3C LDO_1V92, TF_20kHz, TC_4MHz    // set LDO = 1.92V
                                         // set digital filter pass band = 20kHz
                                         // set Touch clock = 4MHz
$   TCH3C LDO_3V84, TF_40kHz, TC_2MHz    // set LDO = 3.84V
                                         // set digital filter pass band = 40kHz
                                         // set Touch clock = 2MHz
```

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

```
$   DFC ST_300, Gain_x1;     // set samples = 300
                             // set ADC Gain = x1
$   DFC ST_150, Gain_x4;     // set samples = 150
                             // set ADC Gain = x4
```

Next, Enable **LDO**, example as below:

```
$   TCHC En_LDO;             // set Enable LDO
```

Next, delay 20us for LDO stability, example as below:

```
.Delay 8*20;                 // System Clock = 8MHz
.Delay 4*20;                 // System Clock = 4MHz
```

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

```
$   TCHC Enable, En_LDO, En_ADC        // set Enable Touch and Enable ADC
```

Then, start the Touch detection:

```
TCH_Start = 1;               // start Touch detection
.Delay 10;                   // wait 10T for Touch initialization
While(!TCH_Done) NULL;       // wait Touch detection result
```

Finally, it can read Touch result when TCH_Done is high:

```
WORD   Data;                 // two bytes result: TCHAMPH and TCHAMPL
Data$1  =  TCHAMPH
Data$0  =  TCHAMPL
```

The Touch can be disabled by using the following method:

```
$   TCHC Disable, Dis_LDO, Dis_ADC
```

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: reserved
01x: reserved
100: reserved
101: reserved
110: ILRC/4
111: ILRC (default)</td><td>000: IHRC/16
001: IHRC/8
010: ILRC/16
011: IHRC/32
100: IHRC/64
110: reserved
1x1: reserved.</td></tr>
<tr><td>4</td><td>0</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>R/W</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/PRSTB function. 0 / 1: PA5 / PRSTB</td></tr>
</table>

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

Bit	Reset	R/W	Description
7	0	R/W	Enable interrupt from Timer3. 0 / 1: disable / enable
6	0	R/W	Enable interrupt from Timer2. 0 / 1: disable / enable
5	0	R/W	Enable interrupt from LPWM. 0 / 1: disable / enable
4	0	R/W	Enable interrupt from comparator. 0 / 1: disable / enable
3	0	R/W	Enable interrupt from Touch Done. 0 / 1: disable / enable
2	0	R/W	Enable interrupt from Timer16 overflow. 0 / 1: disable / enable
1	-	-	Reserved.
0	0	R/W	Enable interrupt from PA0/PA5. 0 / 1: disable / enable

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

Bit	Reset	R/W	Description
7	-	R/W	Interrupt Request from Timer3, this bit is set by hardware and cleared by software. 0 / 1: No request / Request
6	-	R/W	Interrupt Request from Timer2, this bit is set by hardware and cleared by software. 0 / 1: No request / Request
5	-	R/W	Interrupt Request from LPWM, this bit is set by hardware and cleared by software. 0 / 1: No request / Request
4	-	R/W	Interrupt Request from comparator, this bit is set by hardware and cleared by software. 0 / 1: No request / Request
3	-	R/W	Interrupt Request from Touch Done, this bit is set by hardware and cleared by software. 0 / 1: No request / Request
2	-	R/W	Interrupt Request from Timer16, this bit is set by hardware and cleared by software. 0 / 1: No request / Request
1	-	-	Reserved.
0	-	R/W	Interrupt Request from pin PA0/PA5, 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. Interrupt Edge Select Register (*integs*), IO address = 0x0c

Bit	Reset	R/W	Description
7 - 6	00	WO	GPC 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.
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. Please keep 00.
1 - 0	00	WO	PA0 / PA5 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.

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

Bit	Reset	R/W	Description
7 - 6	11	WO	Enable PA7~PA6 digital input and wake up event. 1 / 0 : enable / disable These bits can be set to low to disable wake up from PA7~PA6 toggling.
5	1	WO	Enable PA5 digital input, wake up event and interrupt request. 1 / 0 : enable / disable This bit can be set to low to disable wake up from PA5 toggling and interrupt request from this pin.
4 - 3	11	WO	Enable PA4~PA3 digital input and wake up event. 1 / 0 : enable / disable These bits can be set to low to disable wake up from PA4~PA3 toggling.
2 - 1	-	-	Reserved, and it is recommended to write 00.
0	1	WO	Enable PA0 digital input, 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.

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

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

6.10. 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.

6.11. 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

6.12. Port A Pull-Low Registers (*papl*), IO address = 0x13

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

6.13. Miscellaneous Register (*misc*), IO address = 0x1b

Bit	Reset	R/W	Description
7 - 6	-	-	Reserved. (Keep 0 for future compatibility)
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 16 ILRC clocks (Not for fast boot-up) 1: Fast wake-up. The wake-up time is 8 ILRC clocks
4 - 3	-	-	Reserved. (Keep 0 for future compatibility)
2	0	WO	Disable LVR function. 0 / 1: Enable / Disable
1 - 0	00	WO	Watch dog time out period. 00: 8k ILRC clock period 01: 16k ILRC clock period 10: 64k ILRC clock period 11: 256k ILRC clock period

6.14. Comparator Control Register (*gpcc*), IO address = 0x1a

Bit	Reset	R/W	Description
7	0	R/W	Enable comparator. 0 / 1 : disable / enable When this bit is set to enable, please also set the corresponding analog input pins to be digital disable to prevent IO leakage.
6	-	RO	Comparator result of comparator. 0: plus input < minus input 1: plus input > minus input
5	0	R/W	Select whether the comparator result output will be sampled by TM2_CLK? 0: result output NOT sampled by TM2_CLK 1: result output sampled by TM2_CLK
4	0	R/W	Inverse the polarity of result output of comparator. 0: polarity is NOT inversed. 1: polarity is inversed.
3 - 1	000	R/W	Selection the minus input (-) of comparator. 000: PA3 001: PA4 010: Internal 1.20V bandgap reference voltage 011: $V_{internal\ R}$ 100: PA6 101: reserved 11X: reserved
0	0	R/W	Selection the plus input (+) of comparator. 0: $V_{internal\ R}$ 1: PA4

6.15. Comparator Selection Register (*gpcs*), IO address = 0x1e

Bit	Reset	R/W	Description
7	0	WO	Comparator output enable (to PA0). 0 / 1: disable / enable
6	0	WO	Wakeup by comparator output enable. (The comparator wakeup effectively when gpcc.6 electrical level changed) 0 / 1: disable / enable
5	0	WO	Selection of high range of comparator.
4	0	WO	Selection of low range of comparator.
3 - 0	0000	WO	Selection the voltage level of comparator. 0000 (lowest) ~ 1111 (highest)

6.16. Timer2 Control Register (*tm2c*), IO address = 0x1c

Bit	Reset	R/W	Description
7 - 4	0000	R/W	Timer2 clock selection. 0000: disable 0001: system clock 0010: internal high RC oscillator (IHRC) 0011: reserved 0100: ILRC 0101: comparator output 011x: reserved 1000: PA0 (rising edge) 1001: ~PA0 (falling edge) 101x: reserved 1100: PA4 (rising edge) 1101: ~PA4 (falling edge)
3 - 2	00	R/W	Timer2 output selection. 00: disable 01: reserved 10: PA3 11: PA4
1	0	R/W	Reserved
0	0	R/W	Enable to inverse the polarity of Timer2 output. 0 / 1: disable / enable

6.17. Timer2 Counter Register (*tm2ct*), IO address = 0x1d

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

Note: Timer2 is designed for Period mode, so do not read tm2ct register.

6.18. Timer2 Scalar Register (*tm2s*), IO address = 0x17

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

6.19. Timer2 Bound Register (*tm2b*), IO address = 0x09

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

6.20. Timer3 Control Register (*tm3c*), IO address = 0x2c

Bit	Reset	R/W	Description
7	-	-	Reserved.
6 - 4	000	R/W	Timer3 clock selection. 000: disable 001: system clock (SYSCLK) 010: internal high RC oscillator (IHRC) 011: reserved 100: ILRC 101: comparator output 110: NILRC 111: reserved
3 - 1			reserved
0	0	R/W	NILRC Enable. 0 / 1: disable / enable

6.21. Timer3 Counter Register (*tm3ct*), IO address = 0x2d

Bit	Reset	R/W	Description
7 - 0	0x00	R/W	Bit [7:0] of Timer3 counter register.

Note: Timer2 is designed for Period mode, so do not read tm2ct register.

6.22. Timer3 Scalar Register (*tm3s*), IO address = 0x2e

Bit	Reset	R/W	Description
7	0	WO	Reserved.
6 - 5	00	WO	Timer3 clock pre-scalar. 00: ÷ 1 01: ÷ 4 10: ÷ 16 11: ÷ 64
4 - 2	-	WO	Reserved.
1 - 0	00	WO	Timer3 clock scalar. 00: ÷ 1 01: ÷ 2 10: reserved 11: ÷ 4

6.23. Timer3 Bound Register (*tm3b*), IO address = 0x2f

Bit	Reset	R/W	Description
7 - 0	0x00	WO	Timer3 bound register.

6.24. LPWMG Control Register (*GPC2PWM*), address= 0x33

Bit	Reset	R/W	Description
7 - 4	-	-	Reserved.
3	-	WO	LPWMG clock source selection 0: IHRC = 16MHz 1: IHRC*2 = 32Mhz
2	-	WO	Enable that comparator controls LPWMG2 output 0/1: disable/enable
1	-	WO	Enable that comparator controls LPWMG1 output 0/1: disable/enable
0	-	WO	Enable that comparator controls LPWMG0 output 0/1: disable/enable

6.25. LPWMG0 Control Register (*LPWMG0C*), address= 0x34

Bit	Reset	R/W	Description
7	-	-	Reserved.
6	-	RO	Output status of LPWMG0 generator.
5	0	WO	Enable to inverse the polarity of LPWMG0 generator output. 0 / 1: disable / enable.
4	0	WO	LPWMG0 output selection. 0: LPWMG0 output 1: LPWMG0 XOR LPWMG1 or LPWMG0 OR LPWMG1 (by bit 0 of LPWMG0C)
3	0	R/W	LPWMG0 output pin selection. 0: none 1: PA3
2 - 1	-	-	Reserved.
0	0	R/W	LPWMG0 output pre-selection. 0: LPWMG0 XOR LPWMG1 1: LPWMG0 OR LPWMG1

6.26. LPWMG1 Control Register (*LPWMG1C*), address = 0x35

Bit	Reset	R/W	Description
7	-	-	Reserved.
6	-	RO	Output status of LPWMG1 generator.
5	0	R/W	Enable to inverse the polarity of LPWMG1 generator output. 0 / 1: disable / enable.
4	0	R/W	LPWMG1 output selection. 0: LPWMG1 1: LPWMG2
3	0	R/W	LPWMG1 output pin selection. 0: none 1: PA4
2 - 0	-	R/W	Reserved.

6.27. LPWMG2 Control Register (*LPWMG2C*), address = 0x36

Bit	Reset	R/W	Description
7	-	-	Reserved.
6	-	RO	Output status of LPWMG2 generator.
5	0	R/W	Enable to inverse the polarity of LPWMG2 generator output. 0 / 1: disable / enable.
4	0	R/W	LPWMG2 output selection. 0: LPWMG2 1: LPWMG2 ÷2
3 - 2	00	R/W	LPWMG2 output pin selection. 00: none 01: PA0 10: PA7 11: reserved
1 - 0	-	R/W	Reserved.

6.28. LPWMG Clock Register (*LPWMGCLK*), address = 0x37

Bit	Reset	R/W	Description
7	0	WO	LPWMG disable/enable. 0: LPWMG disable 1: LPWMG enable
6 - 4	000	WO	LPWMG clock pre-scalar. 000: ÷1 001: ÷2 010: ÷4 011: ÷8 100: ÷16 101: ÷32 110: ÷64 111: ÷128
3 - 1	-	-	Reserved.
0	0	WO	LPWMG clock source selection. 0: system clock 1: IHRC or IHRC*2 (Controlled by GPC2PWM.3)

6.29. LPWMG Counter Upper Bound High Register (*LPWMGCUBH*), address = 0x38

Bit	Reset	R/W	Description
7 - 0	-	WO	Bit[10:3] of LPWMG counter upper bound.

6.30. LPWMG Counter Upper Bound Low Register (*LPWMGCUBL*), address = 0x39

Bit	Reset	R/W	Description
7 - 6	-	WO	Bit[2:1] of LPWMG counter upper bound.
5 - 0	-	-	Reserved.

6.31. LPWMG0/1/2 Duty Value High Register (*LPWMGxDTH*, x=0/1/2), address = 0x3A/0x3C/0x3E

Bit	Reset	R/W	Description
7 - 0	-	WO	Duty values bit[10:3] of LPWMG0/LPWMG1/LPWMG2.

6.32. LPWMG0/1/2 Duty Value Low Register (*LPWMGxDTL*, x=0/1/2), address = 0x3B/0x3D/0x3F

Bit	Reset	R/W	Description
7 - 5	-	WO	Duty values bit [2:0] of LPWMG0/LPWMG1/LPWMG2.
4 - 0	-	-	Reserved.

Note: It's necessary to write *LPWMGxDTL* Register before writing *LPWMGxDTH* Register. (x=0/1/2)

6.33. Touch Control Register (*TCHC*), IO address = 0x20

Bit	Reset	R/W	Description
7	0	R/W	Enable Touch function: 0: Disable; 1: Enable
6	-	-	Reserved
5	0	R/W	Enable Touch LDO 0: Disable; 1: Enable
4	0	R/W	Enable ADC function 0: Disable; 1: Enable
3	-	R/W	Touch process control bit. Write "1" to start conversion Read "1" to indicate the Touch is ready or end of conversion
2	0	R/O	Touch Overflow Flag 0: No Overflow; 1: Overflow
1	-	-	Reserved
0	-	-	Reserved

6.34. Touch Control Register 2 (*TCH2C*), IO address = 0x21

Bit	Reset	R/W	Description	
7 - 5	111	R/W	000	Enable PA0/TK0. 0/1: disable/enable
			001	Enable PA3/TK1. 0/1: disable/enable
			010	Enable PA4/TK2. 0/1: disable/enable
			011	Enable PA5/TK3. 0/1: disable/enable
			100	Enable PA6/TK4. 0/1: disable/enable
			101	Enable PA7/TK5. 0/1: disable/enable
			11x	Touch function Disable
4	0	R/W	0: Touch mode; 1: VDD/4 mode	
3	-	-	Reserved	
2 - 0	000	R/W	Touch Gain 000: Gain = x1 001: Gain = x1.5 010: Gain = x2 011: Gain = x3.5 100: Gain = x5 101: Gain = x7.5 110: Gain = x11 111: Gain = x16.5	

6.35. Touch Control Register 3 (*TCH3C*), IO address = 0x24

Bit	Reset	R/W	Description
7	0	R/W	Touch IP LDO Voltage selection 0: 1.92V; 1: 3.84V
6	-	-	Reserved
5 - 4	00	R/W	Digital Filter pass band 00: 20kHz 01: 40kHz 10: 80kHz 11: 100kHz
3 - 2	00	R/W	Touch clock 00: 4MHz 01: 2MHz 10: 1MHz 11: 0.5MHz
1 - 0	-	-	Reserved

6.36. Touch Amplitude High Byte Register (*TCHAMPH*), IO address = 0x22

Bit	Reset	R/W	Description
7 - 0	-	RO	Touch amplitude value [15:8]

6.37. Touch Amplitude Low Byte Register (*TCHAMPL*), IO address = 0x23

Bit	Reset	R/W	Description
7 - 0	-	RO	Touch amplitude value [7:0]

6.38. Digital filter control Register (*DFC*), IO address = 0x15

Bit	Reset	R/W	Description
7 - 6	-	-	Reserved
5 - 4	00	R/W	Digital Filter samples 00: 300 samples 01: 150 samples 10: 75 samples 11: Reserved
3 - 2	-	-	Reserved
1 - 0	00	R/W	Digital Filter ADC input gain 00: Gain = x1 01: Gain = x2 10: Gain = x4 11: Gain = x8

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)
M.n	Only addressed in 0~0x3F (0~63) 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

swapc IO.n	Swap the nth bit of IO port with carry bit
	Example: swapc IO.0;
	Result: C ← IO.0 , IO.0 ← C
	When IO.0 is a port to output pin, carry C will be sent to IO.0;
	When IO.0 is a port from input pin, IO.0 will be sent to carry C;
	Affected flags: 『N』Z 『Y』C 『N』AC 『N』OV
	Application Example1 (serial output) :
	...
	set1 pac.0 ; // set PA.0 as output
	...
	set0 flag.1 ; // C=0
	swapc pa.0 ; // move C to PA.0 (bit operation), PA.0=0
	set1 flag.1 ; // C=1
	swapc pa.0 ; // move C to PA.0 (bit operation), PA.0=1
	...
	Application Example2 (serial input) :
	...
	set0 pac.0 ; // set PA.0 as input
	...
	swapc pa.0 ; // read PA.0 to C (bit operation)
	src a ; // shift C to bit 7 of ACC
	swapc pa.0 ; // read PA.0 to C (bit operation)
	src a ; // shift new C to bit 7, old C
	...
	-- 	

7.6. Conditional Operation Instructions

ceqsn a, I	Compare ACC with immediate data and skip next instruction if both are equal. Flag will be changed like as (a ← a - I) Example: *ceqsn* a, 0x55 ; *inc* MEM ; *goto* error ; Result: If a=0x55, then "goto error"; otherwise, "inc MEM". Affected flags: 『Y』Z 『Y』C 『Y』AC 『Y』OV
ceqsn a, M	Compare ACC with memory and skip next instruction if both are equal. Flag will be changed like as (a ← a - M) Example: *ceqsn* a, MEM; Result: If a=MEM, skip next instruction Affected flags: 『Y』Z 『Y』C 『Y』AC 『Y』OV
cneqsn a, M	Compare ACC with memory and skip next instruction if both are not equal. Flag will be changed like as (a ← a - M) Example: *cneqsn a, MEM;* Result: If a≠MEM, skip next instruction Affected flags: 『Y』Z 『Y』C 『Y』AC 『Y』OV
cneqsn a, I	Compare ACC with immediate data and skip next instruction if both are no equal. Flag will be changed like as (a ← a - I) Example: *cneqsn a,0x55 ;* *inc MEM ;* *goto error ;* Result: If a≠0x55, then "goto error"; Otherwise, "inc MEM". Affected flags: 『Y』Z 『Y』C 『Y』AC 『Y』OV
t0sn IO.n	Check IO bit and skip next instruction if it's low Example: *t0sn* pa.5; Result: If bit 5 of port A is low, skip next instruction Affected flags: 『N』Z 『N』C 『N』AC 『N』OV
t1sn IO.n	Check IO bit and skip next instruction if it's high Example: *t1sn* pa.5 ; Result: If bit 5 of port A is high, skip next instruction Affected flags: 『N』Z 『N』C 『N』AC 『N』OV
t0sn M.n	Check memory bit and skip next instruction if it's low Example: *t0sn* MEM.5 ; Result: If bit 5 of MEM is low, then skip next instruction Affected flags: 『N』Z 『N』C 『N』AC 『N』OV
t1sn M.n	Check memory bit and skip next instruction if it's high Example: *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 programfrom 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, pcadd, ret, reti , idxm*
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	-	-	-	-	*swapc IO.n*	-	Y	-	-	*ceqsn a, I*	Y	Y	Y	Y
cneqsn a, M	Y	Y	Y	Y										

7.10. BIT definition

Bit access of RAM is only available for address from 0x00 to 0x3F.

8. Code Option Table

Option	Selection	Description
Security	Enable	OTP content is protected and program connot be read back
	Disable	OTP content is not protected so program can be read back
EMI	Disable	Disable EMI optimize option
	Enable	The system clock will be slightly vibrated for better EMI performance
LVR	14 levels	4.5V, 4.0V, 3.75V, 3.5V, 3.3V, 3.15V, 3.0V, 2.7V, 2.5V, 2.4V, 2.3V, 2.2V, 2.1V, 2.0V
Bootup_Time	Normal	Please refer to t_{SBP} in Section 4.1
Interrupt_Src0	PA.0	Inten.0 / Intrq.0 used in interrupt PA.0
	PA.5	Inten.0 / Intrq.0 used in interrupt PA.5

Table 8: Code Option

9. Special Notes

This chapter is to remind user who use PMS160B IC in order to avoid frequent errors upon operation.

9.1. Using IC

9.1.1. IO pin usage and setting

(1) IO pin is set to be digital input

- When IO is 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 change with the voltage, temperature and pin voltage. It is not the fixed value.

(2) IO pin is set to be digital input and enable wakeup function

- Configure IO pin as input
- Set PADIER registers to set the corresponding bit to 1.

(3) PA5 is set to be PRSTB input pin

- Configure PA5 as input
- Set CLKMD.0=1 to enable PA5 as PRSTB 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 >33Ω resistor in between PA5 and the long wire
- Avoid using PA5 as input in such application.

9.1.2. 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.

9.1.3. 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.

- Switch system clock from ILRC to IHRC/8
  CLKMD = 0x3C; // 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

9.1.4. Power down mode, wakeup and watchdog

Watchdog is open by default, but when the program executes ADJUST_IC, the watchdog will be closed. To use the watchdog, you need to reconfigure the open. Watchdog will be inactive once ILRC is disabled.

9.1.5. 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.

9.1.6. 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.

9.1.7. LVR

LVR level selection is done at compile time. User must select LVR based on the system working frequency and power supply voltage to make the MCU work stably.

The following are Suggestions for setting operating frequency, power supply voltage and LVR level:

SYSCLK	VDD	LVR
2MHz	≧ 2.0V	≧ 2.0V
4MHz	≧ 2.5V	≧ 2.5V
8MHz	≧ 3.0V	≧ 3.0V

Table 9: LVR setting for reference

(1) The setting of LVR (2.0V ~ 4.5V) will be valid just after successful power-on process.

(2) User can set MISC.2 as “1” to disable LVR. However, V_{DD} must be kept as exceeding the lowest working voltage of chip; Otherwise, IC may work abnormally.

(3) The LVR function will be invalid when IC in stopexe or stopsys mode.

9.1.8. Programming Writing

There are 6 signals for programming PMS160B: PA3, PA4, PA5, PA6, V_{DD}, and GND.

Please follow the instruction displayed at the software to connect the jumper.

- Special notes about voltage and current while Multi-Chip-Package(MCP) or On-Board Programming
  (1) PA5 (V_{PP}) may be higher than 6.5V.
  (2) V_{DD} may be higher than 9.8V, and its maximum current may reach about 20mA.
  (3) All other signal pins level (except GND) is 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.

9.1.8.1. Using 5S-P-003B to write PMS160B

(1). For 5S-P-003B to write PMS160B-S08A, just use jumper2 and it needs downward four spaces on the Textool. Other packages need to convert the file and use jumper7. Taking the writing of PMS160B-S08B as an example, other packages only need to change the chip package and jumper7 connection in the “package setting” interface. The package settings are shown in Figure 28:

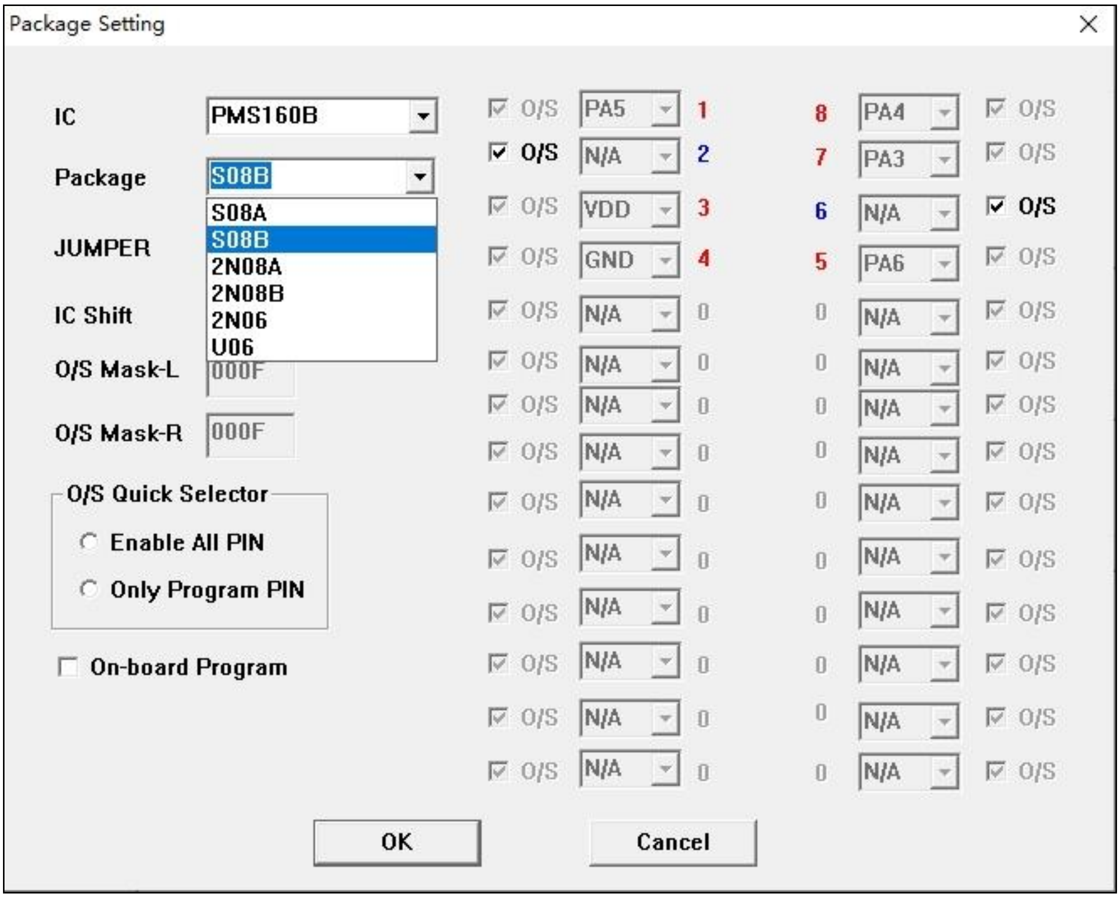


Fig. 29: PMS160B-S08B package setting when using P-003B

(2). As shown in figure 29, it is the Jumper7 connection method.

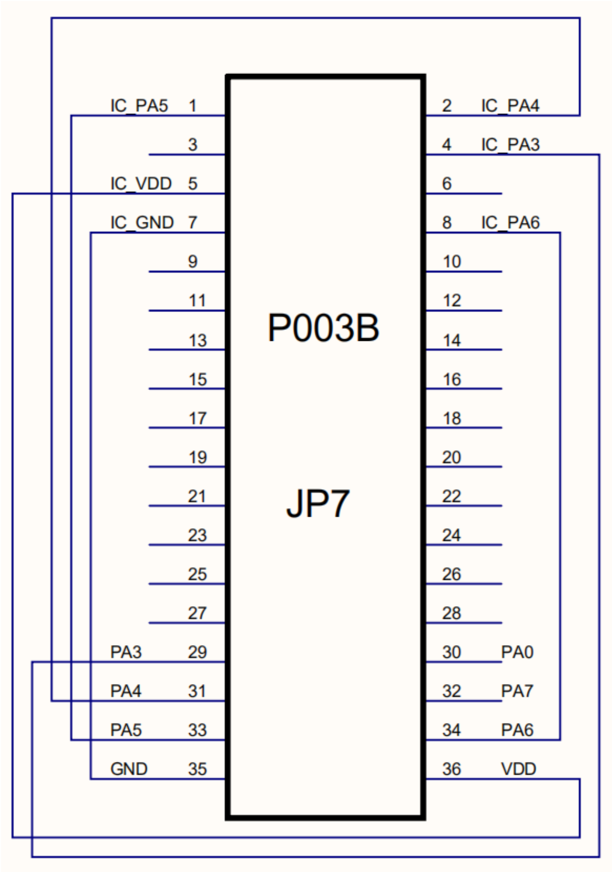


Fig. 30: schematic diagram of PMS160B-S08B Jumper7 when using P-003B

Note: VDD/PA5 DOES NOT need to swap with each other when using P-003B Jumper7.

(3). Insert JP7 and input IC on the socket without shift. After LCDM displays IC ready, it can be written.

9.2. Using ICE

Please note the following items during simulation:

(1) Regarding Simulation Timing

During simulation, 5S-I-S01/2(B) will communicate with the touch simulation board 5S-I-TB003, so the configuration of some registers will be slightly slower than that of the actual IC. The affected registers are as follows:

INTEN/PAPL/TM3C/TM3CT/TM3S/TMSB/TM2C/TM2CT/TM2S/TM2B/COEFH/COEFL/DFC/DFOV/TCHC/TCH2C/TCHAMPH/TCHAMPL/TCH3C/TCHRC/TCHCMP/GPC2PWM/LPWMGCLK/LPWMGCUBH/LPWMGCUBL/LPWMGxC/LPWMGxDTH/LPWMGxDTL, etc.

(2) Regarding Simulation Voltage

The simulation voltage range of this touch panel is limited to 3.5V ~ 4.8V. If used with the 5S-1-S01 standard simulator and the simulator adopts an external power supply for simulation, the external power supply is not recommended to exceed 5V. Otherwise, it may cause failure to simulate or abnormal simulation results due to voltage mismatch between the simulator and the touch panel.

(3) Regarding Each Timer Counter (Timer2/3/16, LPWMG0/1/2)

LPWMG0/1/2 does not support selecting SYSCLK as the counter clock source.

Timer2 and Timer3 do not support selecting GPCRS as the counter clock source.

LPWMG2 and LPWMG0/1 belong to two different ICs for simulation, so there will be slight differences in simulation frequency/phase, and simulation of LPWM function with complementary dead zones is not supported. When using LPWM interrupt, the simulator will additionally add a Code Option: ICE LPWM INTR, for selecting the source of LPWMG interrupt during simulation: from LPWMG0/1 or from LPWMG2.

When simulating the output of LPWM and TM2_PWM (with the clock source being NILRC), the set output pins will be forced to switch to input state (forced switching via the pac register, without affecting the original setting values of registers such as padier and paph). After turning off the LPWM output, the input/output configuration of the corresponding port (pac register) needs to be switched manually by the user. This operation is not required for the actual IC.

(4) Regarding Power Saving and Power Down: stopexe/stopsys

During simulation, the wake-up time of the power saving/power down mode is inconsistent with that of the actual IC, and it is related to the speed of the system clock frequency. Taking a system clock of 1MHz as an example, the simulated wake-up time of stopexe is estimated to be between 30us and 1ms; the wake-up of stopsys takes approximately 700us. If there are requirements for the relevant wake-up time, it is recommended to measure using the actual IC.

Considering the impact of simulation communication on IHRC/ILRC, to avoid false wake-up of the Timer, it is recommended that customers turn off all Timer modules that are not needed for wake-up before using stopexe/stopsys, instead of turning off the Timer clock source. Taking Timer2 as an example, the recommended program writing is: "$ TM2C STOP".

(5) For Simulating PA5 InterruptTo simulate the PA5 interrupt, it is necessary to switch the interrupt source in Code Option to PA5 and manually configure $ MISC3 INT_PA5.

(6) The INTRQ interrupt request bit can be read normally only when the interrupt is enabled.