



PMS160

6 Touch Keys OTP Controller

Datasheet

Version 0.04 – Mar. 4, 2024

Copyright © 2024 by PADAUK Technology Co., Ltd., all rights reserved

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	7
Usage Warning	7
1. Features	8
1.1. Special Features	8
1.2. System Features	8
1.3. CPU Features	8
1.4. Ordering/ Package Information.....	8
2. General Description and Block Diagram	9
3. Pin Definition and Functional Description	10
4. Device Characteristics	13
4.1. DC/AC Characteristics	13
4.2. Absolute Maximum Ratings.....	14
4.3. Typical IHRC Frequency vs. VDD (calibrated to 16MHz).....	14
4.4. Typical ILRC Frequency vs. VDD	15
4.5. Typical NILRC Frequency vs. VDD.....	15
4.6. Typical IHRC Frequency vs. Temperature (calibrated to 16MHz)	16
4.7. Typical ILRC Frequency vs. Temperature	16
4.8. Typical NILRC Frequency vs. Temperature.....	17
4.9. Typical Operating Current vs. VDD and CLK=IHRC/n	17
4.10. Typical Operating Current vs. VDD and CLK=ILRC/n.....	18
4.11. Typical IO pull high resistance.....	18
4.12. Typical IO pull low resistance	19
4.13. Typical IO driving current (I_{OH}) and sink current (I_{OL})	19
4.14. Typical IO input high/ low threshold voltage (V_{IH}/ V_{IL})	20
4.15. Typical power down current (I_{PD}) and power save current (I_{PS})	21
5. Functional Description	22
5.1. Program Memory – OTP	22
5.2. Boot Up.....	22
5.3. Data Memory – SRAM	23

5.4.	Oscillator and clock	23
5.4.1.	Internal High RC oscillator and Internal Low RC oscillator	23
5.4.2.	IHRC calibration	24
5.4.3.	IHRC Frequency Calibration and System Clock.....	24
5.4.4.	System Clock and LVR levels	25
5.4.5.	System Clock Switching.....	26
5.5.	Comparator	27
5.5.1.	Internal reference voltage ($V_{\text{internal R}}$)	28
5.5.2.	Using the comparator	30
5.5.3.	Using the comparator and Bandgap 1.20V	31
5.6.	16-bit Timer (Timer16).....	32
5.7.	Watchdog Timer	33
5.8.	Interrupt.....	34
5.9.	Power-Save and Power-Down	37
5.9.1.	Power-Save mode (“stopexe”)	37
5.9.2.	Power-Down mode (“stopsys”).....	38
5.9.3.	Wake-up	39
5.10.	IO Pins	39
5.11.	Reset	40
5.12.	8-bit Timer (Timer2)	41
5.12.1.	Using the Timer2 to generate periodical waveform	42
5.13.	8-bit Timer (Timer3)	43
5.14.	11-bit SuLED LPWM Generation (LPWMG0/1/2)	44
5.14.1.	LPWM Waveform.....	44
5.14.2.	Hardware Diagram.....	45
5.14.3.	Equations for 11-bit LPWM Generator	46
5.14.4.	Examples of LPWM Waveforms with Complementary Dead Zones	47
5.15.	Touch Function	50
5.15.1.	Sample of touch detection program	52
6.	IO Registers	54
6.1.	ACC Status Flag Register (<i>flag</i>), IO address = 0x00	54
6.2.	Stack Pointer Register (<i>sp</i>), IO address = 0x02.....	54
6.3.	Clock Mode Register (<i>clkmd</i>), IO address = 0x03.....	54
6.4.	Interrupt Enable Register (<i>inten</i>), IO address = 0x04.....	55
6.5.	Interrupt Request Register (<i>intrq</i>), IO address = 0x05	55
6.6.	Timer 16 mode Register (<i>t16m</i>), IO address = 0x06.....	56
6.7.	Interrupt Edge Select Register (<i>integs</i>), IO address = 0x0c.....	56

6.8.	Port A Digital Input Enable Register (<i>padier</i>), IO address = 0x0d	57
6.9.	Port A Data Registers (<i>pa</i>), IO address = 0x10	57
6.10.	Port A Control Registers (<i>pac</i>), IO address = 0x11.....	57
6.11.	Port A Pull-High Registers (<i>paph</i>), IO address = 0x12.....	57
6.12.	Port A Pull-Low Registers (<i>papl</i>), IO address = 0x13.....	57
6.13.	Miscellaneous Register (<i>misc</i>), IO address = 0x1b.....	58
6.14.	Comparator Control Register (<i>gpcc</i>), IO address = 0x1a	58
6.15.	Comparator Selection Register (<i>gpcs</i>), IO address = 0x1e.....	59
6.16.	Timer2 Control Register (<i>tm2c</i>), IO address = 0x1c.....	59
6.17.	Timer2 Counter Register (<i>tm2ct</i>), IO address = 0x1d	59
6.18.	Timer2 Scalar Register (<i>tm2s</i>), IO address = 0x17.....	60
6.19.	Timer2 Bound Register (<i>tm2b</i>), IO address = 0x09	60
6.20.	Timer3 Control Register (<i>tm3c</i>), IO address = 0x2c.....	60
6.21.	Timer3 Counter Register (<i>tm3ct</i>), IO address = 0x2d	60
6.22.	Timer3 Scalar Register (<i>tm3s</i>), IO address = 0x2e.....	61
6.23.	Timer3 Bound Register (<i>tm3b</i>), IO address = 0x2f	61
6.24.	LPWMG Control Register (<i>GPC2PWM</i>), address= 0x33	61
6.25.	LPWMG0 Control Register (<i>LPWMG0C</i>), address= 0x34.....	62
6.26.	LPWMG1 Control Register (<i>LPWMG1C</i>), address = 0x35.....	62
6.27.	LPWMG2 Control Register (<i>LPWMG2C</i>), address = 0x36.....	63
6.28.	LPWMG Clock Register (<i>LPWMGCLK</i>), address = 0x37	63
6.29.	LPWMG Counter Upper Bound High Register (<i>LPWMGCUBH</i>), address = 0x38	64
6.30.	LPWMG Counter Upper Bound Low Register (<i>LPWMGCUBL</i>), address = 0x39	64
6.31.	LPWMG0/1/2 Duty Value High Register, address = 0x3A/0x3C/0x3E	64
6.32.	LPWMG0/1/2 Duty Value Low Register, address = 0x3B/0x3D/0x3F	64
6.33.	Touch Control Register 2 (<i>IFC2C</i>), IO address = 0x20	64
6.34.	Touch Control Register (<i>IFCC</i>), IO address = 0x21	65
6.35.	Touch Key Counter High Register (<i>IFCCRH</i>), IO address = 0x22	65
6.36.	Touch Key Counter Low Register (<i>IFCCRL</i>), IO address = 0x23.....	65
6.37.	LDO Control Register (<i>IFCLDO</i>), IO address = 0x24.....	65
6.38.	Touch capacitor charge/discharge setting register(<i>CHDIS</i>) , IO address = 0x25.....	66
6.39.	Touch capacitor control register(<i>EXCAP</i>) , IO address = 0x26.....	66
7.	Instructions.....	67
7.1.	Data Transfer Instructions	68

7.2.	Arithmetic Operation Instructions	70
7.3.	Shift Operation Instructions	72
7.4.	Logic Operation Instructions.....	73
7.5.	Bit Operation Instructions	75
7.6.	Conditional Operation Instructions.....	76
7.7.	System control Instructions	77
7.8.	Summary of Instructions Execution Cycle	78
7.9.	Summary of affected flags by Instructions	79
7.10.	BIT definition	79
8.	Code Option Table.....	80
9.	Special Notes	81
9.1.	Using IC	81
9.1.1.	IO pin usage and setting	81
9.1.2.	Interrupt	81
9.1.3.	System clock switching.....	82
9.1.4.	Power down mode, wakeup and watchdog.....	82
9.1.5.	TIMER time out.....	82
9.1.6.	IHRC.....	82
9.1.7.	LVR	83
9.1.8.	Programming Writing	83
9.2.	Usage of ICE.....	88

Revision History

Revision	Date	Description
0.03	2023/02/14	1. Amend "IMPORTANT NOTICE"
0.04	2024/03/04	1. Updated description of Using Warning, Reset and Watchdog. 2. Modify the value of Scalar.

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.

<http://www.padauk.com.tw/en/product/show.aspx?num=162&kw=PMS160>

(The following picture are for reference only.)

◆◆ PMS160 ◆◆

- ◆ Not supposed to use in AC RC step-down powered or high EFT requirement applications
- ◆ Operating temperature : -20°C ~ 70°C

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		
APN015	Capacitive touch screen PCB design guide		
APN019	E-PAD PCB layout guideline		

1. Features

1.1. Special Features

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

1.2. System Features

- ◆ 1.5KW OTP program memory
- ◆ 96 Bytes data RAM
- ◆ Maximum 6 IO pins can be selected as TOUCH PAD individually
- ◆ 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

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

- ◆ PMS160-S08A: SOP8 (150mil)
- ◆ PMS160-S08B: SOP8 (150mil)
- ◆ PMS160-2N08A: DFN8(2*2mm)
- ◆ PMS160-2N08B: DFN8(2*2mm)
- ◆ PMS160-2N06: DFN6(2*2mm)
- ◆ PMS160-U06A: SOT23-6 (60mil)
- Please refer to the official website file for package size information: "Package information "

2. General Description and Block Diagram

The PMS160 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 PMS160. Besides, PMS160 also includes 1.5KW OTP program memory, 96 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).

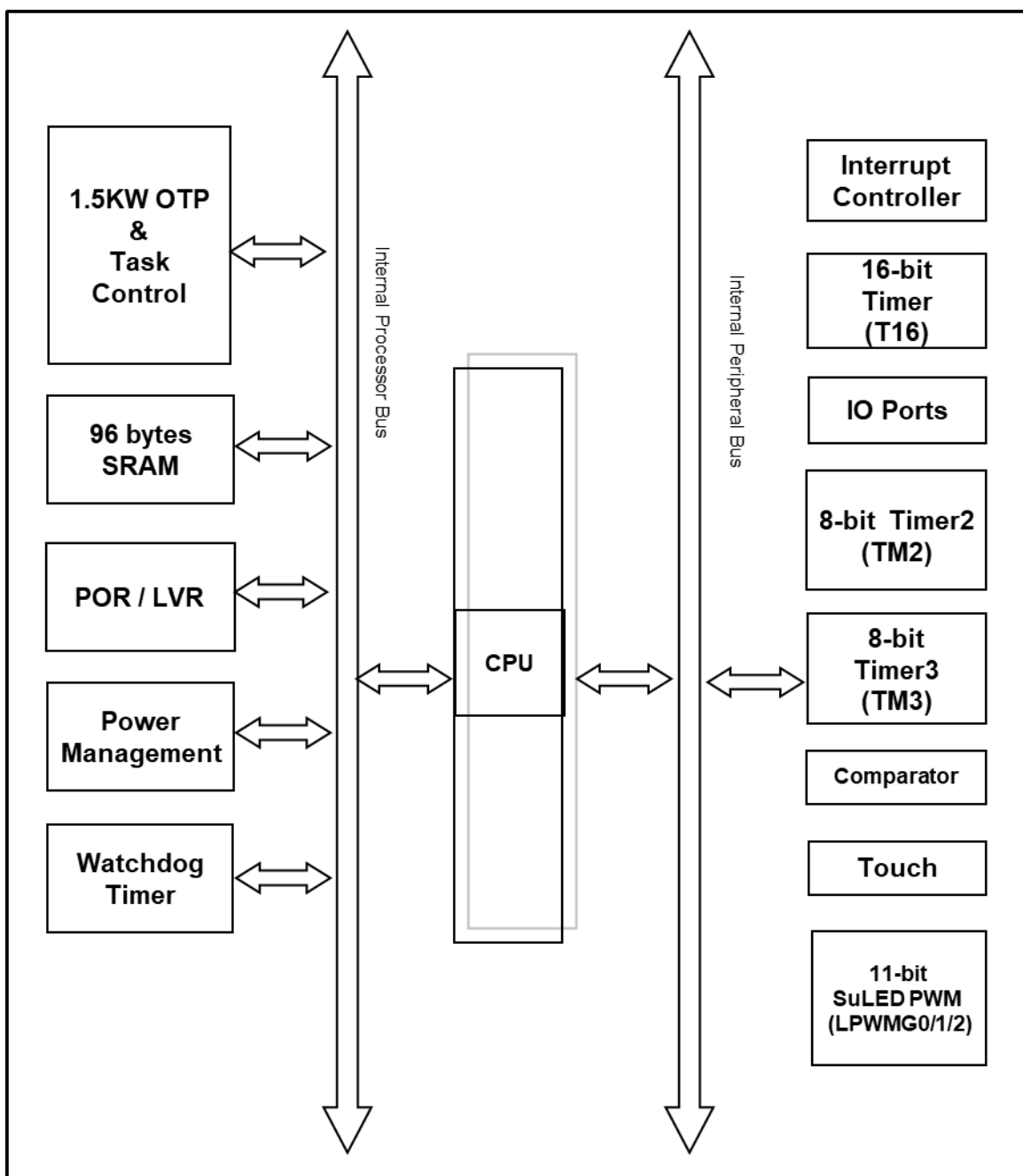
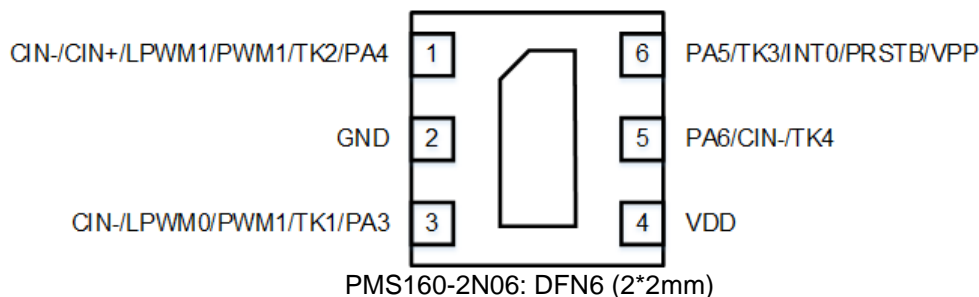
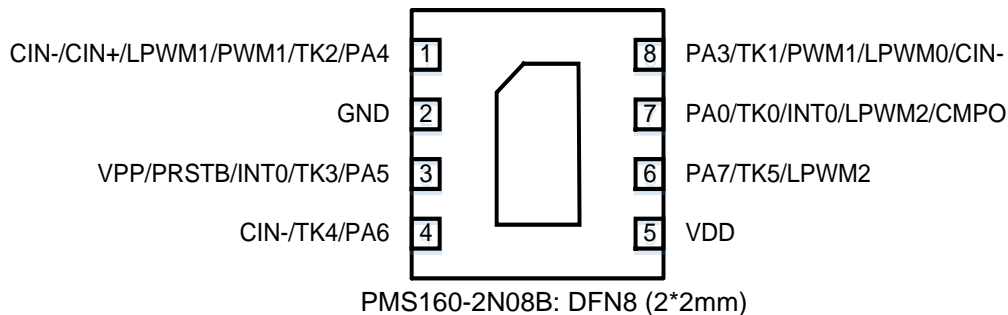
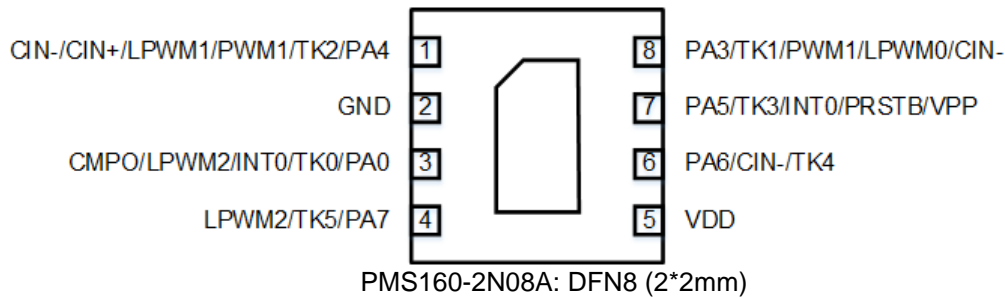
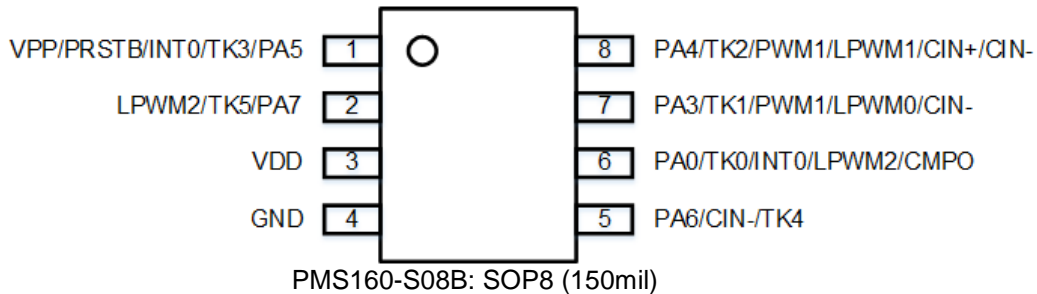
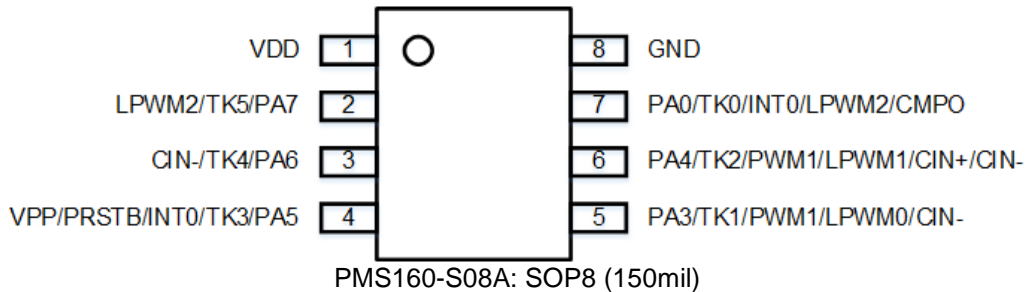


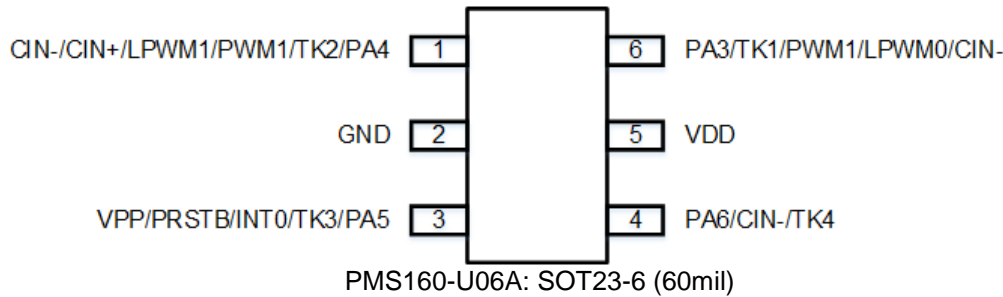
Fig. 1: PMS160 Block Diagram

3. Pin Definition and Functional Description



PMS160

6 Touch Keys OTP Controller



Pin Name	Pin & Buffer Type	Description
PA6 / TK4 / CIN-	IO ST / CMOS / Analog	<p>This pin can be used as:</p> <ul style="list-style-type: none"> (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. <p>When this pin is configured as analog input, please use bit 6 of register padier to disable the digital input to prevent leakage current.</p>
PA5 / TK3 / INT0 / PRSTB / VPP	IO ST / CMOS	<p>This pin can be used as:</p> <ul style="list-style-type: none"> (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. <u>Both rising edge and falling edge are accepted to request interrupt service.</u> (4) External reset pin (5) VPP for OTP programming <p>This pin can be used to wake-up system during sleep mode; however, wake-up function is also disabled if bit 5 of padier register is "0". <u>Please put 33Ω resistor in series to have high noise immunity when this pin is in input mode.</u></p>
PA4 / TK2 / PWM1 / LPWM1 / CIN+ / CIN-	IO ST / CMOS / Analog	<p>This pin can be used as:</p> <ul style="list-style-type: none"> (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. <p>When this pin is configured as analog input, please use bit 4 of register padier to disable the digital input to prevent leakage current.</p>

PMS160

6 Touch Keys OTP Controller

Pin Name	Pin & Buffer Type	Description
PA3 / TK1 / PWM1 / LPWM0 / CIN-	IO ST / CMOS / Analog	<p>This pin can be used as:</p> <ul style="list-style-type: none"> (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. <p>When this pin is configured as analog input, please use bit 3 of register padier to disable the digital input to prevent leakage current.</p>
PA0 / TK0 / INT0 / LPWM2 / CMPO	IO ST / CMOS / Analog	<p>This pin can be used as:</p> <ul style="list-style-type: none"> (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. <u>Both rising edge and falling edge are accepted to request interrupt service.</u> (4) Channel 2 output of LPWM (5) Output result of comparator. <p>When this pin is configured as analog input, please use bit 0 of register padier to disable the digital input to current leakage current.</p>
PA7 / TK5 / LPWM2	IO ST / CMOS	<p>This pin can be used as:</p> <ul style="list-style-type: none"> (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 <p>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.</p>
VDD	VDD	VDD: Digital positive power
GND	GND	GND: Digital negative power
<p>Notes: IO: Input/Output; ST: Schmitt Trigger input; Analog: Analog input pin; CMOS: CMOS voltage level</p>		

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	2.0 [#]		5.5	V	#Subject to LVR tolerance
LVR%	Low Voltage Reset Tolerance	-5		5	%	
f_{SYS}	System clock (CLK)* = IHRC/2	0		8M	Hz	$V_{DD} \geq 3.0V$, No IFC Touch $V_{DD} \geq 2.5V$, No IFC Touch $V_{DD} \geq 2.0V$, No IFC Touch $V_{DD} = 5V$
	IHRC/4	0		4M		
	IHRC/8	0		2M		
	ILRC		46K			
	IHRC/16	0		1M	Hz	Maximum system frequency when IFC Touch conversion
V_{POR}	Power on reset		1.8		V	
I_{OP}	Operating Current		0.6		mA	$f_{SYS}=IHRC/16=1MIPS@5.0V$ $f_{SYS}=ILRC=46KHz@5.0V$
			37		uA	
I_{PD}	Power Down Current (by <i>stopsys</i> command)		0.2		uA	$V_{DD} = 5V$ $V_{DD} = 3.3V$
			0.12			
I_{PS}	Power Save Current (by <i>stopexe</i> 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)		18		mA	$V_{DD}=5.0V$, $V_{OL}=0.5V$
I_{OH}	IO lines drive current (Normal)		13		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	$V_{DD} + 0.3 \geq V_{IN} \geq -0.3$
R_{PH}	Pull-high Resistance		71		K Ω	$V_{DD}=5.0V$
R_{PL}	Pull-low Resistance		64		K Ω	$V_{DD}=5.0V$
f_{IHRC}	Frequency of IHRC after calibration *	15.76*	16*	16.24*	MHz	25°C, $V_{DD} = 2.2V \sim 5.5V$
		15.20*	16*	16.80*		$V_{DD} = 2.2V \sim 5.5V$, -20°C < T_a < 70°C*
		14.60*	16*	17.40*		$V_{DD} = 2.0V \sim 5.5V$, -20°C < T_a < 70°C
f_{ILRC}	Frequency of ILRC *		46		KHz	$V_{DD} = 5V$
f_{NILRC}	Frequency of NILRC *		14		KHz	$V_{DD} = 5.0V$
t_{INT}	Interrupt pulse width	30			ns	$V_{DD} = 5.0V$
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
	System boot-up period from power-on for Fast boot-up		720		us	@ V _{DD} =5V
t _{RST}	External reset pulse width	120			us	@ V _{DD} =5V

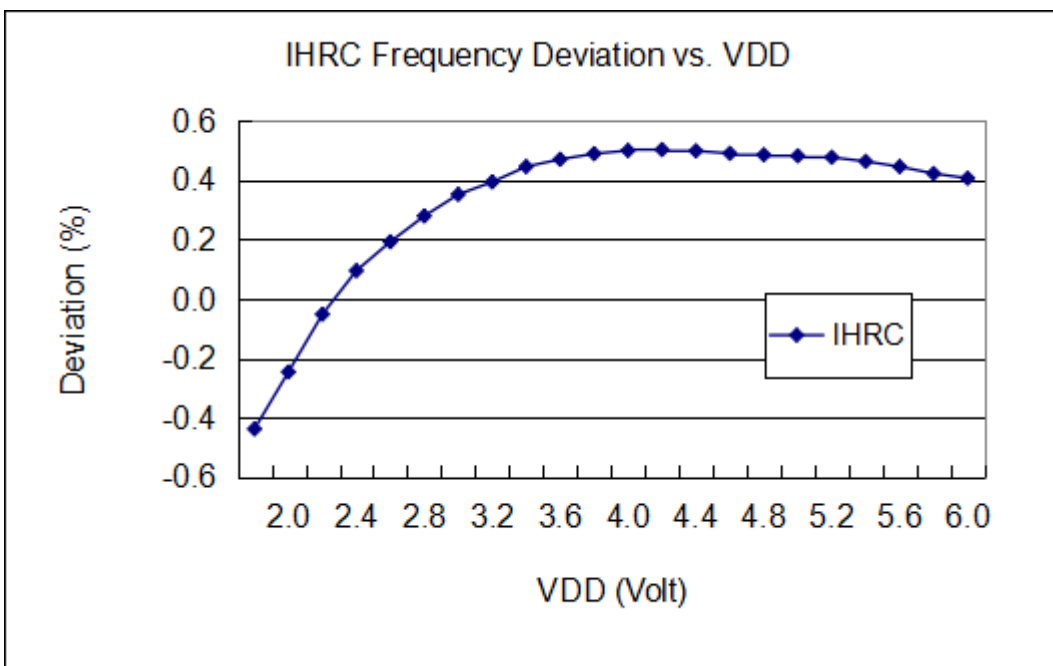
*These parameters are for design reference, not tested for every chip.

*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.

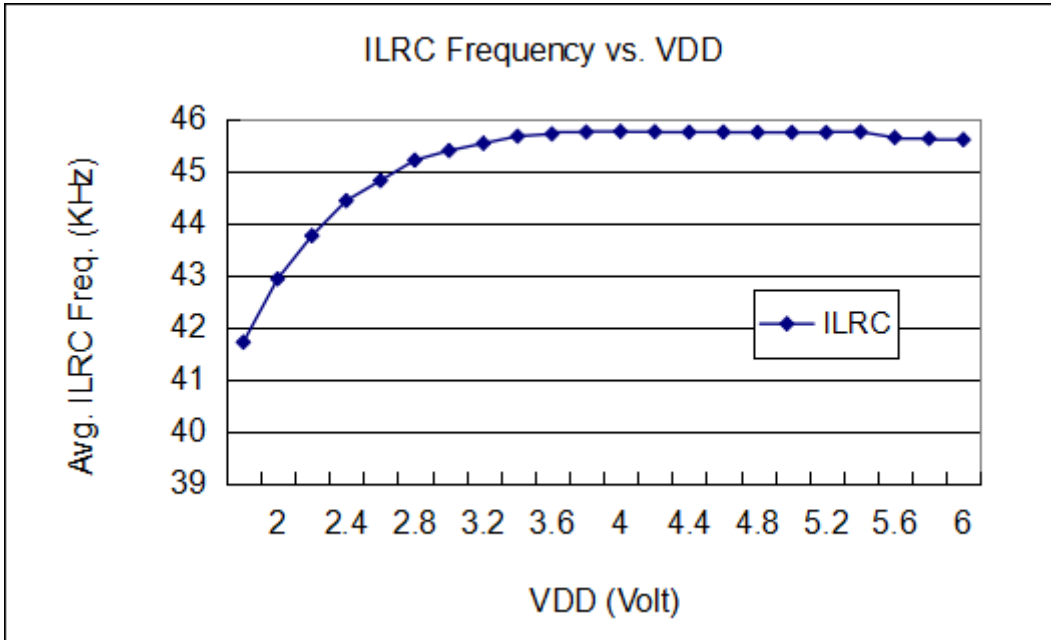
4.2. Absolute Maximum Ratings

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

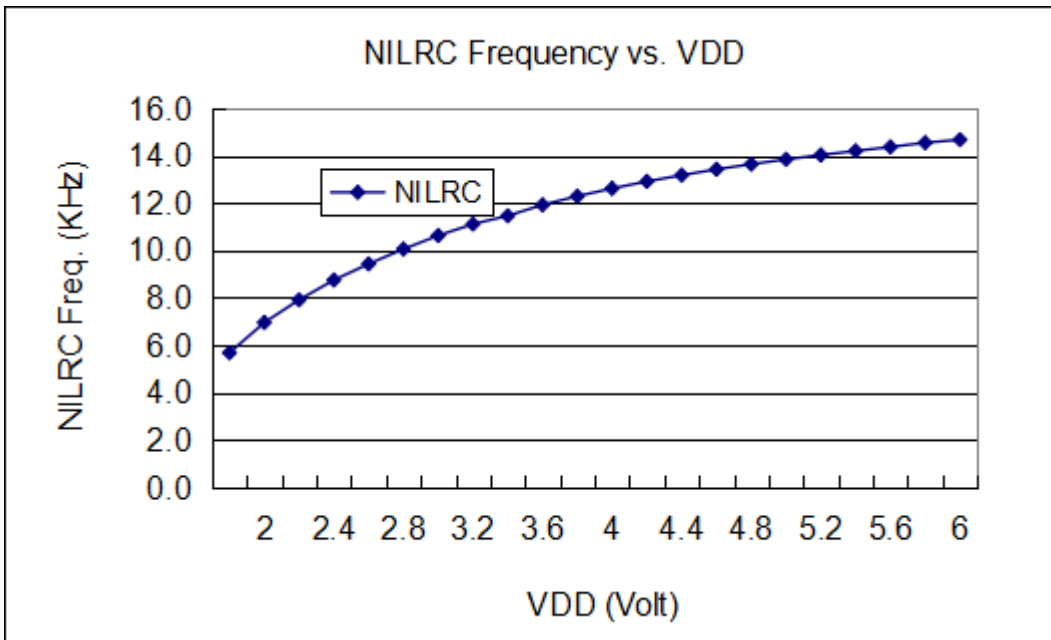
4.3. Typical IHRC Frequency vs. VDD (calibrated to 16MHz)



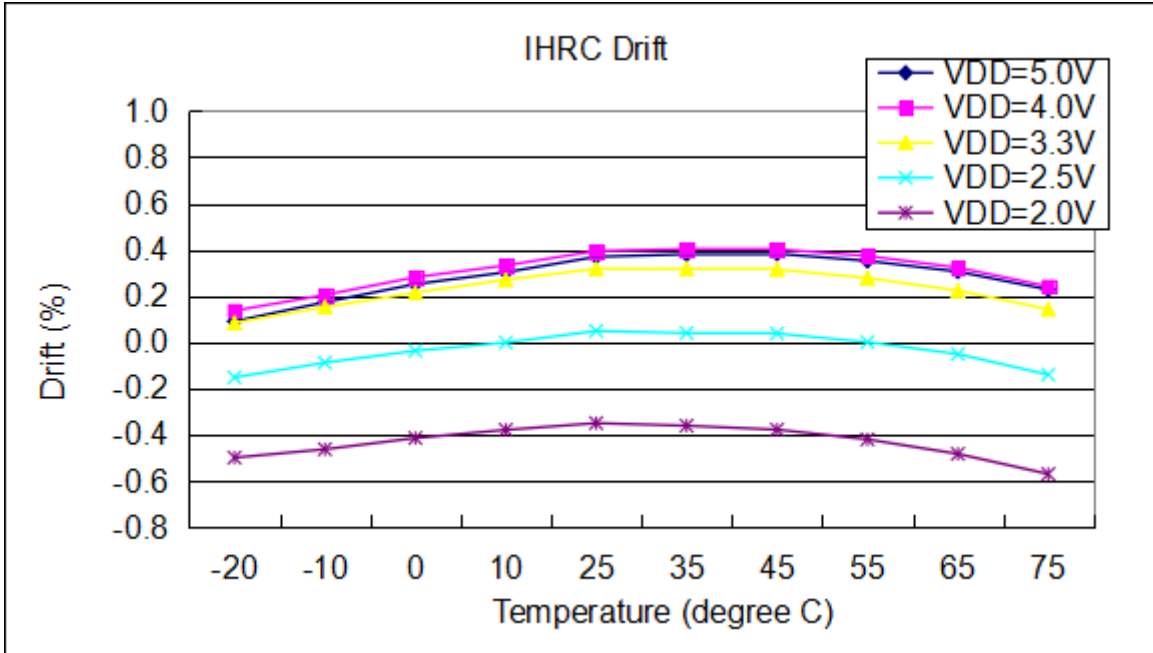
4.4. Typical ILRC Frequency vs. VDD



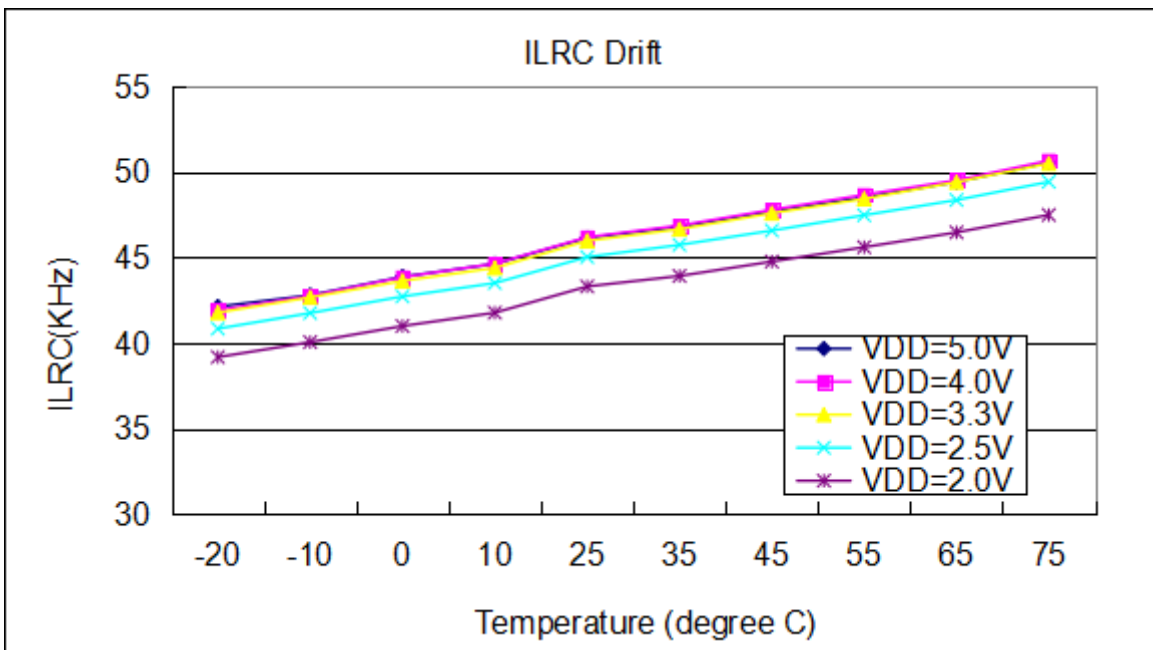
4.5. Typical NILRC Frequency vs. VDD



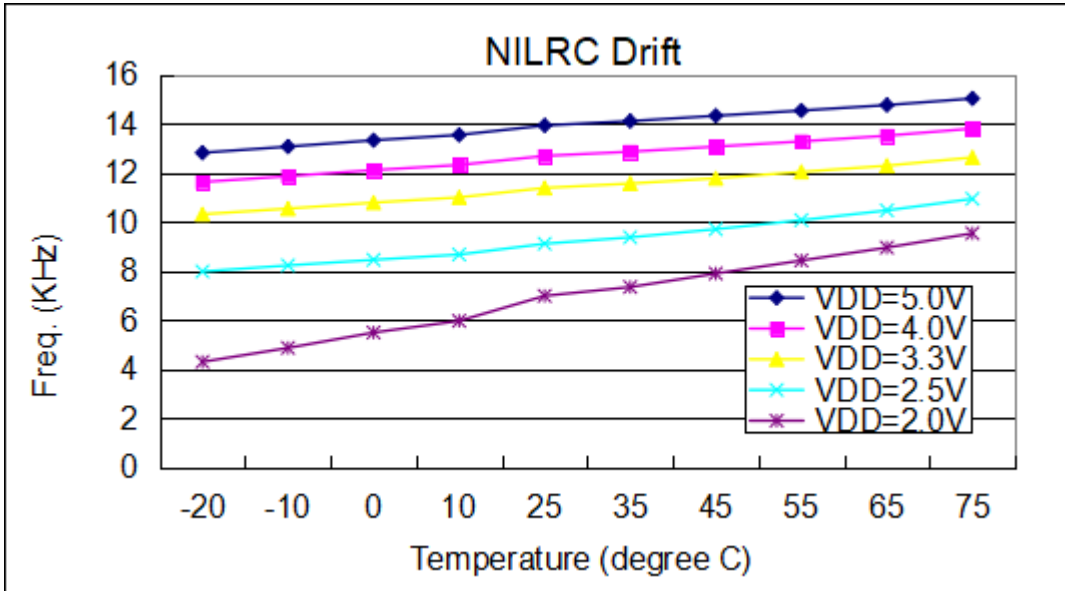
4.6. Typical IHRC Frequency vs. Temperature (calibrated to 16MHz)



4.7. Typical ILRC Frequency vs. Temperature

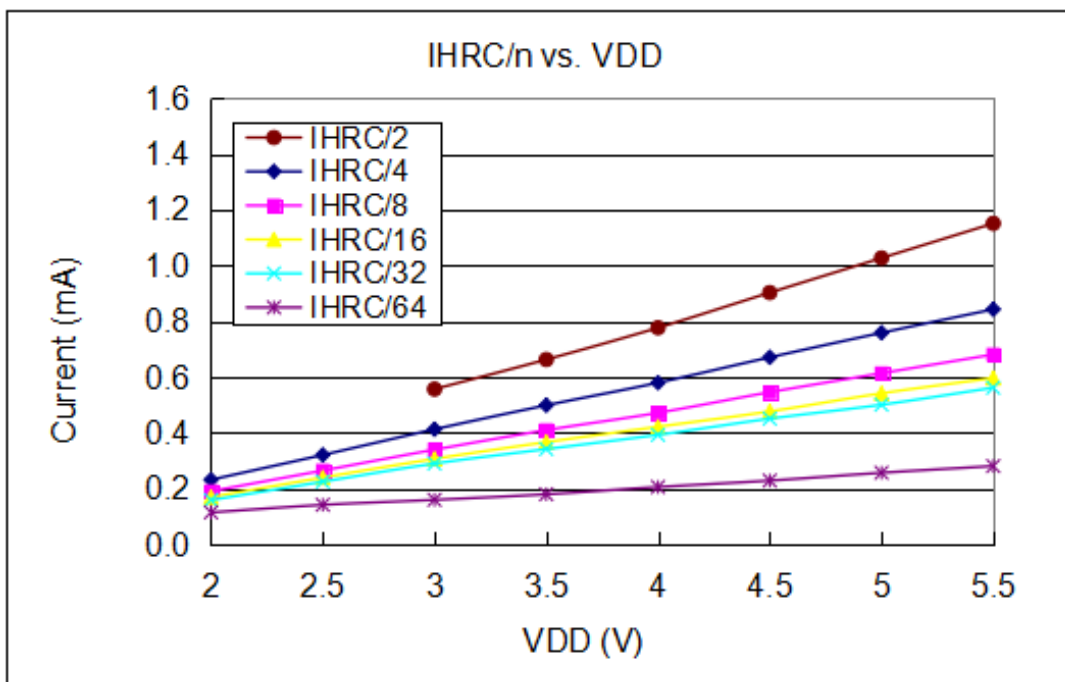


4.8. Typical NILRC Frequency vs. Temperature



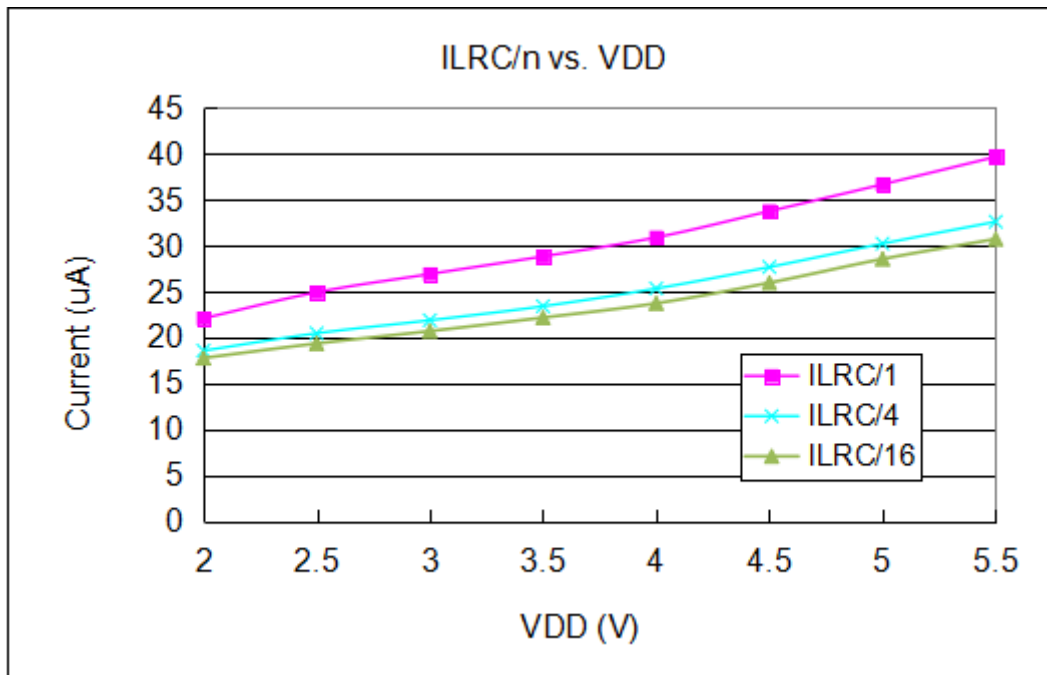
4.9. Typical Operating Current vs. VDD and CLK=IHRC/n

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

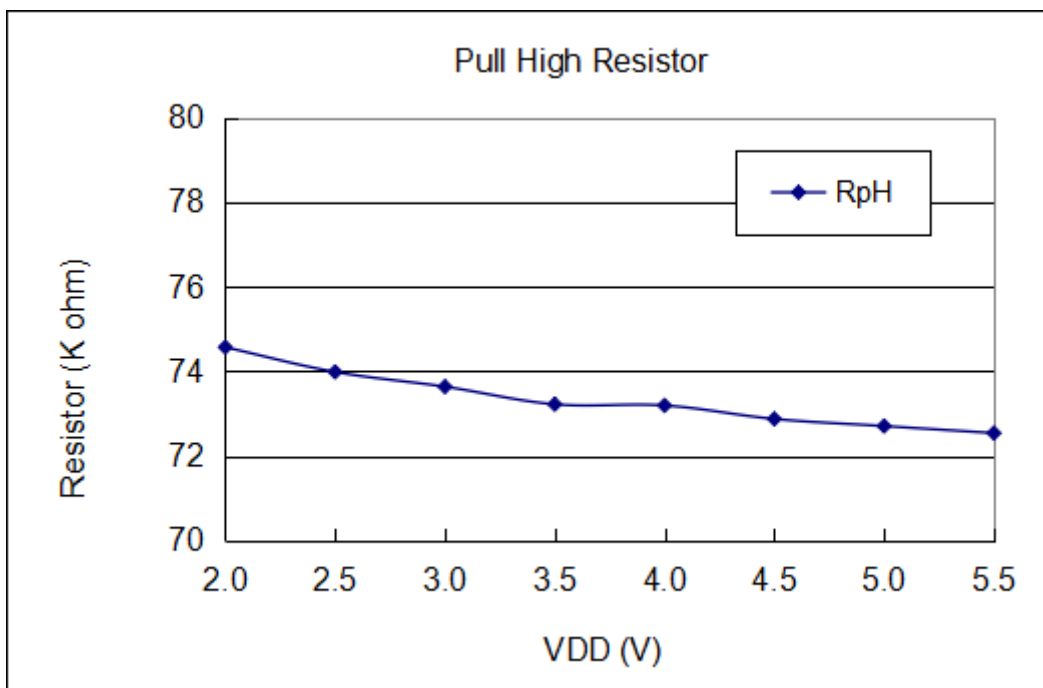


4.10. Typical Operating Current vs. VDD and CLK=ILRC/n

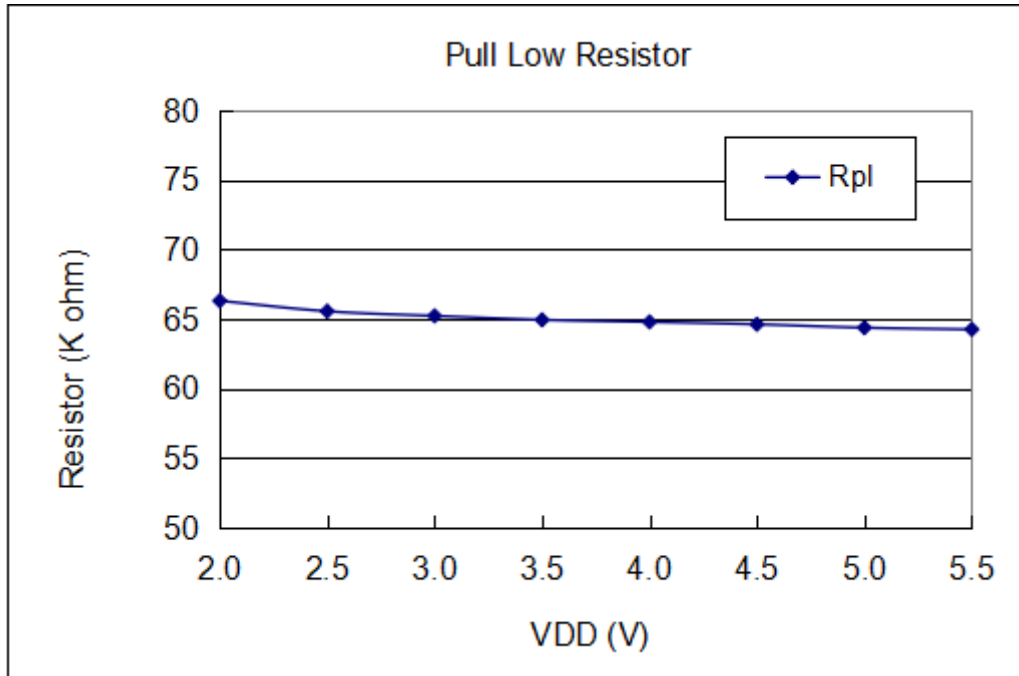
- Conditions:
 - tog pa0(1s), **ON**: Bandgap, LVR, IHRC
 - no t16m, no interrupt, no floating IO pins, disable ILRC, **Touch**: Disable



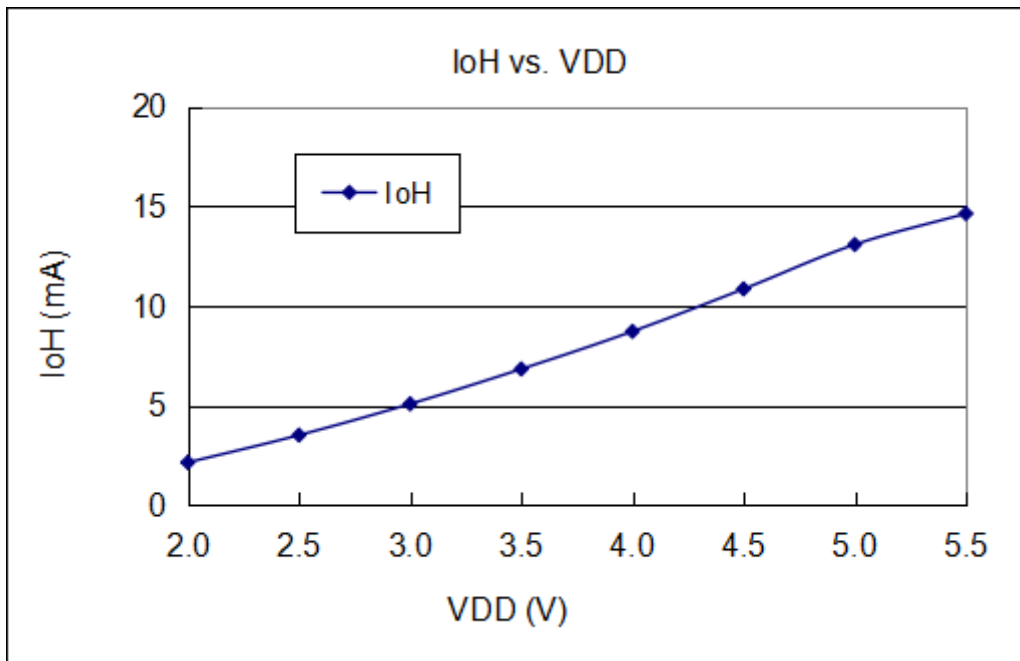
4.11. Typical IO pull high resistance

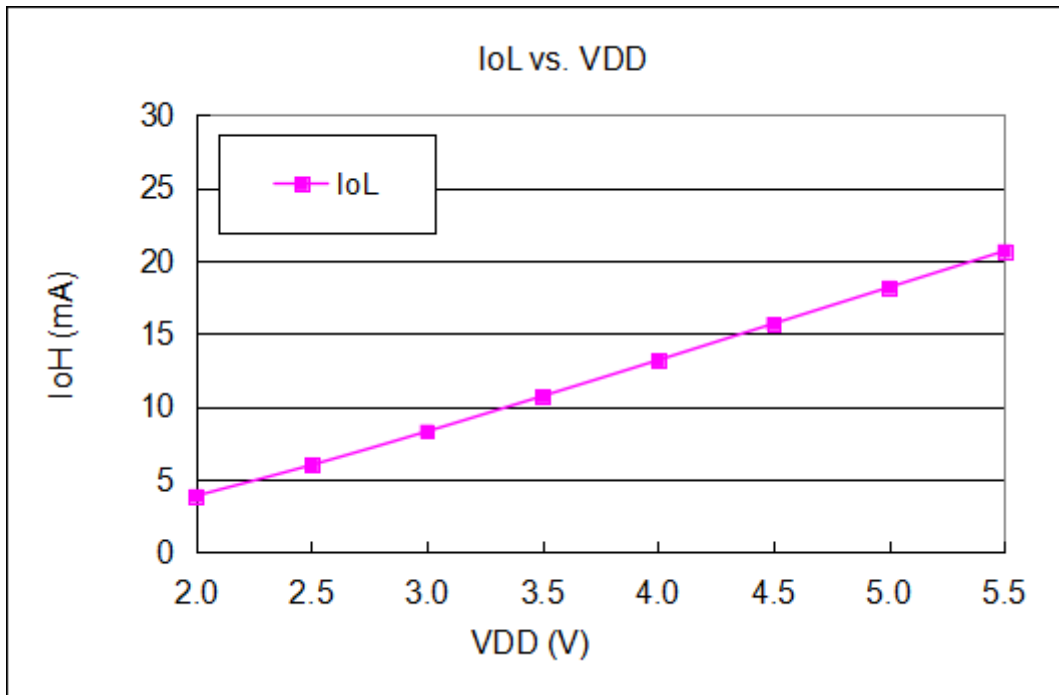


4.12. Typical IO pull low resistance

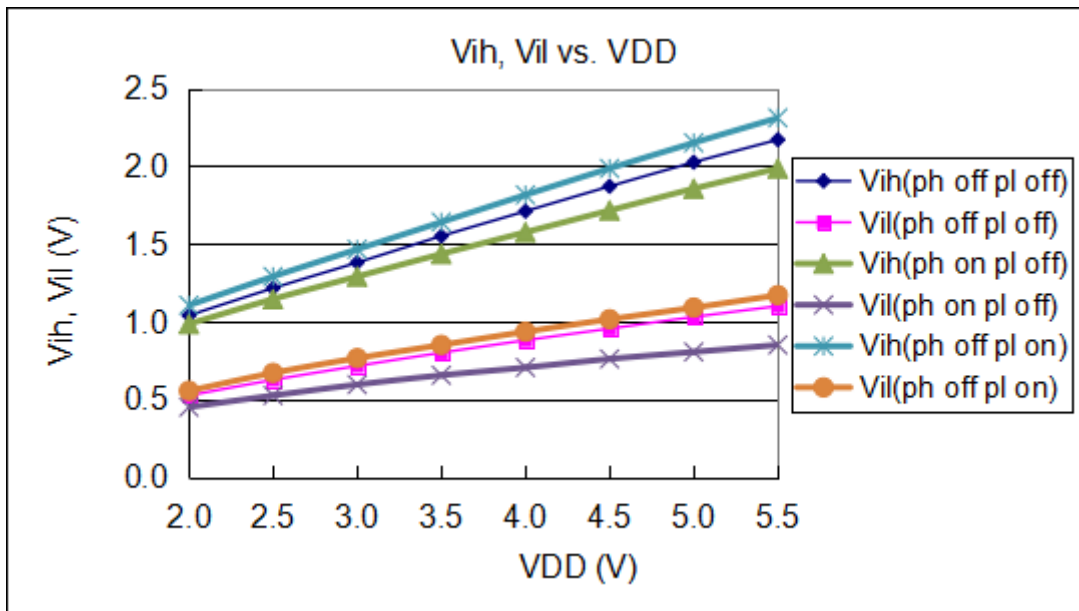


4.13. Typical IO driving current (I_{OH}) and sink current (I_{OL}) ($V_{OH}=0.9*VDD$, $V_{OL}=0.1*VDD$)

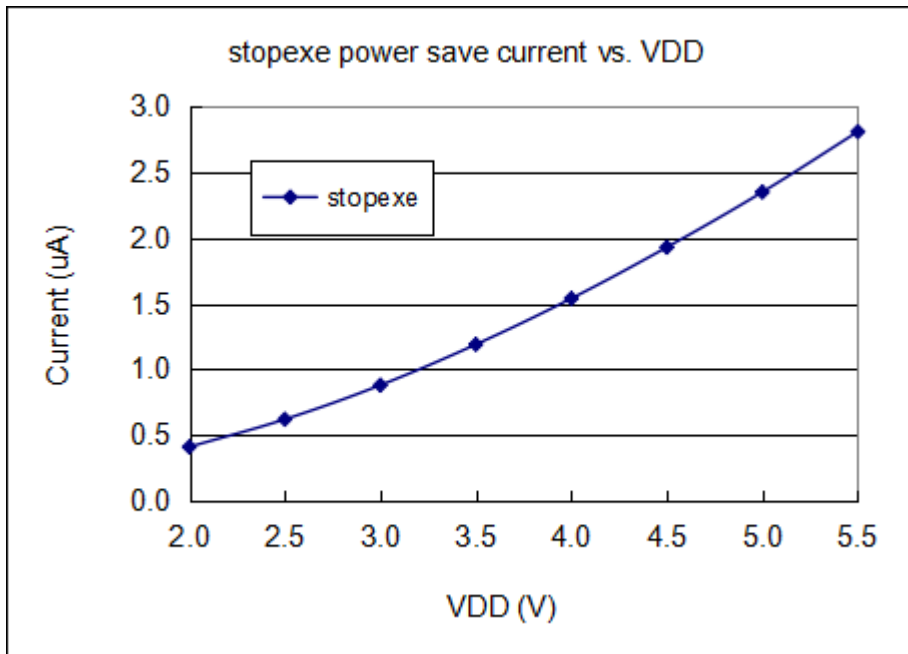
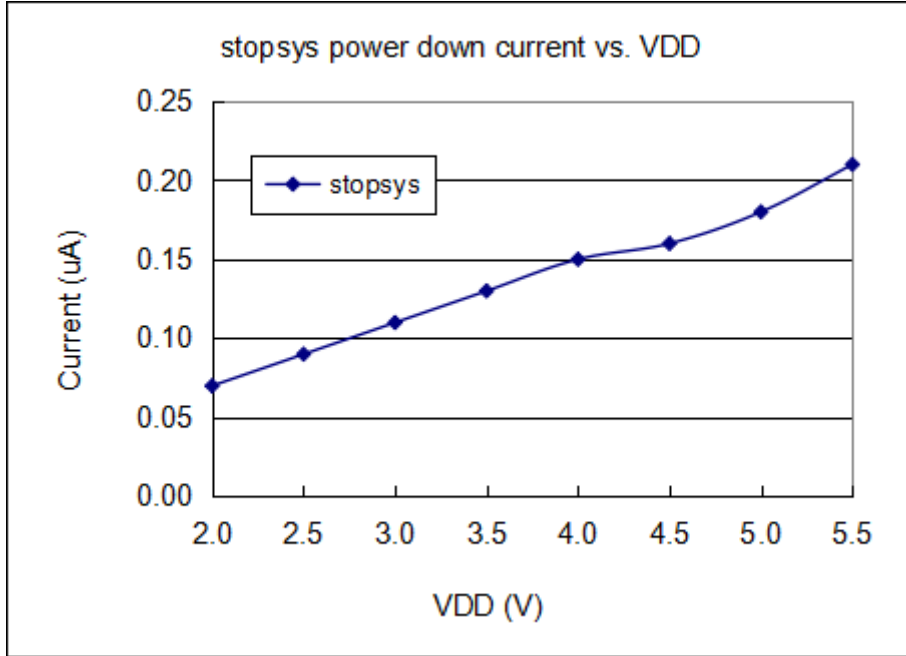




4.14. Typical IO input high/ low threshold voltage (V_{IH} / V_{IL})



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 PMS160 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.

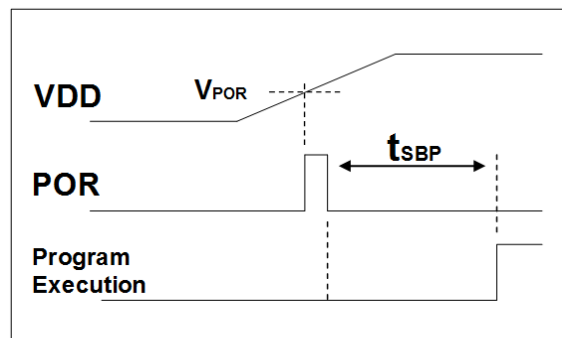
Address	Function
0x000	FPP0 reset – goto instruction
0x001	User program
•	•
•	•
0x00F	User program
0x010	Interrupt entry address
0x011	User program
•	•
0x5EF	User program
0x5F0	System Using
•	•
0x5FF	System Using

Table 1: Program Memory Organization

5.2. Boot Up

POR (Power-On-Reset) is used to reset PMS160 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 96 bytes data memory of PMS160 can be accessed by indirect access mechanism.

5.4. Oscillator and clock

There are three oscillator circuits provided by PMS160: 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	<code>clkmd.4</code>
ILRC	<code>clkmd.2</code>
NILRC	<code>tm3c.0</code>

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, PMS160 provide the IHRC frequency calibration to eliminate this variation, and this function can be selected when compiling user's program and the command will be inserted into user's program automatically. The calibration command is shown as below:

```
.ADJUST_IC      SYSCLK=IHRC/(p1), IHRC=(p2)MHz, VDD=(p3)V
```

Where,

p1=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 starting the system. The program code for IHRC frequency calibration is executed only one time that occurs in writing the codes into OTP memory; after then, it will not be executed again. If the different option for IHRC calibration is chosen, the system status is also different after boot. The following shows the status of PMS160 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_IC SYSCLK=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, VDD=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, VDD=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 PMS160 is shown as Fig. 3.

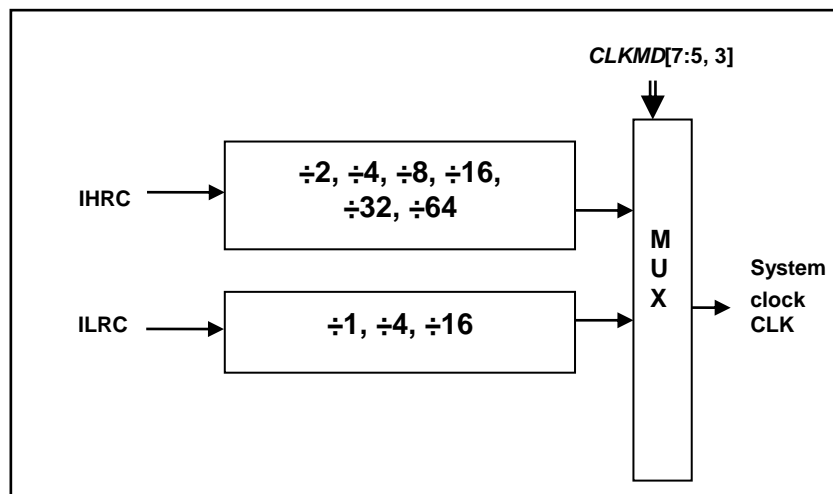


Fig. 3: Options of System Clock

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 PMS160 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.

Case 1: 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
...

```

Case 2: 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
...

```

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

```

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

```

Case 4: 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 PMS160; Fig.4 shows its hardware diagram. It can compare signals between two pins or with either internal reference voltage $V_{\text{internal R}}$ 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 $V_{\text{internal R}}$ selected by bit [3:1] of gpcc register, and the plus input of comparator can be PA4 or $V_{\text{internal R}}$ 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.

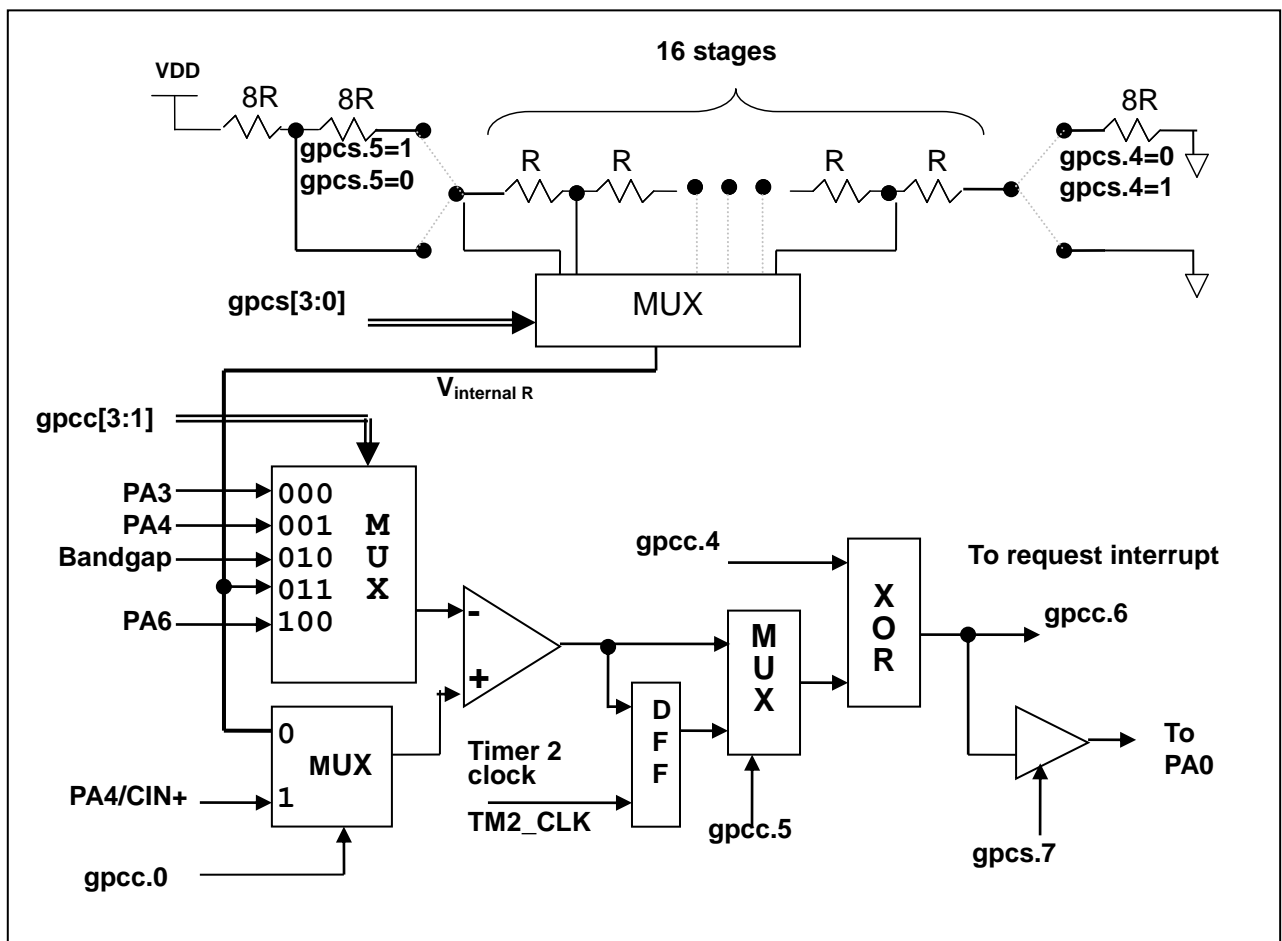


Fig.4: Hardware diagram of comparator

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

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

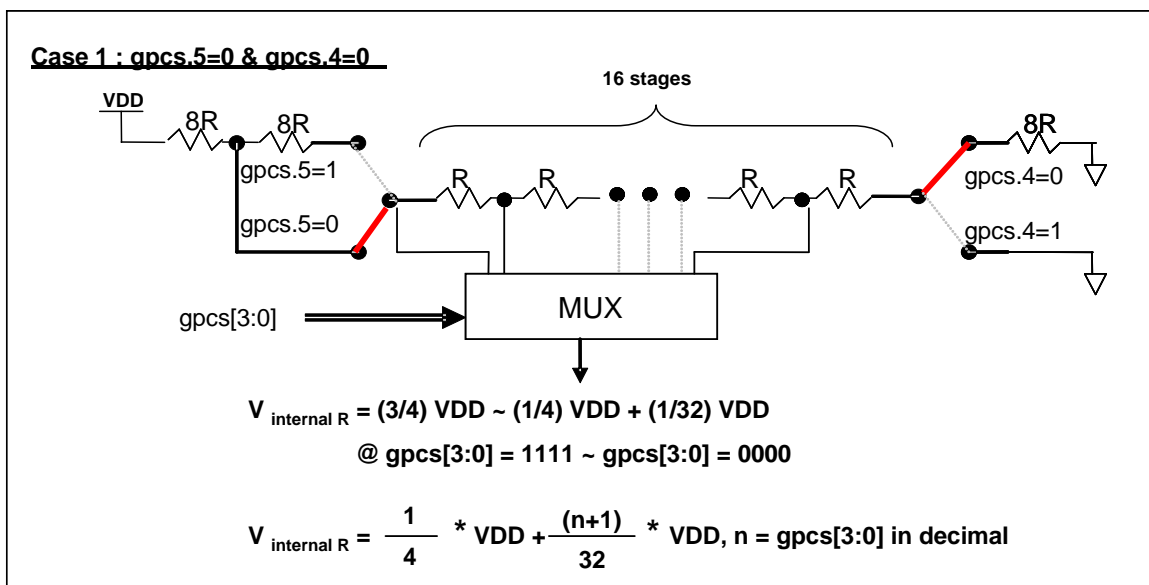


Fig.5: $V_{\text{internal R}}$ hardware connection if gpcs.5=0 and gpcs.4=0

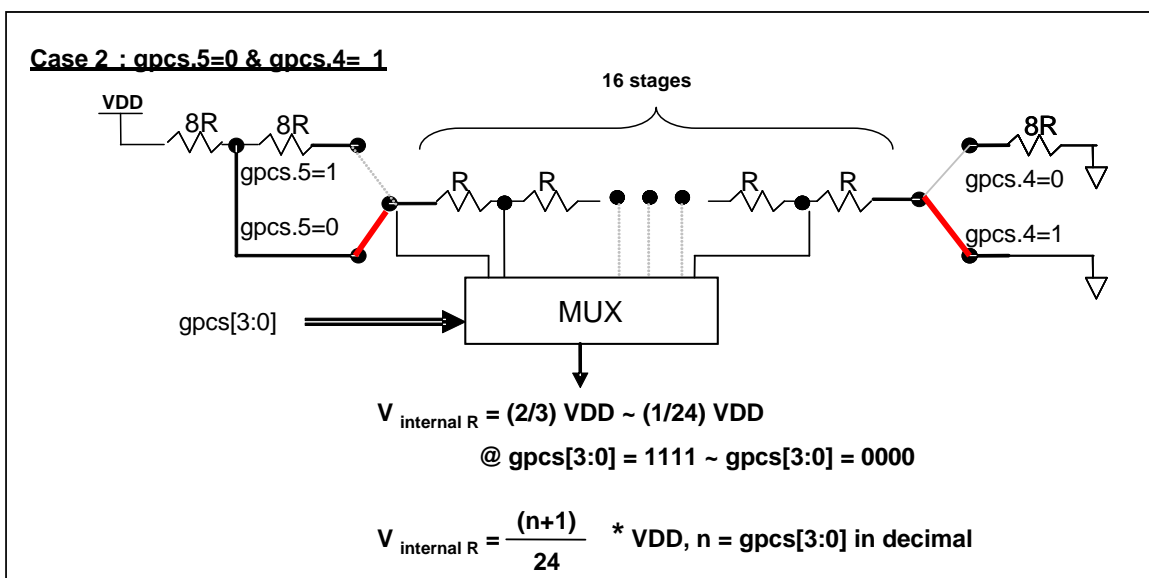


Fig.6: $V_{\text{internal R}}$ hardware connection if gpcs.5=0 and gpcs.4=1

PMS160

6 Touch Keys OTP Controller

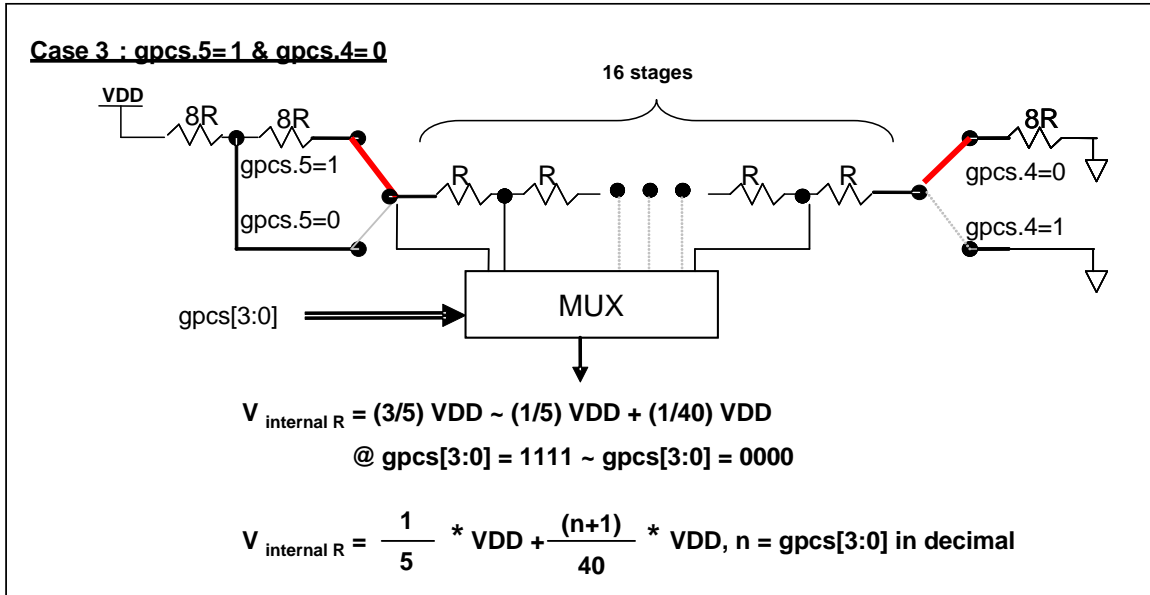


Fig.7: $V_{\text{internal } R}$ hardware connection if gpcs.5=1 and gpcs.4=0

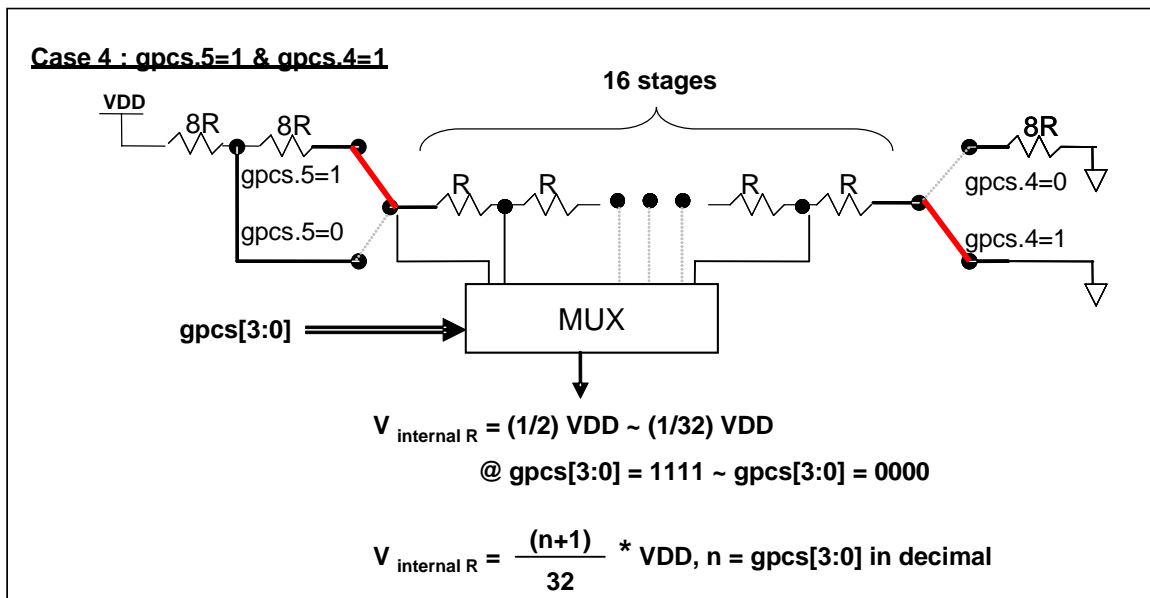


Fig.8: $V_{\text{internal } R}$ hardware connection if gpcs.5=1 and gpcs.4=1

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;           //  $V_{internal R} = V_{DD}*(18/32)$ 
gpcc = 0b1_0_0_0_000_0;         // enable comp, - input: PA3, + input:  $V_{internal R}$ 
padier = 0bxxxx_0_xxx;          // disable PA3 digital input to prevent leakage current

```

or

```

$ GPCS  $V_{DD}*18/32$ ;
$ GPCC Enable, N_PA3, P_R;      // - input: N_xx , + input: P_R( $V_{internal R}$ )
PADIER = 0bxxxx_0_xxx;

```

Case 2:

Choosing $V_{internal R}$ as minus input with $(22/40)*V_{DD}$ 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,  $V_{internal R} = V_{DD}*(22/40)$ 
gpcc = 0b1_0_0_1_011_1;         // Inverse output, - input:  $V_{internal R}$ , + input: PA4
padier = 0bxxx_0_xxxx;          // disable PA4 digital input to prevent leakage current

```

or

```

$ GPCS Output,  $V_{DD}*22/40$ ;
$ GPCC Enable, Inverse, N_R, P_PA4; // - input: N_R( $V_{internal R}$ ) , + 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 $V_{\text{internal R}}$ is selected as plus input, the supply voltage of $V_{\text{internal R}}$ is V_{DD} , the V_{DD} voltage level can be detected by adjusting the voltage level of $V_{\text{internal R}}$ to compare with bandgap. If N (gpcs[3:0] in decimal) is the number to let $V_{\text{internal R}}$ closest to bandgap 1.20 volt, the supply voltage V_{DD} can be calculated by using the following equations:

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

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

Case 1:

```

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

```

5.6. 16-bit Timer (Timer16)

PMS160 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 **intgs.4**. The hardware diagram of Timer16 is shown as Fig. 9.

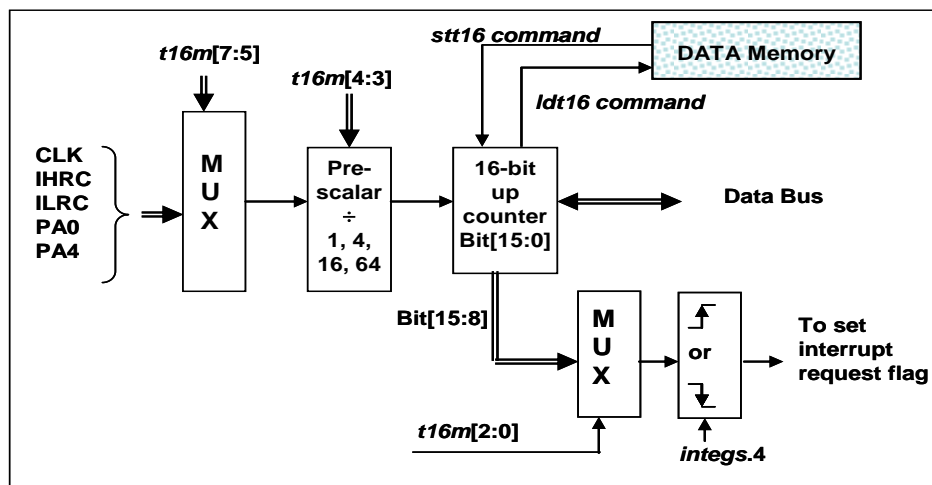


Fig. 9: Hardware diagram of Timer16

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, PMS160 will be reset to restart the program execution. The relative timing diagram of watchdog timer is shown as Fig.10.

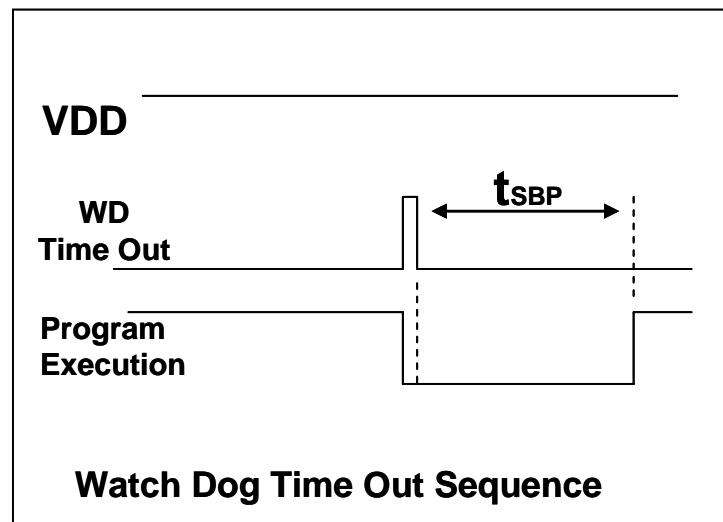


Fig. 10: Sequence of Watch Dog Time Out

5.8. Interrupt

There are 6 interrupt lines for PMS160:

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

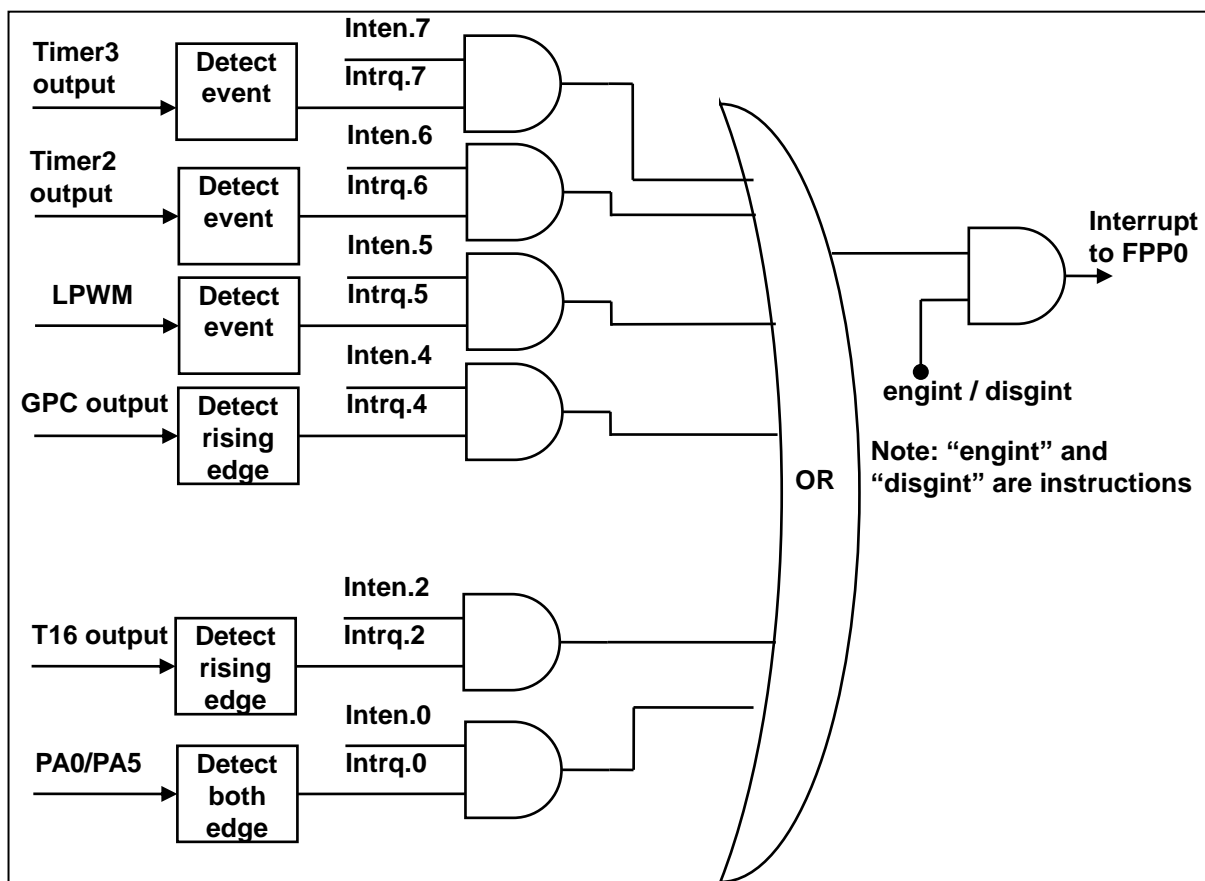


Fig. 11: Hardware diagram of Interrupt controller

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 SYCLK, the system will be awakened when the Timer counter reaches the set value.
 - 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 PMS160 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 PMS160 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 PMS160, registers *padier* and *pbdiar* 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

Please notice that when Fast boot-up is selected, no matter which wake-up mode is selected in *misc.5*, the wake-up mode will be forced to be FAST. If Normal boot-up is selected, the wake-up mode is determined by *misc.5*.

5.10. IO Pins

All the IO pins have the same structure. When PMS160 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 PADIAR 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.

<i>pa.0</i>	<i>pac.0</i>	<i>paph.0</i>	<i>papl.0</i>	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

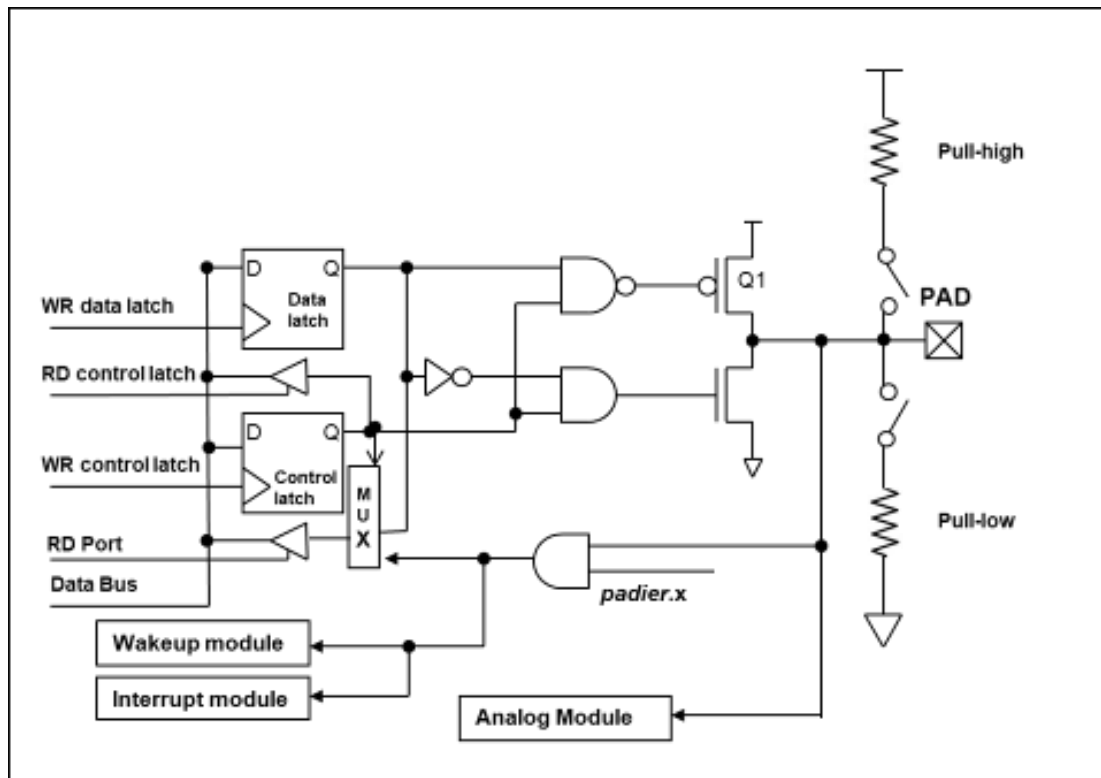


Fig. 12: Hardware diagram of IO buffer

5.11. Reset

There are many causes to reset the PMS160, once reset is asserted, all the registers in PMS160 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 PMS160, 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.

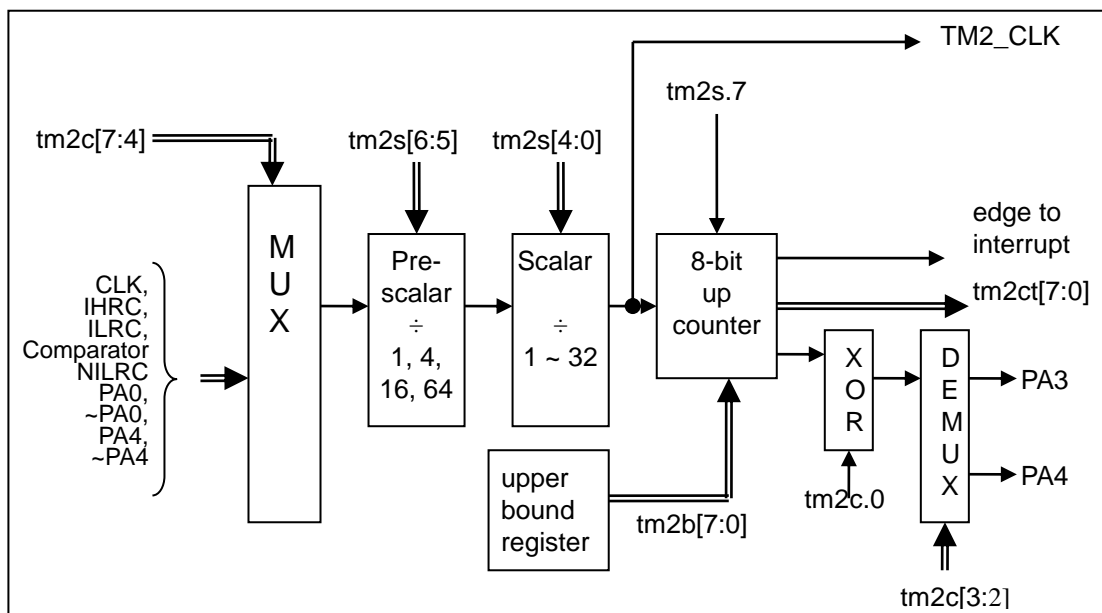
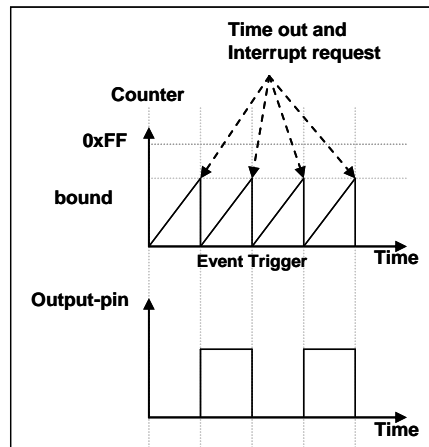


Fig. 13: Timer2 hardware diagram



Mode 0 – Period Mode

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:

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

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 ($S2 = 0 \sim 31$)

Example 1:

$tm2c = 0b0001_1000$, $Y=8\text{MHz}$

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

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

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

Example 2:

$tm2c = 0b0001_1000$, $Y=8\text{MHz}$

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

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

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

Example 3:

$tm2c = 0b0001_1000$, $Y=8\text{MHz}$

$tm2b = 0b0000_1111$, $K=15$

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

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

Example 4:

$tm2c = 0b0001_1000$, $Y=8\text{MHz}$

$tm2b = 0b0000_0001$, $K=1$

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

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

The sample program for using the Timer2 to generate periodical waveform 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 PMS160, the clock sources of Timer3 may come from system clock, internal high RC oscillator (IHRC), internal low RC oscillator (ILRC/NILRC), comparator and IFC, 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.

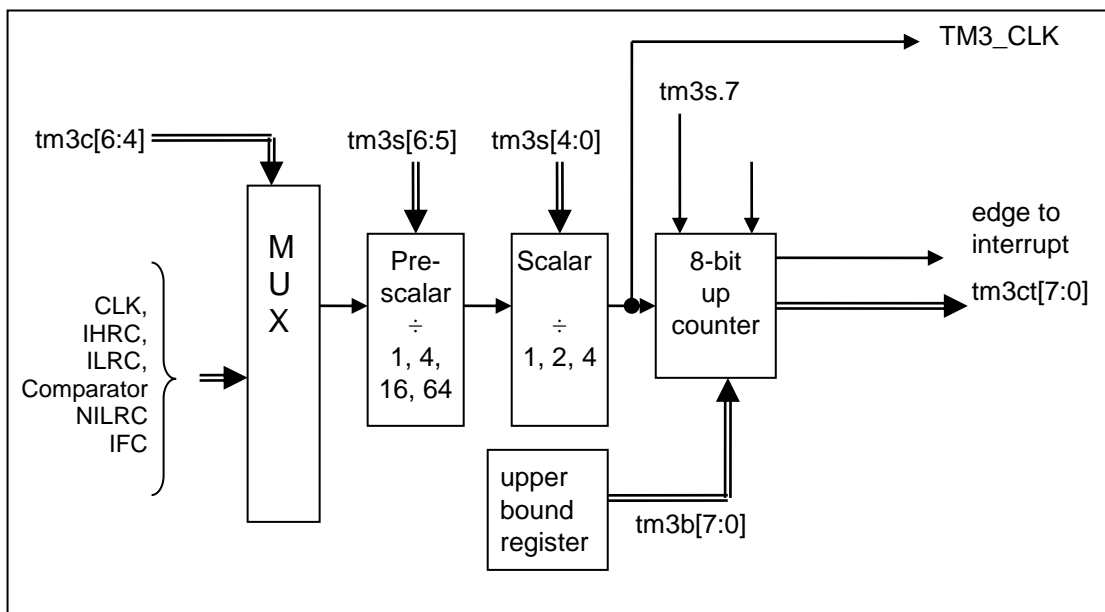
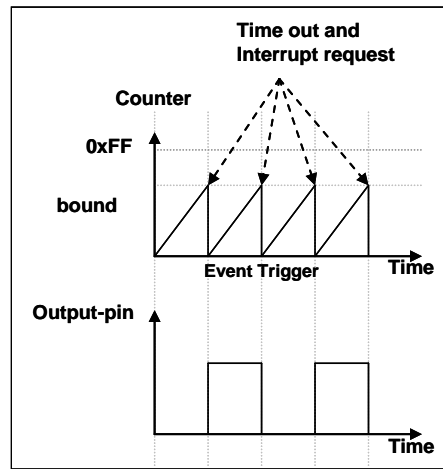


Fig. 15: Timer3 hardware diagram



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 PMS160. 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 ($T_{\text{Period}} = \text{Time of Period}$) and a time with output high level (Duty Cycle). The frequency of the LPWM output is the inverse of the period ($f_{\text{LPWM}} = 1/T_{\text{Period}}$),

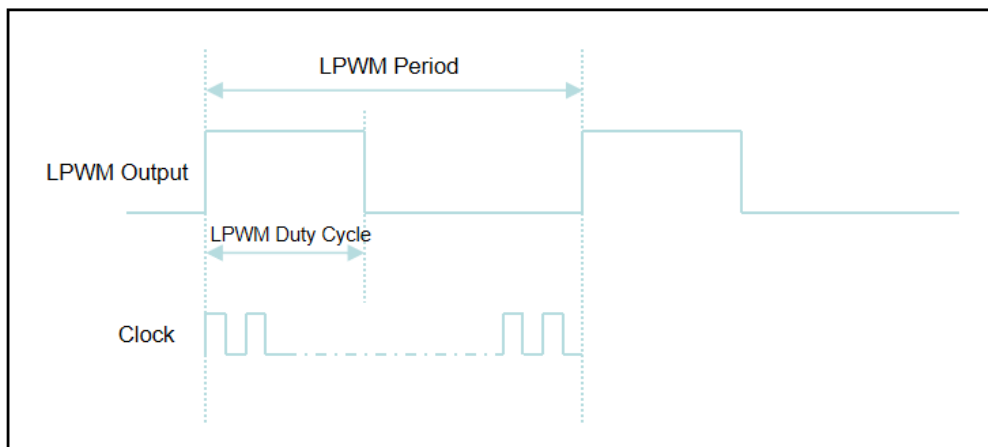


Fig. 17: PWM Output Waveform

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.

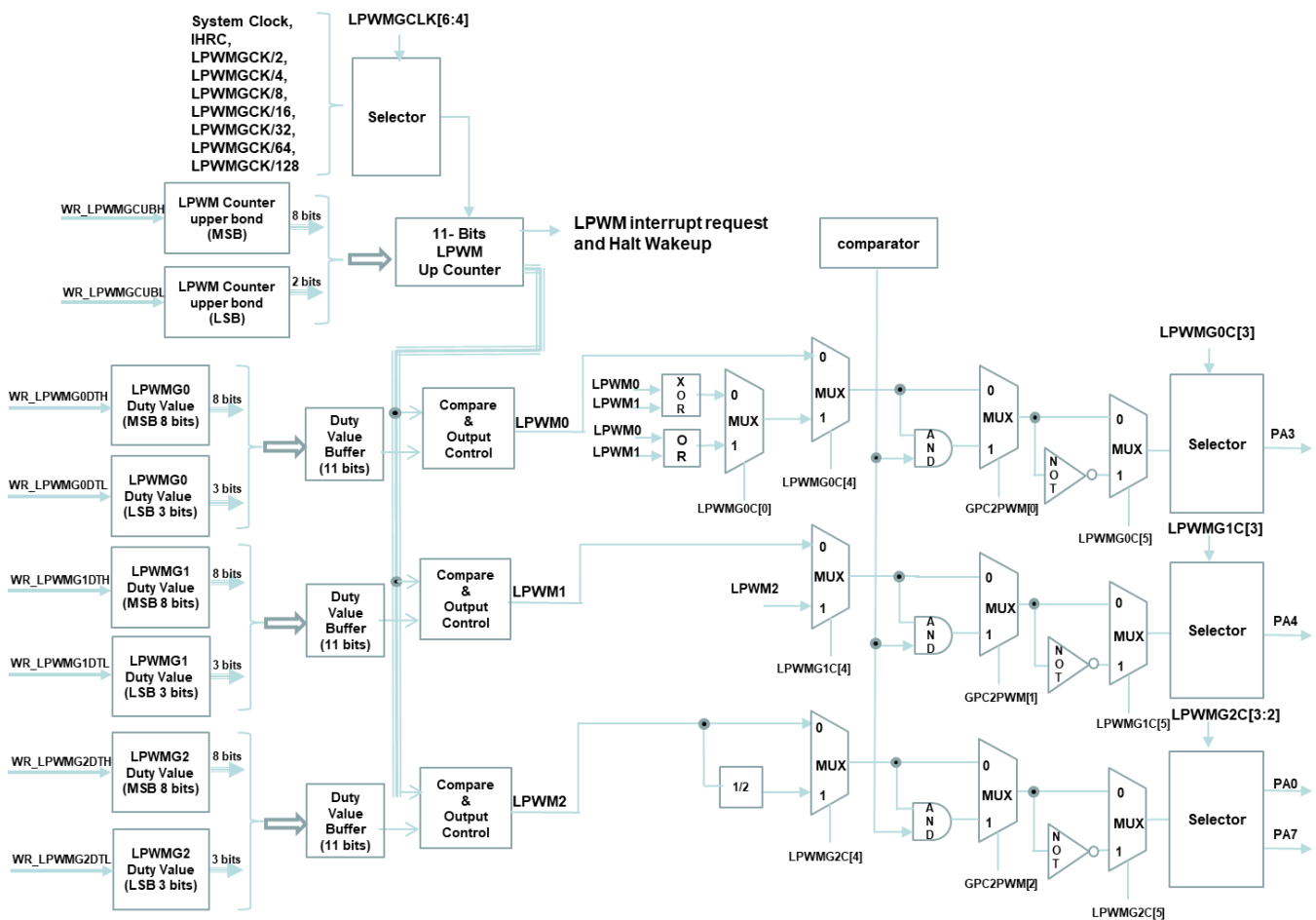


Fig. 18: Hardware Diagram of three 11-bit LPWM Generators

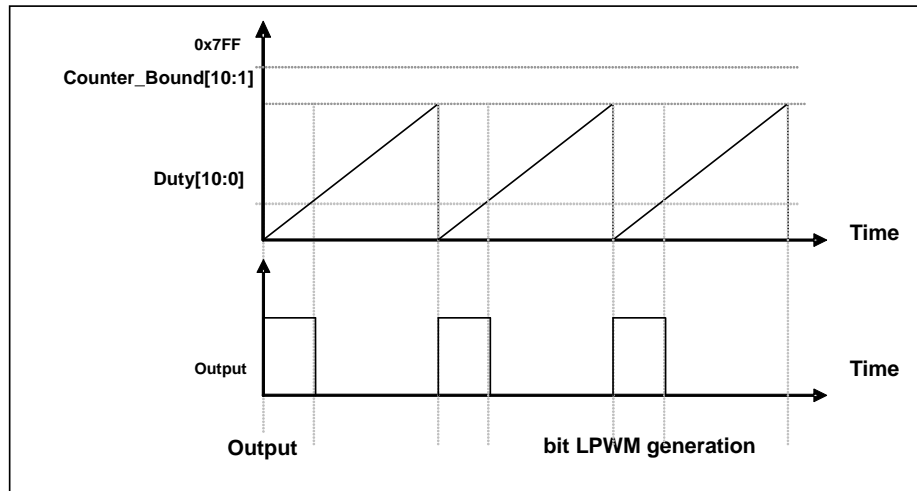


Fig. 19: Output Timing Diagram of 11-bit LPWM Generator

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.

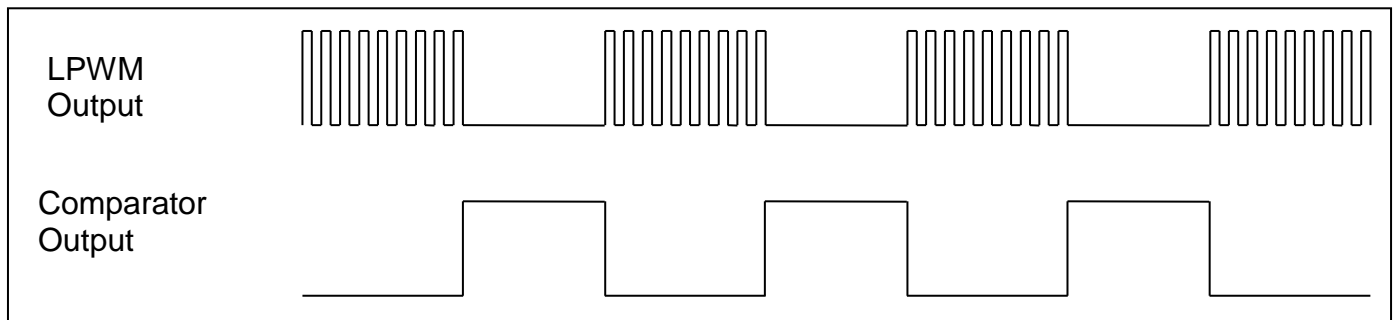


Fig.20 : Comparator controls the output of LPWM waveform

5.14.3. Equations for 11-bit LPWM Generator

$$\text{LPWM Frequency } F_{\text{LPWM}} = F_{\text{clock source}} \div [P \times (\text{CB10_1} + 1)]$$

$$\text{LPWM Duty(in time)} = (1 / F_{\text{LPWM}}) \times (\text{DB10_1} + \text{DB0} \times 0.5 + 0.5) \div (\text{CB10_1} + 1)$$

$$\text{LPWM Duty(in percentage)} = (\text{DB10_1} + \text{DB0} \times 0.5 + 0.5) \div (\text{CB10_1} + 1) \times 100\%$$

Where,

$$P = \text{LPWMGCLK}[6:4]; \text{ pre-scalar } P = 1, 2, 4, 8, 16, 32, 64, 128$$

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

$$\text{DB0} = \text{Duty_Bound}[0] = \text{LPWMGxDTL}[5] \quad (x=0/1/2)$$

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

5.14.4. Examples of LPWM Waveforms with Complementary Dead Zones

Based on the specific 11 bit SuLED LPWM architecture of PMS160, 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:

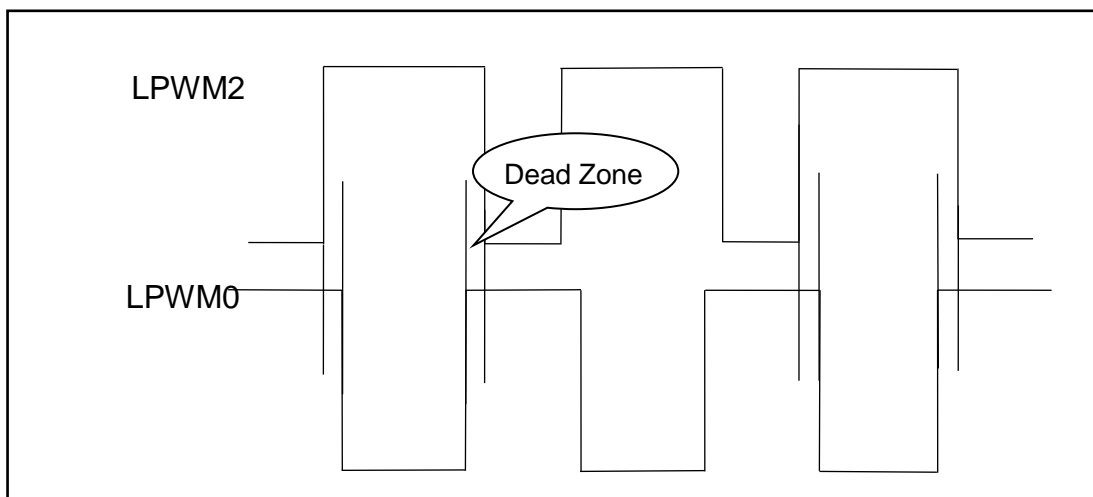


Fig. 21: Complementary LPWM waveform with dead zones

2. PWM waveform when switching two duty cycles:

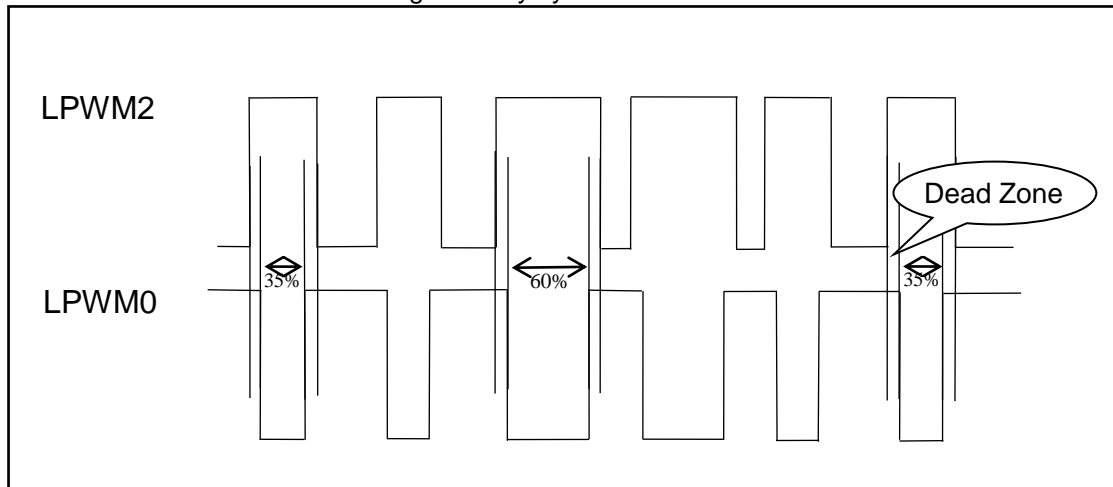


Fig. 22: Complementary LPWM waveform with dead zones

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

An IFC touch detecting circuit is included in PMS160. Its functional block diagram is shown as Fig.23.

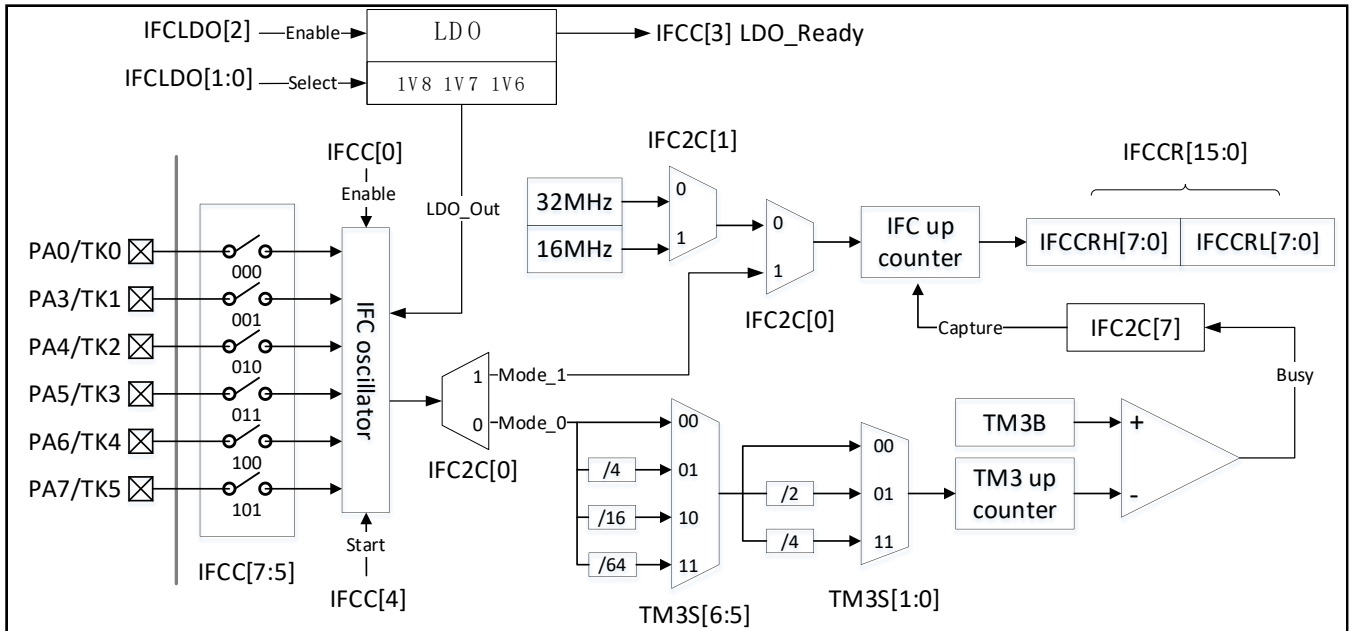


Fig. 23: Functional block diagram of the touch detecting circuit

The Touch detecting circuits in PMS160 apply the method of capacitive sensing, detecting the capacitive virtual ground effect of a finger, or the capacitance between sensors. This touch mode contains IFC oscillation circuit and hardware LDO voltage reference, and LDO can select three reference voltage (1.8V, 1.7V, 1.6V). There are two touch conversion modes in PMS160, Mode_0 and Mode_1, which are described in more detail as follows.

By reading the value of IFC up counter IFCCRH and IFCCRL, users can monitor capacitance changes on the sensor. The values read from Counter is related to CP(Capacitive Touch Sensor Pin), which represents the total capacitance which is the combination of PCB, wire and touch pad whose capacitance can be varied by finger's touch. Once the CP value is altered, the value from counter IFCCRH and IFCCRL will also change, and the change can be used to judge whether there is a touch action.

Mode 0 operates by counting IFC oscillating pulses using TM3. When the count value of TM3_Counter reaches the value set by TM3B, IFC timing will be stopped, then by reading IFC counter (IFCCRH and IFCCRL) value, the touch channel's touch count value can be obtained. Whether touch pad is pressed or released can be judged by the difference of IFC count.

When the IFCC.4 start signal is given, busy flag of IFC2C.7 will be set to "1", and then touch detection and conversion start. IFC2C.7 busy flag will be cleared when TM3B count limit is reached, which represents conversion completed. Then read the value of counter IFCCRH and IFCCRL.

Mode_1 is setting the output of the IFC oscillator as the IFC up counter's clock resource. The IFC up counter directly counts the output frequency by IFC oscillator. User determines the working time of IFC up counter through other timing methods of the program.

Notes:

- During the IFC Touch detection and conversion process, the maximum value of system clock (SYSCLK) shall not exceed 1MHz. Before executing the command, switch the system clock to 1 MHz or below, and it can switch to a higher frequency only after IFC Touch conversion is complete.

```

CLKMD_Save=CLKMD;
$ CLKMD IHRC/16,En_IHRC,En_ILRC;
.....
.....
CLKMD=CLKMD_Save;

```

Suggestion: When the IFC conversion program is included in the project, it is recommended to directly use 1MHz system clock. Because it can avoid the trouble of frequently switching system clock which may lead to inaccurate timing and longer interrupt running time.

- Before enable LDO, ensure that ILRC is enabled. The interval between LDO stop and restart must be longer than 2 ILRC clock periods or over 50uS.

```

$ IFCLDO;           // LDO Off
CLKMD.EN_ILRC = 1; // Make sure ILRC is enabled before enabling LDO
.delay 50;          // Delay 50us before enabling LDO (@sysclk=1MHz)
$ IFCC ;           // Clear IFCC, clear the flag of LDO_Ready
$ IFCLDO Enable, 1V8 // Enable LDO
while (!IFCC.LDO_Ready) NOP;
EXCAP = 0x08;      // 0x08 as default
CHDIS = 0x40;      // 0x40 as default

```

- When configuring IFC register, the TK channel must be enabled first, and the IFC oscillator can be enabled after delay 1uS. After the IFC oscillator is enabled, the IFC up counter can be enabled only after delaying 20uS to wait for the stability of the oscillation.

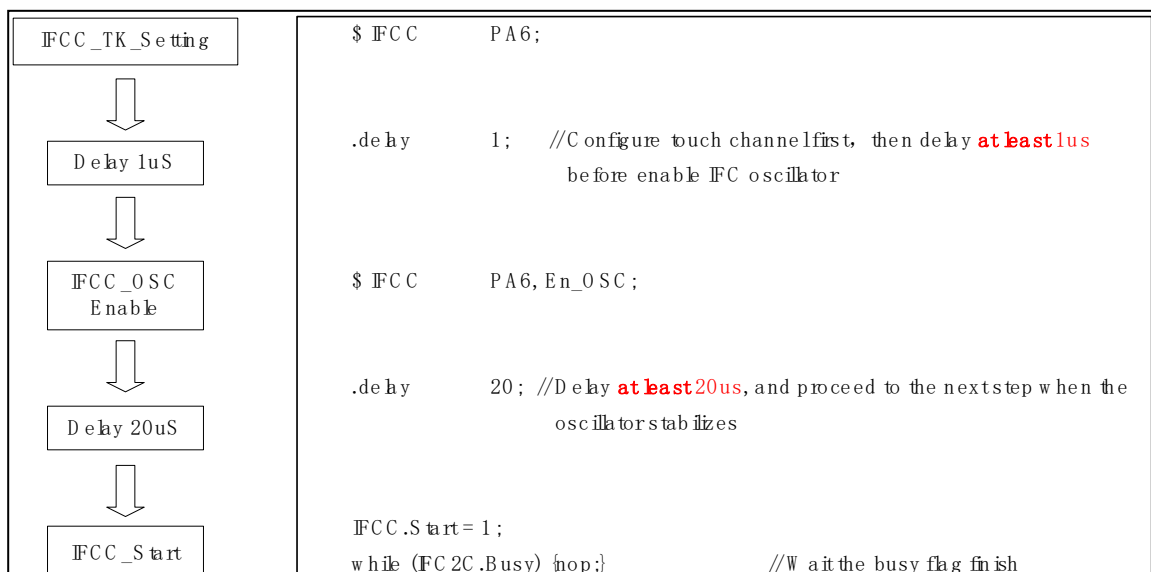


Fig. 24: Sequence diagram of IFCC register configuration

5.15.1. Sample of touch detection program

Support macro of PMS160 IFC Touch conversion since MCU_IDE_0.92C7:

Touch_TKProcess Parameter 0, Parameter1, Parameter 2, Parameter 3

Parameter 0: IFC Touch Channel Setting, 0 ~ 5. Set 0 = TK0, 1 = TK1.

Parameter 1: IFC Touch Count value. (16Bits)

Parameter 2: IFC Mode , 0 = Mode_0; 1 = Mode_1

Parameter 3: /2; /4; /8; /16....System frequency switch (IHRC) after IFC Touch conversion as users' require.

About more details of Touch_TKProcess Macro, please refer to PMS160.INC file in IDE

Sample program:

```
#include    "extern.h"

Debug_Printf    => 1
Debug_CMD      => 1
Div            => 8

word TK_Data[6],TK_Avg[6];

void    FPPA0 (void)
{
    .ADJUST_IC    SYSCLK=IHRC/Div, IHRC=16MHz, VDD=4V, INIT_RAM;

    .if Debug_CMD==1
    .DBG_CMD      LOG_CREATE    "log.bin";
    .DBG_CMD      SAVE_BIN      WAV, "P13:TK1:P13:TK2:P13:TK3:P13:TK4:P13:TK5"
    .endif

    $ TM2S /64, /25;
    TM2B = 200;
    $ TM2C IHRC;

    while (1)
    {
        //    wdreset;
```

```
if (INTRQ.TM2)
{
    INTRQ.TM2    =    0;
    word temp;
    .for N,<1,2,3,4,5>
        Touch_TKProcess TK#N#,TK_Data[N],0,/Div
        temp      =    TK_Avg[N] << 2;
        temp      =    temp - TK_Avg[N];
        temp      =    temp + TK_Data[N];
        TK_Avg[N]=    temp >> 2;
    .endm

    .if Debug_CMD==1
        .for N,<1,2,3,4,5>
            A=TK_Data[N]$0;
            .DBG_CMD    SAVE_ALU; //Store A in Log and send Touch Count WReviewdows
            A=TK_Data[N]$1;
            .DBG_CMD    SAVE_ALU; //Store A in Log and send Touch Count WReviewdows
        .endm
    .endif

    .if Debug_Printf==1
        .printf(rep,"TK1:%d TK2:%d TK3:%d TK4:%d TK5:%d\n", TK_Data[1], TK_Data[2], TK_Data[3], TK_Data[4],
        TK_Data[5]);
    .endif
}
}
}
```

6. IO Registers

6.1. ACC Status Flag Register (*flag*), IO address = 0x00

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

6.2. Stack Pointer Register (*sp*), IO address = 0x02

Bit	Reset	R/W	Description
7 - 0	-	R/W	Stack Pointer Register. Read out the current stack pointer or write to change the stack pointer. Please notice that bit 0 should be kept 0 due to program counter is 16 bits.

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

Bit	Reset	R/W	Description													
7 - 5	111	R/W	System clock selection:													
			<table style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 50%; text-align: center;">Type 0, clkmd[3]=0</td> <td style="width: 50%; text-align: center;">Type 1, clkmd[3]=1</td> </tr> <tr> <td>000: IHRC/4</td> <td>000: IHRC/16</td> </tr> <tr> <td>001: reserved</td> <td>001: IHRC/8</td> </tr> <tr> <td>01x: reserved</td> <td>010: ILRC/16</td> </tr> <tr> <td>100: reserved</td> <td>011: IHRC/32</td> </tr> <tr> <td>101: reserved</td> <td>100: IHRC/64</td> </tr> <tr> <td>110: ILRC/4</td> <td>110: reserved</td> </tr> <tr> <td>111: ILRC (default)</td> <td>1x1: reserved.</td> </tr> </table>	Type 0, clkmd[3]=0	Type 1, clkmd[3]=1	000: IHRC/4	000: IHRC/16	001: reserved	001: IHRC/8	01x: reserved	010: ILRC/16	100: reserved	011: IHRC/32	101: reserved	100: IHRC/64	110: ILRC/4
Type 0, clkmd[3]=0	Type 1, clkmd[3]=1															
000: IHRC/4	000: IHRC/16															
001: reserved	001: IHRC/8															
01x: reserved	010: ILRC/16															
100: reserved	011: IHRC/32															
101: reserved	100: IHRC/64															
110: ILRC/4	110: reserved															
111: ILRC (default)	1x1: reserved.															
4	0	R/W	IHRC oscillator Enable. 0 / 1: disable / enable													
3	0	R/W	Clock Type Select. This bit is used to select the clock type in bit [7:5]. 0 / 1: Type 0 / Type 1													
2	1	R/W	ILRC Enable. 0 / 1: disable / enable If ILRC is disabled, watchdog timer is also disabled.													
1	1	R/W	Watch Dog Enable. 0 / 1: disable / enable													
0	0	R/W	Pin PA5/PRSTB function. 0 / 1: PA5 / PRSTB													

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	-	-	Reserved.
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	-	-	Reserved.
2	-	R/W	Interrupt Request from Timer16, this bit is set by hardware and cleared by software. 0 / 1: No request / Request
1	-	-	Reserved.
0	-	R/W	Interrupt Request from pin PA0/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: IFC
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	RO	LPWMG disable/enable. 0: LPWMG disable 1: LPWMG enable
6 - 4	000	RO	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	RO	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	-	RO	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	-	RO	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	-	RO	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	-	RO	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 2 (*IFC2C*), IO address = 0x20

Bit	Reset	R/W	Description
7	0	R/W	IFC busy flag
6 - 2	-	-	Reserved
1	0	R/W	IFC counter clock selection 0: 32MHz 1: 16MHz
0	0	R/W	0: mode_0. Please keep 0. (Default mode_0) 1: mode_1

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

Bit	Reset	R/W	Description	
7 - 5	-	R/W	Data	
			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	IFC function Disable
4	0	WO	Command to start IFC up counter, automatically clear at the end of conversion.	
3	0	R/W	The flag of LDO_Ready and should be cleared before enable LDO. This bit is set by hardware and cleared by software.	
2	0	RO	Flag of IFC up counter overflow	
1	-	-	Reserve and should be set to 0.	
0	0	R/W	IFC oscillator enable	

Note: Here to emphasize the points when IFCC register configuration. Firstly, enable TK Channel; Secondly, IFC oscillator should be enabled after delay 1uS; Thirdly, after enable IFC oscillator, it needs to delay 20us to wait for oscillatory stability; Finally, enable IFC touch count.

6.35. Touch Key Counter High Register (*IFCCRH*), IO address = 0x22

Bit	Reset	R/W	Description
7 - 0	-	RO	IFCCR [15:8] of touch key charge counter.

6.36. Touch Key Counter Low Register (*IFCCRL*), IO address = 0x23

Bit	Reset	R/W	Description
7 - 0	-	RO	IFCCR [7:0] of touch key charge counter.

6.37. LDO Control Register (*IFCLDO*), IO address = 0x24

Bit	Reset	R/W	Description
7 - 3	-	-	Reserved
2	0	R/W	Enable LDO module. Before enable LDO, clear flag of LDO_Ready in IFCC.3. 0: Disable; 1: Enable.
1 - 0	00	R/W	LDO reference voltage selection: 00 : 1V8, 1.8V 01 : Reserved 10 : 1V7, 1.7V 11 : 1V6, 1.6V

Note: Before enable LDO, ensure that ILRC is enabled. The interval between LDO stop and restart must be longer than 2 ILRC clock periods or over 50uS.

6.38. Touch capacitor charge/discharge setting register(*CHDIS*) , IO address = 0x25

Bit	Reset	R/W	Description
7 - 4	-	R/W	0100: default values by program
3 - 0	-	R/W	Reserved, program writes 0.

Note: Every time after LDO Enable and LDO Ready, CHDIS = 0x40 must be reset.

6.39. Touch capacitor control register(*EXCAP*) , IO address = 0x26

Bit	Reset	R/W	Description
7	-	R/W	Reserved, program writes 0.
6 - 4	-	R/W	Reserved, program writes 0
3 - 0	00	R/W	Reference coefficient of IFC Touch converter capacitance Set Range:0x01 ~ 0x0F 0000 : Unavailable 0001 : Large Touch value (Mode_0) 1000 : Write-in values by program default 1111 : Small Touch value (Mode_0) Other : Can be set as users require

Note: Every time after LDO_Enable and LDO_Ready, the value of excap register must be reset, default value is 0x08.

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)
\bar{T}	NEG (2's complement)
OV	Overflow (The operational result is out of range in signed 2's complement number system)
Z	Zero (If the result of ALU operation is zero, this bit is set to 1)
C	Carry (The operational result is to have carry out for addition or to borrow carry for subtraction in unsigned number system)
AC	Auxiliary Carry (If there is a carry out from low nibble after the result of ALU operation, this bit is set to 1)
M.n	Only addressed in 0~0x3F (0~63) is allowed

7.1. Data Transfer Instructions

<i>mov</i> a, l	<p>Move immediate data into ACC. Example: <i>mov</i> a, 0x0f; Result: a ← 0fh; Affected flags: 『N』 Z 『N』 C 『N』 AC 『N』 OV</p>
<i>mov</i> M, a	<p>Move data from ACC into memory Example: <i>mov</i> MEM, a; Result: MEM ← a Affected flags: 『N』 Z 『N』 C 『N』 AC 『N』 OV</p>
<i>mov</i> a, M	<p>Move data from memory into ACC Example: <i>mov</i> a, MEM ; Result: a ← MEM; Flag Z is set when MEM is zero. Affected flags: 『Y』 Z 『N』 C 『N』 AC 『N』 OV</p>
<i>mov</i> a, IO	<p>Move data from IO into ACC Example: <i>mov</i> a, pa ; Result: a ← pa; Flag Z is set when pa is zero. Affected flags: 『Y』 Z 『N』 C 『N』 AC 『N』 OV</p>
<i>mov</i> IO, a	<p>Move data from ACC into IO Example: <i>mov</i> pa, a; Result: pa ← a Affected flags: 『N』 Z 『N』 C 『N』 AC 『N』 OV</p>
<i>ldt16</i> word	<p>Move 16-bit counting values in Timer16 to memory in word. Example: <i>ldt16</i> word; Result: word ← 16-bit timer Affected flags: 『N』 Z 『N』 C 『N』 AC 『N』 OV</p> <p>Application Example:</p> <pre> ----- 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 ----- </pre>

<i>stt16</i> word	<p>Store 16-bit data from memory in word to Timer16.</p> <p>Example: <i>stt16</i> word;</p> <p>Result: 16-bit timer ←word</p> <p>Affected flags: 『N』 Z 『N』 C 『N』 AC 『N』 OV</p> <p>Application Example:</p> <hr style="border-top: 1px dashed black;"/> <pre style="margin: 0;">word T16val ; // declare a RAM word ... mov a, 0x34 ; mov lb@ T16val , a ; // move 0x34 to T16val (LSB) mov a, 0x12 ; mov hb@ T16val , a ; // move 0x12 to T16val (MSB) stt16 T16val ; // initial T16 with 0x1234 ...</pre> <hr style="border-top: 1px dashed black;"/>
<i>idxm</i> a, index	<p>Move data from specified memory to ACC by indirect method. It needs 2T to execute this instruction.</p> <p>Example: <i>idxm</i> a, index;</p> <p>Result: a ← [index], where index is declared by word.</p> <p>Affected flags: 『N』 Z 『N』 C 『N』 AC 『N』 OV</p> <p>Application Example:</p> <hr style="border-top: 1px dashed black;"/> <pre style="margin: 0;">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</pre> <hr style="border-top: 1px dashed black;"/>
<i>ldxm</i> index, a	<p>Move data from ACC to specified memory by indirect method. It needs 2T to execute this instruction.</p> <p>Example: <i>ldxm</i> index, a;</p> <p>Result: [index] ← a; where index is declared by word.</p> <p>Affected flags: 『N』 Z 『N』 C 『N』 AC 『N』 OV</p> <p>Application Example:</p> <hr style="border-top: 1px dashed black;"/> <pre style="margin: 0;">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 ; ldxm RAMIndex, a ; // move 0xA5 to memory in address 0x5B</pre> <hr style="border-top: 1px dashed black;"/>

<i>xch M</i>	<p>Exchange data between ACC and memory</p> <p>Example: <code>xch MEM ;</code></p> <p>Result: <code>MEM ← a , a ← MEM</code></p> <p>Affected flags: 『N』 Z 『N』 C 『N』 AC 『N』 OV</p>
<i>pushaf</i>	<p>Move the ACC and flag register to memory that address specified in the stack pointer.</p> <p>Example: <code>pushaf ;</code></p> <p>Result: <code>[sp] ← {flag, ACC};</code> <code>sp ← sp + 2 ;</code></p> <p>Affected flags: 『N』 Z 『N』 C 『N』 AC 『N』 OV</p> <p>Application Example:</p> <hr style="border-top: 1px dashed black;"/> <pre>.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 ;</pre> <hr style="border-top: 1px dashed black;"/>
<i>popaf</i>	<p>Restore ACC and flag from the memory which address is specified in the stack pointer.</p> <p>Example: <code>popaf ;</code></p> <p>Result: <code>sp ← sp - 2 ;</code> <code>{Flag, ACC} ← [sp] ;</code></p> <p>Affected flags: 『Y』 Z 『Y』 C 『Y』 AC 『Y』 OV</p>

7.2. Arithmetic Operation Instructions

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

<i>addc</i> a	Add carry with ACC, then put result into ACC Example: <i>addc</i> a ; Result: $a \leftarrow a + C$ Affected flags: 『Y』 Z 『Y』 C 『Y』 AC 『Y』 OV
<i>addc</i> M	Add carry with memory, then put result into memory Example: <i>addc</i> MEM ; Result: $MEM \leftarrow MEM + C$ Affected flags: 『Y』 Z 『Y』 C 『Y』 AC 『Y』 OV
<i>sub</i> a, I	Subtraction immediate data from ACC, then put result into ACC. Example: <i>sub</i> a, 0x0f; Result: $a \leftarrow a - 0fh$ ($a + [2's \text{ complement of } 0fh]$) Affected flags: 『Y』 Z 『Y』 C 『Y』 AC 『Y』 OV
<i>sub</i> a, M	Subtraction data in memory from ACC, then put result into ACC Example: <i>sub</i> a, MEM ; Result: $a \leftarrow a - MEM$ ($a + [2's \text{ complement of } M]$) Affected flags: 『Y』 Z 『Y』 C 『Y』 AC 『Y』 OV
<i>sub</i> M, a	Subtraction data in ACC from memory, then put result into memory Example: <i>sub</i> MEM, a ; Result: $MEM \leftarrow MEM - a$ ($MEM + [2's \text{ complement of } a]$) Affected flags: 『Y』 Z 『Y』 C 『Y』 AC 『Y』 OV
<i>subc</i> a, M	Subtraction data in memory and carry from ACC, then put result into ACC Example: <i>subc</i> a, MEM ; Result: $a \leftarrow a - MEM - C$ Affected flags: 『Y』 Z 『Y』 C 『Y』 AC 『Y』 OV
<i>subc</i> M, a	Subtraction ACC and carry bit from memory, then put result into memory Example: <i>subc</i> MEM, a ; Result: $MEM \leftarrow MEM - a - C$ Affected flags: 『Y』 Z 『Y』 C 『Y』 AC 『Y』 OV
<i>subc</i> a	Subtraction carry from ACC, then put result into ACC Example: <i>subc</i> a ; Result: $a \leftarrow a - C$ Affected flags: 『Y』 Z 『Y』 C 『Y』 AC 『Y』 OV
<i>subc</i> M	Subtraction carry from the content of memory, then put result into memory Example: <i>subc</i> MEM ; Result: $MEM \leftarrow MEM - C$ Affected flags: 『Y』 Z 『Y』 C 『Y』 AC 『Y』 OV
<i>inc</i> M	Increment the content of memory Example: <i>inc</i> MEM ; Result: $MEM \leftarrow MEM + 1$ Affected flags: 『Y』 Z 『Y』 C 『Y』 AC 『Y』 OV
<i>dec</i> M	Decrement the content of memory Example: <i>dec</i> MEM ; Result: $MEM \leftarrow MEM - 1$ Affected flags: 『Y』 Z 『Y』 C 『Y』 AC 『Y』 OV
<i>clear</i> M	Clear the content of memory Example: <i>clear</i> MEM ; Result: $MEM \leftarrow 0$ Affected flags: 『N』 Z 『N』 C 『N』 AC 『N』 OV

7.3. Shift Operation Instructions

<i>sr a</i>	Shift right of ACC, shift 0 to bit 7 Example: <i>sr a</i> ; Result: $a(0, b7, b6, b5, b4, b3, b2, b1) \leftarrow a(b7, b6, b5, b4, b3, b2, b1, b0)$, $C \leftarrow a(b0)$ Affected flags: 『N』 Z 『Y』 C 『N』 AC 『N』 OV
<i>src a</i>	Shift right of ACC with carry bit 7 to flag Example: <i>src a</i> ; Result: $a(c, b7, b6, b5, b4, b3, b2, b1) \leftarrow a(b7, b6, b5, b4, b3, b2, b1, b0)$, $C \leftarrow a(b0)$ Affected flags: 『N』 Z 『Y』 C 『N』 AC 『N』 OV
<i>sr M</i>	Shift right the content of memory, shift 0 to bit 7 Example: <i>sr MEM</i> ; Result: $MEM(0, b7, b6, b5, b4, b3, b2, b1) \leftarrow MEM(b7, b6, b5, b4, b3, b2, b1, b0)$, $C \leftarrow MEM(b0)$ Affected flags: 『N』 Z 『Y』 C 『N』 AC 『N』 OV
<i>src M</i>	Shift right of memory with carry bit 7 to flag Example: <i>src MEM</i> ; Result: $MEM(c, b7, b6, b5, b4, b3, b2, b1) \leftarrow MEM(b7, b6, b5, b4, b3, b2, b1, b0)$, $C \leftarrow MEM(b0)$ Affected flags: 『N』 Z 『Y』 C 『N』 AC 『N』 OV
<i>sl a</i>	Shift left of ACC shift 0 to bit 0 Example: <i>sl a</i> ; Result: $a(b6, b5, b4, b3, b2, b1, b0, 0) \leftarrow a(b7, b6, b5, b4, b3, b2, b1, b0)$, $C \leftarrow a(b7)$ Affected flags: 『N』 Z 『Y』 C 『N』 AC 『N』 OV
<i>slc a</i>	Shift left of ACC with carry bit 0 to flag Example: <i>slc a</i> ; Result: $a(b6, b5, b4, b3, b2, b1, b0, c) \leftarrow a(b7, b6, b5, b4, b3, b2, b1, b0)$, $C \leftarrow a(b7)$ Affected flags: 『N』 Z 『Y』 C 『N』 AC 『N』 OV
<i>sl M</i>	Shift left of memory, shift 0 to bit 0 Example: <i>sl MEM</i> ; Result: $MEM(b6, b5, b4, b3, b2, b1, b0, 0) \leftarrow MEM(b7, b6, b5, b4, b3, b2, b1, b0)$, $C \leftarrow MEM(b7)$ Affected flags: 『N』 Z 『Y』 C 『N』 AC 『N』 OV
<i>slc M</i>	Shift left of memory with carry bit 0 to flag Example: <i>slc MEM</i> ; Result: $MEM(b6, b5, b4, b3, b2, b1, b0, C) \leftarrow MEM(b7, b6, b5, b4, b3, b2, b1, b0)$, $C \leftarrow MEM(b7)$ Affected flags: 『N』 Z 『Y』 C 『N』 AC 『N』 OV
<i>swap a</i>	Swap the high nibble and low nibble of ACC Example: <i>swap a</i> ; Result: $a(b3, b2, b1, b0, b7, b6, b5, b4) \leftarrow a(b7, b6, b5, b4, b3, b2, b1, b0)$ Affected flags: 『N』 Z 『N』 C 『N』 AC 『N』 OV

7.4. Logic Operation Instructions

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

<i>not</i> a	<p>Perform 1's complement (logical complement) of ACC</p> <p>Example: <i>not</i> a ;</p> <p>Result: a ← $\sim a$</p> <p>Affected flags: 『Y』 Z 『N』 C 『N』 AC 『N』 OV</p> <p>Application Example:</p> <hr style="border-top: 1px dashed black;"/> <pre style="margin-left: 20px;"> mov a, 0x38 ; // ACC=0X38 not a ; // ACC=0XC7 </pre> <hr style="border-top: 1px dashed black;"/>
<i>not</i> M	<p>Perform 1's complement (logical complement) of memory</p> <p>Example: <i>not</i> MEM ;</p> <p>Result: MEM ← $\sim \text{MEM}$</p> <p>Affected flags: 『Y』 Z 『N』 C 『N』 AC 『N』 OV</p> <p>Application Example:</p> <hr style="border-top: 1px dashed black;"/> <pre style="margin-left: 20px;"> mov a, 0x38 ; mov mem, a ; // mem = 0x38 not mem ; // mem = 0xC7 </pre> <hr style="border-top: 1px dashed black;"/>
<i>neg</i> a	<p>Perform 2's complement of ACC</p> <p>Example: <i>neg</i> a ;</p> <p>Result: a ← \overline{a}</p> <p>Affected flags: 『Y』 Z 『N』 C 『N』 AC 『N』 OV</p> <p>Application Example:</p> <hr style="border-top: 1px dashed black;"/> <pre style="margin-left: 20px;"> mov a, 0x38 ; // ACC=0X38 neg a ; // ACC=0XC8 </pre> <hr style="border-top: 1px dashed black;"/>
<i>neg</i> M	<p>Perform 2's complement of memory</p> <p>Example: <i>neg</i> MEM ;</p> <p>Result: MEM ← $\overline{\text{MEM}}$</p> <p>Affected flags: 『Y』 Z 『N』 C 『N』 AC 『N』 OV</p> <p>Application Example:</p> <hr style="border-top: 1px dashed black;"/> <pre style="margin-left: 20px;"> mov a, 0x38 ; mov mem, a ; // mem = 0x38 not mem ; // mem = 0xC8 </pre> <hr style="border-top: 1px dashed black;"/>

7.5. Bit Operation Instructions

<i>set0</i> IO.n	<p>Set bit n of IO port to low Example: <i>set0</i> pa.5 ; Result: set bit 5 of port A to low Affected flags: 『N』 Z 『N』 C 『N』 AC 『N』 OV</p>
<i>set1</i> IO.n	<p>Set bit n of IO port to high Example: <i>set1</i> pa.5 ; Result: set bit 5 of port A to high Affected flags: 『N』 Z 『N』 C 『N』 AC 『N』 OV</p>
<i>set0</i> M.n	<p>Set bit n of memory to low Example: <i>set0</i> MEM.5 ; Result: set bit 5 of MEM to low Affected flags: 『N』 Z 『N』 C 『N』 AC 『N』 OV</p>
<i>set1</i> M.n	<p>Set bit n of memory to high Example: <i>set1</i> MEM.5 ; Result: set bit 5 of MEM to high Affected flags: 『N』 Z 『N』 C 『N』 AC 『N』 OV</p>
<i>swapc</i> IO.n	<p>Swap the nth bit of IO port with carry bit Example: <i>swapc</i> IO.0; Result: $C \leftarrow IO.0, IO.0 \leftarrow C$ 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) : <pre> ... set1 pac.0 ; // set PA.0 as output ... set0 flag.1 ; // C=0 swapc pa.0 ; // move C to PA.0 (bit operation), PA.0=0 set1 flag.1 ; // C=1 swapc pa.0 ; // move C to PA.0 (bit operation), PA.0=1 ... </pre> Application Example2 (serial input) : <pre> ... 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 ... </pre> </p>

7.6. Conditional Operation Instructions

<i>ceqsn a, l</i>	<p>Compare ACC with immediate data and skip next instruction if both are equal. Flag will be changed like as ($a \leftarrow a - l$) Example: <i>ceqsn a, 0x55</i> ; <i>inc MEM</i> ; <i>goto error</i> ;</p> <p>Result: If $a=0x55$, then “goto error”; otherwise, “inc MEM”. Affected flags: 『Y』 Z 『Y』 C 『Y』 AC 『Y』 OV</p>
<i>ceqsn a, M</i>	<p>Compare ACC with memory and skip next instruction if both are equal. Flag will be changed like as ($a \leftarrow a - M$) Example: <i>ceqsn a, MEM</i> ;</p> <p>Result: If $a=MEM$, skip next instruction Affected flags: 『Y』 Z 『Y』 C 『Y』 AC 『Y』 OV</p>
<i>cneqsn a, M</i>	<p>Compare ACC with memory and skip next instruction if both are not equal. Flag will be changed like as ($a \leftarrow a - M$) Example: <i>cneqsn a, MEM</i> ;</p> <p>Result: If $a \neq MEM$, skip next instruction Affected flags: 『Y』 Z 『Y』 C 『Y』 AC 『Y』 OV</p>
<i>cneqsn a, l</i>	<p>Compare ACC with immediate data and skip next instruction if both are no equal. Flag will be changed like as ($a \leftarrow a - l$) Example: <i>cneqsn a, 0x55</i> ; <i>inc MEM</i> ; <i>goto error</i> ;</p> <p>Result: If $a \neq 0x55$, then “goto error”; Otherwise, “inc MEM”. Affected flags: 『Y』 Z 『Y』 C 『Y』 AC 『Y』 OV</p>
<i>t0sn IO.n</i>	<p>Check IO bit and skip next instruction if it's low Example: <i>t0sn pa.5</i> ;</p> <p>Result: If bit 5 of port A is low, skip next instruction Affected flags: 『N』 Z 『N』 C 『N』 AC 『N』 OV</p>
<i>t1sn IO.n</i>	<p>Check IO bit and skip next instruction if it's high Example: <i>t1sn pa.5</i> ;</p> <p>Result: If bit 5 of port A is high, skip next instruction Affected flags: 『N』 Z 『N』 C 『N』 AC 『N』 OV</p>
<i>t0sn M.n</i>	<p>Check memory bit and skip next instruction if it's low Example: <i>t0sn MEM.5</i> ;</p> <p>Result: If bit 5 of MEM is low, then skip next instruction Affected flags: 『N』 Z 『N』 C 『N』 AC 『N』 OV</p>
<i>t1sn M.n</i>	<p>Check memory bit and skip next instruction if it's high Example: <i>t1sn MEM.5</i> ;</p> <p>Result: If bit 5 of MEM is high, then skip next instruction Affected flags: 『N』 Z 『N』 C 『N』 AC 『N』 OV</p>
<i>izsn a</i>	<p>Increment ACC and skip next instruction if ACC is zero Example: <i>izsn a</i> ;</p> <p>Result: $a \leftarrow a + 1$, skip next instruction if $a = 0$ Affected flags: 『Y』 Z 『Y』 C 『Y』 AC 『Y』 OV</p>

<i>dzn</i> a	Decrement ACC and skip next instruction if ACC is zero Example: <i>dzn</i> a; Result: $A \leftarrow A - 1$, skip next instruction if a = 0 Affected flags: 『Y』 Z 『Y』 C 『Y』 AC 『Y』 OV
<i>izsn</i> M	Increment memory and skip next instruction if memory is zero Example: <i>izsn</i> MEM; Result: $MEM \leftarrow MEM + 1$, skip next instruction if MEM= 0 Affected flags: 『Y』 Z 『Y』 C 『Y』 AC 『Y』 OV
<i>dzn</i> M	Decrement memory and skip next instruction if memory is zero Example: <i>dzn</i> MEM; Result: $MEM \leftarrow MEM - 1$, skip next instruction if MEM = 0 Affected flags: 『Y』 Z 『Y』 C 『Y』 AC 『Y』 OV

7.7. System control Instructions

<i>call</i> label	Function call, address can be full range address space Example: <i>call</i> function1; Result: [sp] \leftarrow pc + 1 pc \leftarrow function1 sp \leftarrow sp + 2 Affected flags: 『N』 Z 『N』 C 『N』 AC 『N』 OV
<i>goto</i> label	Go to specific address which can be full range address space Example: <i>goto</i> error; Result: Go to error and execute program. Affected flags: 『N』 Z 『N』 C 『N』 AC 『N』 OV
<i>ret</i> l	Place immediate data to ACC, then return Example: <i>ret</i> 0x55; Result: $A \leftarrow 55h$ ret ; Affected flags: 『N』 Z 『N』 C 『N』 AC 『N』 OV
<i>ret</i>	Return to program which had function call Example: <i>ret</i> ; Result: sp \leftarrow sp - 2 pc \leftarrow [sp] Affected flags: 『N』 Z 『N』 C 『N』 AC 『N』 OV
<i>reti</i>	Return to program from interrupt service routine. After this command is executed, global interrupt is enabled automatically. Example: <i>reti</i> ; Affected flags: 『N』 Z 『N』 C 『N』 AC 『N』 OV
<i>nop</i>	No operation Example: <i>nop</i> ; Result: nothing changed Affected flags: 『N』 Z 『N』 C 『N』 AC 『N』 OV
<i>pcadd</i> a	Next program counter is current program counter plus ACC. Example: <i>pcadd</i> a; Result: pc \leftarrow pc + a Affected flags: 『N』 Z 『N』 C 『N』 AC 『N』 OV

	<p>Application Example:</p> <p>-----</p> <pre> ... mov a, 0x02 ; pcadd a ; // PC <- PC+2 goto err1 ; goto correct ; // jump here goto err2 ; goto err3 ; ... correct: // jump here ... </pre> <p>-----</p>
<i>engint</i>	<p>Enable global interrupt enable</p> <p>Example: <i>engint</i>;</p> <p>Result: Interrupt request can be sent to FPP0</p> <p>Affected flags: 『N』 Z 『N』 C 『N』 AC 『N』 OV</p>
<i>disgint</i>	<p>Disable global interrupt enable</p> <p>Example: <i>disgint</i>;</p> <p>Result: Interrupt request is blocked from FPP0</p> <p>Affected flags: 『N』 Z 『N』 C 『N』 AC 『N』 OV</p>
<i>stopsys</i>	<p>System halt.</p> <p>Example: <i>stopsys</i>;</p> <p>Result: Stop the system clocks and halt the system</p> <p>Affected flags: 『N』 Z 『N』 C 『N』 AC 『N』 OV</p>
<i>stopexe</i>	<p>CPU halt. The oscillator module is still active to output clock, however, system clock is disabled to save power.</p> <p>Example: <i>stopexe</i>;</p> <p>Result: Stop the system clocks and keep oscillator modules active.</p> <p>Affected flags: 『N』 Z 『N』 C 『N』 AC 『N』 OV</p>
<i>reset</i>	<p>Reset the whole chip, its operation will be same as hardware reset.</p> <p>Example: <i>reset</i>;</p> <p>Result: Reset the whole chip.</p> <p>Affected flags: 『N』 Z 『N』 C 『N』 AC 『N』 OV</p>
<i>wdreset</i>	<p>Reset Watchdog timer.</p> <p>Example: <i>wdreset</i> ;</p> <p>Result: Reset Watchdog timer.</p> <p>Affected flags: 『N』 Z 『N』 C 『N』 AC 『N』 OV</p>

7.8. Summary of Instructions Execution Cycle

2T		<i>goto, call, pcadd, ret, reti, idxm</i>
2T	Condition is fulfilled.	<i>ceqsn, cneqsn, t0sn, t1sn, dzsn, izsn</i>
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
<i>mov a, l</i>	-	-	-	-	<i>mov M, a</i>	-	-	-	-	<i>mov a, M</i>	Y	-	-	-
<i>mov a, IO</i>	Y	-	-	-	<i>mov IO, a</i>	-	-	-	-	<i>ldt16 word</i>	-	-	-	-
<i>stt16 word</i>	-	-	-	-	<i>idxm a, index</i>	-	-	-	-	<i>idxm index, a</i>	-	-	-	-
<i>xch M</i>	-	-	-	-	<i>pushaf</i>	-	-	-	-	<i>popaf</i>	Y	Y	Y	Y
<i>add a, l</i>	Y	Y	Y	Y	<i>add a, M</i>	Y	Y	Y	Y	<i>add M, a</i>	Y	Y	Y	Y
<i>addc a, M</i>	Y	Y	Y	Y	<i>addc M, a</i>	Y	Y	Y	Y	<i>addc a</i>	Y	Y	Y	Y
<i>addc M</i>	Y	Y	Y	Y	<i>sub a, l</i>	Y	Y	Y	Y	<i>sub a, M</i>	Y	Y	Y	Y
<i>sub M, a</i>	Y	Y	Y	Y	<i>subc a, M</i>	Y	Y	Y	Y	<i>subc M, a</i>	Y	Y	Y	Y
<i>subc a</i>	Y	Y	Y	Y	<i>subc M</i>	Y	Y	Y	Y	<i>inc M</i>	Y	Y	Y	Y
<i>dec M</i>	Y	Y	Y	Y	<i>clear M</i>	-	-	-	-	<i>sra</i>	-	Y	-	-
<i>src a</i>	-	Y	-	-	<i>sr M</i>	-	Y	-	-	<i>src M</i>	-	Y	-	-
<i>sl a</i>	-	Y	-	-	<i>slc a</i>	-	Y	-	-	<i>sl M</i>	-	Y	-	-
<i>slc M</i>	-	Y	-	-	<i>swap a</i>	-	-	-	-	<i>and a, l</i>	Y	-	-	-
<i>and a, M</i>	Y	-	-	-	<i>and M, a</i>	Y	-	-	-	<i>or a, l</i>	Y	-	-	-
<i>or a, M</i>	Y	-	-	-	<i>or M, a</i>	Y	-	-	-	<i>xor a, l</i>	Y	-	-	-
<i>xor IO, a</i>	-	-	-	-	<i>xor a, M</i>	Y	-	-	-	<i>xor M, a</i>	Y	-	-	-
<i>not a</i>	Y	-	-	-	<i>not M</i>	Y	-	-	-	<i>neg a</i>	Y	-	-	-
<i>neg M</i>	Y	-	-	-	<i>set0 IO.n</i>	-	-	-	-	<i>set1 IO.n</i>	-	-	-	-
<i>set0 M.n</i>	-	-	-	-	<i>set1 M.n</i>	-	-	-	-	<i>ceqsn a, l</i>	Y	Y	Y	Y
<i>ceqsn a, M</i>	Y	Y	Y	Y	<i>t0sn IO.n</i>	-	-	-	-	<i>t1sn IO.n</i>	-	-	-	-
<i>t0sn M.n</i>	-	-	-	-	<i>t1sn M.n</i>	-	-	-	-	<i>izsn a</i>	Y	Y	Y	Y
<i>dzsn a</i>	Y	Y	Y	Y	<i>izsn M</i>	Y	Y	Y	Y	<i>dzsn M</i>	Y	Y	Y	Y
<i>call label</i>	-	-	-	-	<i>goto label</i>	-	-	-	-	<i>ret l</i>	-	-	-	-
<i>ret</i>	-	-	-	-	<i>reti</i>	-	-	-	-	<i>nop</i>	-	-	-	-
<i>pcadd a</i>	-	-	-	-	<i>engint</i>	-	-	-	-	<i>disgint</i>	-	-	-	-
<i>stopsys</i>	-	-	-	-	<i>stopexe</i>	-	-	-	-	<i>reset</i>	-	-	-	-
<i>wdreset</i>	-	-	-	-	<i>swapc IO.n</i>	-	Y	-	-	<i>ceqsn a, l</i>	Y	Y	Y	Y
<i>cneqsn a, M</i>	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 cannot 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	Slow	Please refer to t_{WUP} and t_{SBP} in Section 4.1
	Fast	Please refer to t_{WUP} and 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 (ICE doesn't support.)
ICE_LPWM_INTR (Only use in simulation)	As_G01	LPWM interrupt source from LPWMG0/1 when simulation, and don't influence actual IC
	As_G2	LPWM interrupt source from LPWMG2 when simulation, and don't influence actual IC

Table 8: Code Option

9. Special Notes

This chapter is to remind user who use PMS160 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 V_{ih} and V_{il} would changes with the voltage and temperature. Please follow the minimum value of V_{ih} and the maximum value of V_{il} .
 - ◆ 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 $>33\Omega$ 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 PMS160: 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-003 to write PMS160

5S-P-003 writing all packages of PMS160 need special convert. Taking the writing of PMS160-S08B as an example, other packages only need to change the chip package in the “package setting” interface and jumper7 connection.

1. Convert the PDK file

Enter the writing interface from the IDE, then click “Convert” -> “To Package”. In the “Package Setting” interface, select the package with the suffix [P003] (as shown in Figure.26), then click “Only Program PIN” and “VDD/PA5 Swap on JP7 adapter”. After confirming information about the IC pin, save and use the newly generated PDK file. Please refer to Figure 25 and Figure 26 for specific operation steps.

PMS160

6 Touch Keys OTP Controller

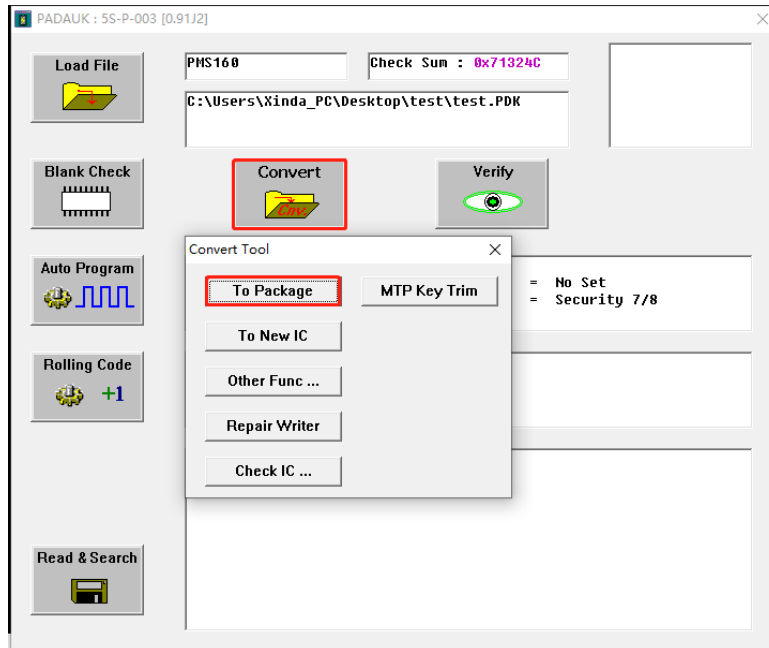


Fig. 25: convert the PDK file

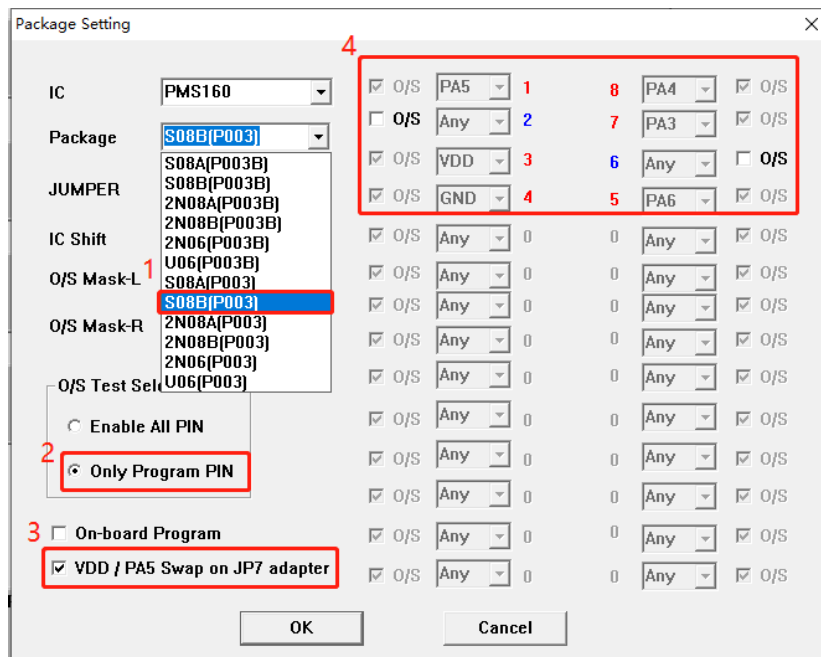


Fig. 26: PMS160-S08B package setting when using P003

2. As shown in figure 27, it is the Jumper7 connection method.

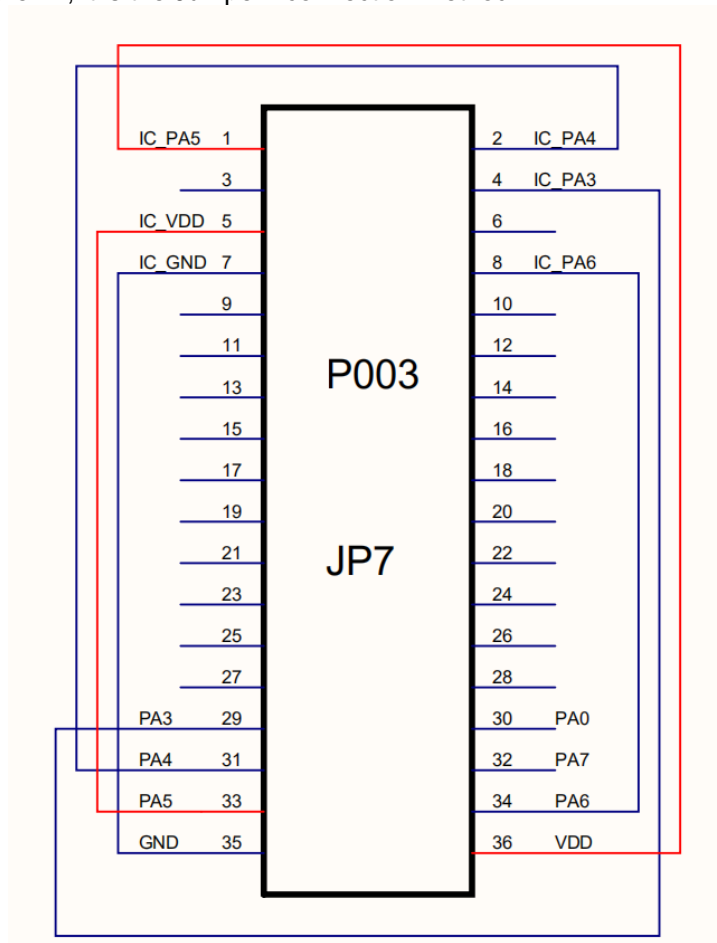


Fig. 27: schematic diagram of PMS160-S08B Jumper7 when using p003

Note: VDD / PA5 needs to swap with each other when using jumper7. For example, writer VDD-PIN connect to IC-PA5 and Writer PA5-PIN connect to IC-VDD.

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

9.1.8.2. Using 5S-P-003B to write PMS160

1. For 5S-P-003B to write PMS160-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 PMS160-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:

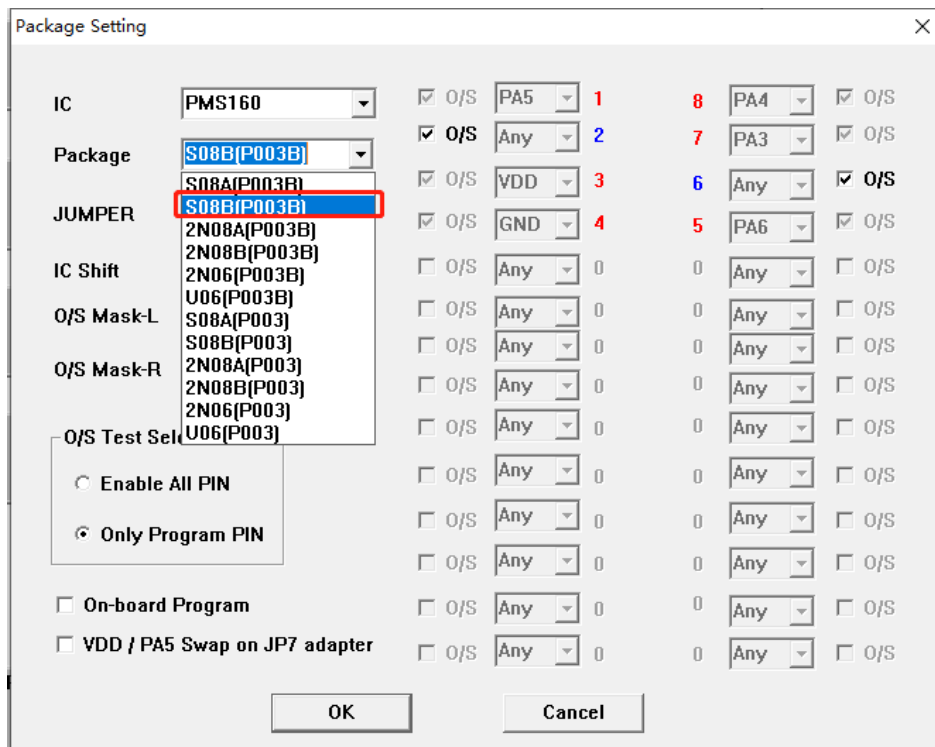


Fig. 28: PMS160-S08B package setting when using P003B

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

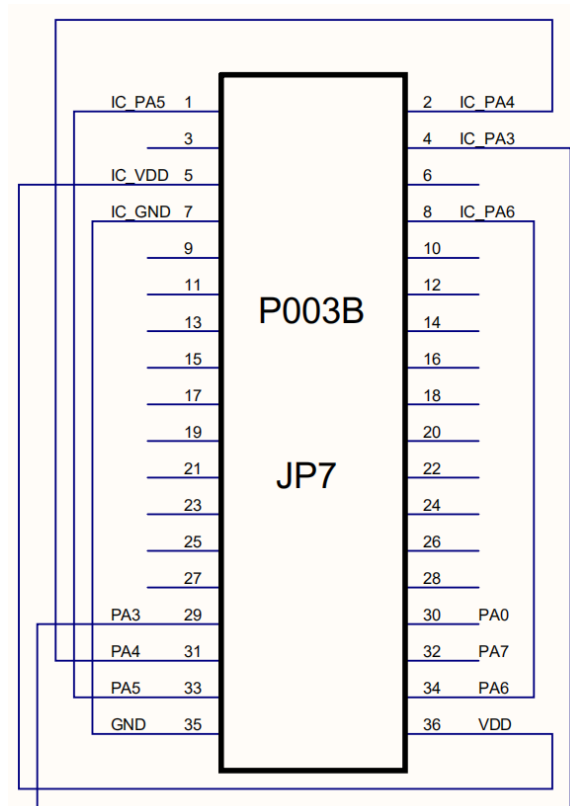


Fig. 29: schematic diagram of PMS160-S08B Jumper7 when using P003B

Note: VDD/PA5 DOES NOT need to swap with each other when using -5S-P003B Jumper7.

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

9.2. Usage of ICE

Please use 5S-I-S01/2(B) external touch pad 5S-I-TB002 (PMS160_ICE_Touch kit) to simulate PMS160. Refer to the following pictures:

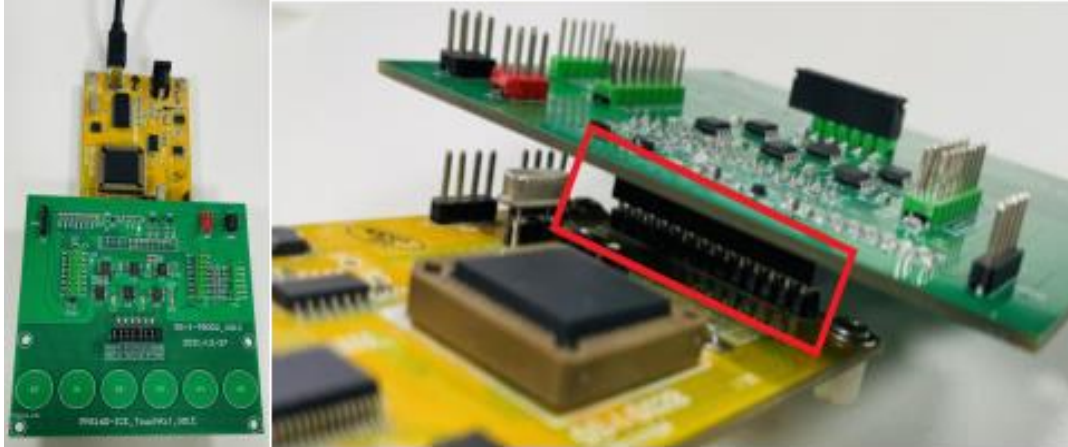


Fig. 30: PMS160 simulation tool connection diagram

Please pay attention to the following items during simulation :

(1) Simulation communication:

This touch simulation board adopts the way of external PMS160 actual IC to simulate, and the simulation of every function module is more consistent with the actual IC. But because of communication with the main simulator, the simulation speed will be slightly slowed down, so it is not recommended to enter interrupt too fast and frequently (the frequency of the simulation interrupt is recommended not to exceed 1KHz). Actual IC is not subject to this restriction.

Due to the influence of simulation communication, the oscillator module of IHRC/ILRC will always keep oscillating. Setting off IHRC/ILRC in the program will only be effective for the actual IC, so this difference should be especially noted when using IHRC/ILRC as the clock source of each Timer.

(2) Simulation voltage:

The simulation voltage range of the touch pad is limited to 3.5V~4.8V. If using 5S-I-S01 emulator and it is powered by external power supply, it is not recommended that external power supply exceeds 5V. Otherwise, the simulation may fail, or the results may be abnormal due to the voltage mismatch between the emulator and the touch pad.

(3) About Each Timer counter (Timer2/3/16, LPWMG0/1/2):

Not support Timer3, LPWMG0/1/2 selecting SYSCLK as timer clock source.

Not support Timer2, Timer3 selecting GPCRS as timer clock source.

Not support Timer2 selecting NILRC as timer clock source.

LPWMG2 and LPWMG0/1 belong to two different IC for simulation, and the simulation frequency/phase is slightly different, and LPWM function simulation with complementary dead zone is not supported. When using LPWM interrupts, the emulator also adds an additional Code Option: ICE_LPWM_INTR to select the source of LPWMG interrupts during emulation: LPWMG0/1 or LPWMG2.

LPWMG0/1/2 output pin will switch forcibly to the input state when simulation(through PAC register to force switching, and not affect the original set value of other registers such as PADIER and PAPH). After disable the LPWM output, the input/output configuration (PAC register) of the corresponding port needs to be switched by users. This operation is not required for actual IC.

(4) Power save/down: stopexe/stopsys

During the simulation, the wake-up time of power save/down mode is inconsistent with the actual IC, and is related to the system clock frequency. Taking system clock 1MHz as an example, stopexe simulation wake-up time is estimated to be between 30uS and 1mS; Stopsys wake-up time takes about 700uS. It is recommended to use actual IC measurement if there is a requirement for wake-up time.

Considering the influence of simulation communication on IHRC/ILRC, it is recommended that customers close all Timer modules which are not used to wake up before using stopexe/stopsys instead of close Timer clock source to avoid Timer false wake up. Using Timer2 as an example, the recommended program statement is "\$TM2C STOP".

(5) 5S-I-S01/2(B) doesn't support PA6 as the CIN- of the comparator, please connect to the PB6 for CIN- input when simulating. In actual chip it still in PA6.

(6) The following items should be noted when using 5S-I-S01/2(B) to emulate PMS160:

- 5S-I-S01/2(B) doesn't support SYSCLK=ILRC/16 and ILRC/64.
- 5S-I-S01/2(B) doesn't support Tm3c.NILRC wake-up function.
- 5S-I-S01/2(B) doesn't support PA5 as the interrupt source.
- 5S-I-S01/2(B) doesn't support the code options: GPC_PWM, TMx_source, TMx_bit.
- 5S-I-S01/2(B) doesn't support ALL touch function.
- The PA3 output function will be affected when GPCS selects output to PA0 output.
- When simulating PWM waveform, please check the waveform during program running. When the ICE is suspended or single-step running, its waveform may be inconsistent with the reality.
- The ILRC frequency of the 5S-I-S01/2(B) simulator is different from the actual IC and is uncalibrated, with a frequency range of about 34K~38KHz.
- Fast Wakeup time is different from 5S-I-S01/2(B): 128 SysClk, PMS160: 45 ILRC.
- Watch dog time out period is different from 5S-I-S01/2(B).

WDT period	5S-I-S01/2(B)	PMS160
misc[1:0]=00	2048 * T _{ILRC}	8192 * T _{ILRC}
misc[1:0]=01	4096 * T _{ILRC}	16384 * T _{ILRC}
misc[1:0]=10	16384 * T _{ILRC}	65536 * T _{ILRC}
misc[1:0]=11	256 * T _{ILRC}	262144 * T _{ILRC}