PMS154G

8bit OTP Type IO Controller

Datasheet

Version 0.00 – April 8, 2024

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

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

IMPORTANT NOTICE

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

PADAUK Technology products are not warranted to be suitable for use in life-support applications or other critical applications. PADAUK Technology assumes no liability for such applications. Critical applications include, but are not limited to, those that may involve potential risks of death, personal injury, fire or severe property damage.

Any programming software provided by PADAUK Technology to customers is of a service and reference nature and does not have any responsibility for software vulnerabilities. PADAUK Technology assumes no responsibility for any issue caused by a customer's product design. Customers should design and verify their products within the ranges guaranteed by PADAUK Technology. In order to minimize the risks in customers' products, customers should design a product with adequate operating safeguards.

Table of Contents

Revision History

Revision	Date	Description
0.00	2024/04/08	Preliminary version.

Usage Warning

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. (The following picture are for reference only.)

https://www.padauk.com.tw/en/technical/index.aspx?kind=24

Major Differences between PMS154B and PMS154G

Item	Function	PMS154B	PMS154G
1	Operating voltage range	2.2V ~ 5.5V	1.8V ~ 5.5V
2	11-bit PWM	One Set: PWMG0	Three Sets: PWMG0, PWMG1 & PWMG2
3	Comparator Edge	None	Code Option

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: -40°C ~ 85°C

1.2. System Features

- 2KW OTP program memory
- 128 Bytes data RAM
- One hardware 16-bit timer
- Two hardware 8-bit timer with PWM generators
- Three hardware 11-bit PWM generators
- Provide one hardware comparator
- 14 IO pins with optional pull-high resistor
- Three different IO driving capability groups to meet different application requirement
- Optional IO driving capability for each port：normal drive and low drive
- Every IO pin can be configured to enable wake-up function
- Built-in VDD/2 bias voltage generator to provide maximum 4x10 dots LCD display
- Clock sources: IHRC, ILRC & EOSC(XTAL mode)
- For every wake-up enabled IO, two optional wake-up speed are supported: normal and fast
- Eight levels of LVR: 4.0V, 3.5V, 3.0V, 2.75V, 2.5V, 2.2V, 2.0V, 1.8V
- Two external interrupt pins

1.3. CPU Features

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

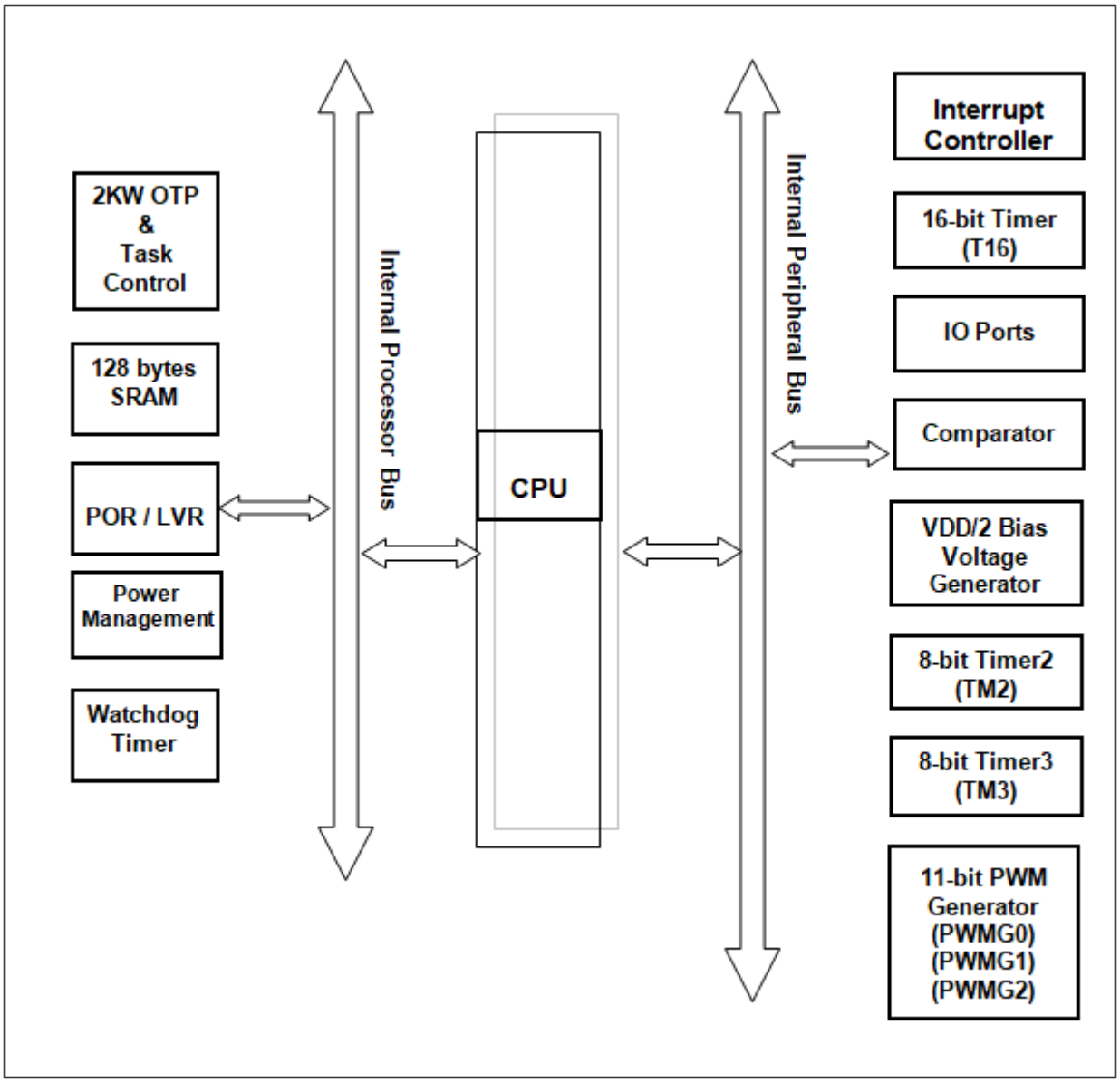
1.4. Ordering/ Package Information

- PMS154G-S16: SOP16 (150mil)
- PMS154G-S14: SOP14 (150mil)
- PMS154G-M10: MSOP10 (118mil)
- PMS154G-S08: SOP8 (150mil)
- PMS154G-U06: SOT23-6 (60mil)
- PMS154G-1J16A: QFN3*3-16pin (0.5pitch)

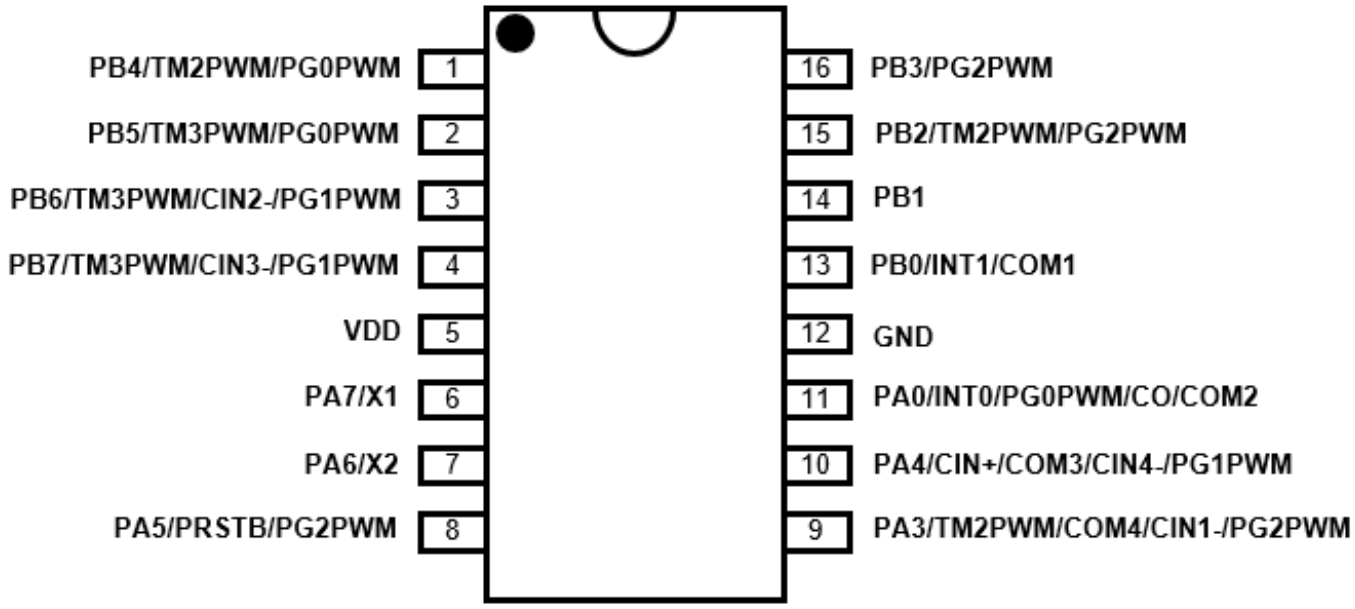
- Please refer to the official website file for package size information：”Package information”

2. General Description and Block Diagram

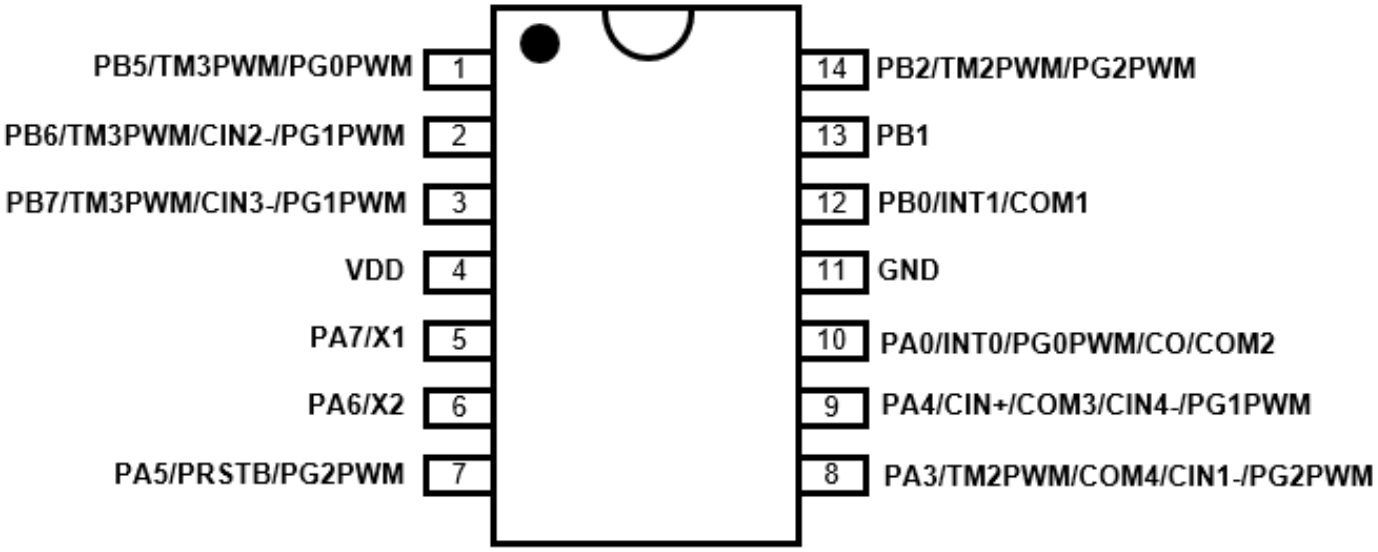
The PMS154G is an IO-Type, fully static, OTP-based CMOS 8-bit microcontroller; it employs RISC architecture and most the instructions are executed in one cycle except that few instructions are two cycles that handle indirect memory access. 2KW OTP program memory and 128 bytes data SRAM are inside, one hardware 16-bit timer, two hardware 8-bit timers with PWM generation (Timer2, Timer3) and Three hardware 11-bit timers with PWM generation (PWMG0,1,2) is also included, PMS154G also supports one hardware comparator and VDD/2 bias voltage generator for LCD display application.



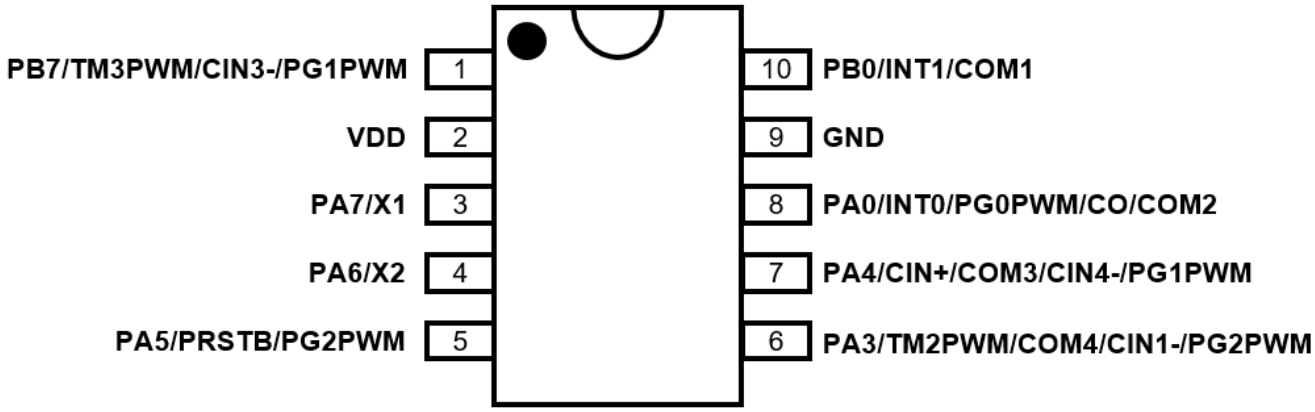
3. Pin Definition and Functional Description



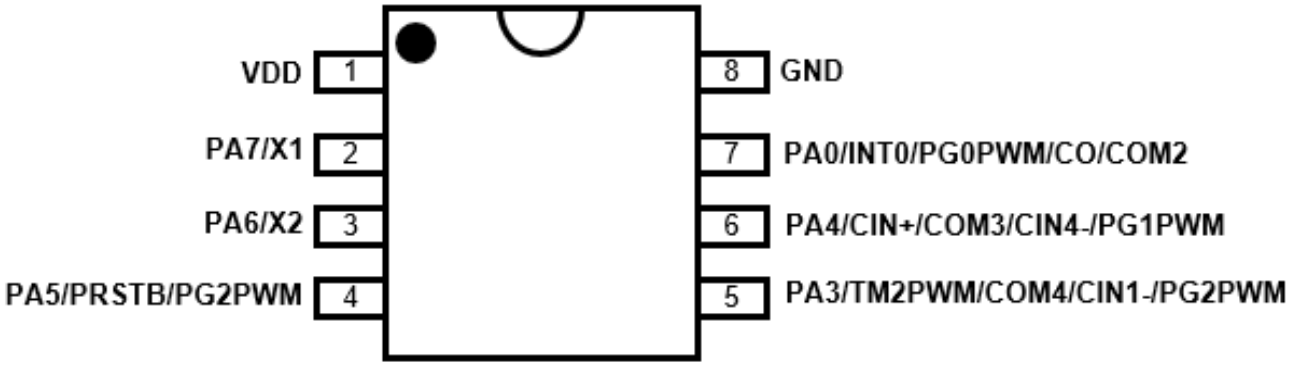
PMS154G-S16:SOP16 (150mil)



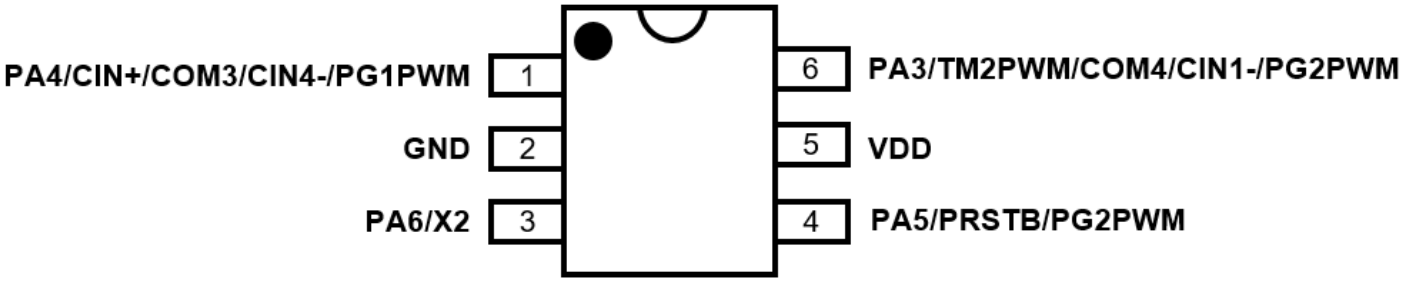
PMS154G-S14:SOP14 (150mil)



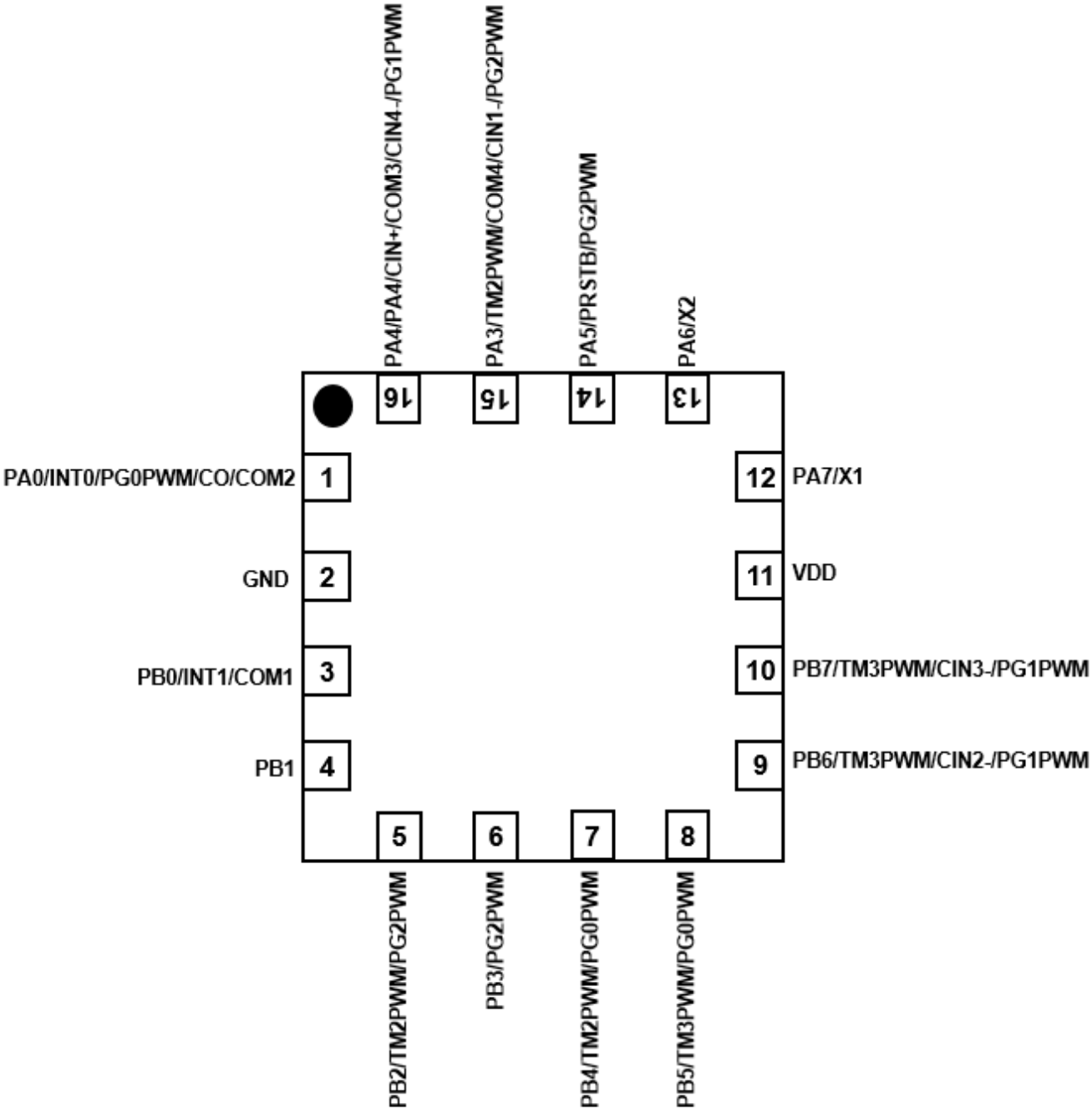
PMS154G-M10:MSOP10(118mil)



PMS154G-S08: SOP8 (150mil)



PMS154G-U06(SOT23-6 60mil)



PMS154G-1J16A: (QFN3*3-16L-0.5pitch)

Pin Name	Pin & Buffer Type	Description
PA7 / X1	IO ST / CMOS	This pin can be used as: (1) Bit 7 of port A. It can be configured as digital input or two-state output, with pull-high resistor by software independently. (2) X1 when crystal oscillator is used. When this pin is configured as crystal oscillator function, please use bit 7 of register ***padier*** to disable the digital input to prevent current leakage. This pin can be used to wake-up system during sleep mode; however, wake-up function is also disabled if bit 7 of ***padier*** register is “0”.
PA6 / X2	IO ST / CMOS	This pin can be used as: (1) Bit 6 of port A. It can be configured as digital input or two-state output, with pull-high resistor by software independently. (2) X2 when crystal oscillator is used. When this pin is configured as crystal oscillator function, please use bit 6 of register ***padier*** to disable the digital input to prevent current leakage. This pin can be used to wake-up system during sleep mode; however, wake-up function is also disabled if bit 6 of ***padier*** register is “0”.
PA5 / PRSTB / PG2PWM	IO ST / CMOS	This pin can be used as: (1) Bit 5 of port A. It can be configured as digital input or open-drain output, with pull-high resistor. (2) Hardware reset (3) Output of 11-bit PWM generator PWMG2. (ICE does NOT Support.) This pin can be used to wake-up system during sleep mode; however, wake-up function is also disabled if bit 5 of padier register is “0”. Please put 33Ω resistor in series to have high noise immunity when this pin is in input mode.
PA4 / COM3 / CIN+ / CIN4- / PG1PWM	IO ST / CMOS	This pin can be used as: (1) Bit 3 of port A. It can be configured as digital input or two-state output, with pull-high resistor by software independently. (2) COM3 to provide (VDD/2) for LCD display (3) Plus input source of comparator (4) Minus input source 4 of comparator (5) Output of 11-bit PWM generator PWMG2 This pin can be used to wake-up system during sleep mode; however, wake-up function is also disabled if bit 4 of ***padier*** register is “0”.
PA3 / TM2PWM / COM4 / CIN1- / PG2PWM	IO ST / CMOS	This pin can be used as: (1) Bit 3 of port A. It can be configured as digital input or two-state output, with pull-high resistor by software independently. (2) Minus input source 1 of comparator (3) Output of 8-bit Timer2 (TM2) (4) COM4 to provide (VDD/2) for LCD display (5) Output of 11-bit PWM generator PWMG2 This pin can be used to wake-up system during sleep mode; however, wake-up function is also disabled if bit 3 of ***padier*** register is “0”.

Pin Name	Pin & Buffer Type	Description
PA0 / INT0 / PG0PWM / CO / COM2	IO ST / CMOS	This pin can be used as: (1) Bit 0 of port A. It can be configured as digital input or two-state output, with pull-high resistor by software independently. (2) External interrupt line 0. Both rising edge and falling edge are accepted to request interrupt service. (3) Output of comparator (4) Output of 11-bit PWM generator PWMG0 (5) COM2 to provide (VDD/2) for LCD display This pin can be used to wake-up system during sleep mode; however, wake-up function is also disabled if bit 0 of ***padier*** register is “0”.
PB7 / TM3PWM / CIN3- / PG1PWM	IO ST / CMOS	This pin can be used as: (1) Bit 7 of port B. It can be configured as digital input or two-state output, with pull-high resistor by software independently. (2) Minus input source 3 of comparator. (3) Output of 8-bit timer Timer3 (TM3) (4) Output of 11-bit PWM generator PWMG1 This pin can be used to wake-up system during sleep mode; however, wake-up function is also disabled if bit 7 of ***pbdier*** register is “0”.
PB6 / TM3PWM / CIN2- / PG1PWM	IO ST / CMOS	This pin can be used as: (1) Bit 6 of port B. It can be configured as digital input or two-state output, with pull-high resistor by software independently. (2) Minus input source 2 of comparator. (3) Output of 8-bit timer Timer3 (TM3) (4) Output of 11-bit PWM generator PWMG1 This pin can be used to wake-up system during sleep mode; however, wake-up function is also disabled if bit 6 of ***pbdier*** register is “0”.
PB5 / TM3PWM / PG0PWM	IO ST / CMOS	This pin can be used as: (1) Bit 5 of port B. It can be configured as digital input or two-state output, with pull-high resistor by software independently. (2) Output of 8-bit timer Timer3 (TM3) (3) Output of 11-bit PWM generator PWMG0 This pin can be used to wake-up system during sleep mode; however, wake-up function is also disabled if bit 5 of ***pbdier*** register is “0”.
PB4 / TM2PWM / PG0PWM	IO ST / CMOS	This pin can be used as: (1) Bit 4 of port B. It can be configured as digital input or two-state output, with pull-high resistor by software independently. (2) Output of 8-bit timer Timer2 (TM2) (3) Output of 11-bit PWM generator PWMG0 This pin can be used to wake-up system during sleep mode; however, wake-up function is also disabled if bit 4 of ***pbdier*** register is “0”.

Pin Name	Pin & Buffer Type	Description
PB3 / PG2PWM	IO ST / CMOS	This pin can be used as: (1) Bit 3 of port B. It can be configured as digital input or two-state output, with pull-high resistor by software independently. (2) Output of 11-bit PWM generator PWMG2 This pin can be used to wake-up system during sleep mode; however, wake-up function is also disabled if bit 3 of ***pbdier*** register is “0”.
PB2 / TM2PWM / PG2PWM	IO ST / CMOS	This pin can be used as: (1) Bit 2 of port B. It can be configured as digital input or two-state output, with pull-high resistor by software independently. (2) Output of 8-bit timer Timer2 (TM2) (3) Output of 11-bit PWM generator PWMG2 This pin can be used to wake-up system during sleep mode; however, wake-up function is also disabled if bit 2 of ***pbdier*** register is “0”.
PB1	IO ST / CMOS	This pin can be used as: (1) Bit 1 of port B. It can be configured as digital input or two-state output, with pull-high resistor by software independently. This pin can be used to wake-up system during sleep mode; however, wake-up function is also disabled if bit 1 of ***pbdier*** register is “0”.
PB0 / INT1 / COM1	IO ST / CMOS	This pin can be used as: (1) Bit 0 of port B. It can be configured as digital input or two-state output, with pull-high resistor by software independently. (2) External interrupt line 1. Both rising edge and falling edge are accepted to request interrupt service. (3) COM1 to provide (VDD/2) for LCD display This pin can be used to wake-up system during sleep mode; however, wake-up function is also disabled if bit 0 of ***pbdier*** register is “0”.
VDD		Positive power
GND		Ground
Notes: IO: Input/Output; **ST**: Schmitt Trigger input; **CMOS**: CMOS voltage level		

4. Device Characteristics

4.1. DC/AC Characteristics

All data are acquired under the conditions of Ta = -40°C ~ 85°C, V_{DD}=3.3V, f_{SYS}=2MHz unless noted.

Symbol	Description	Min	Typ	Max	Unit	Conditions(Ta=25ºC)
V_{DD}	Operating Voltage	1.8#		5.5	V	# Subject to LVR tolerance
LVR%	Low Voltage Reset tolerance	-5		5	%	
f_{SYS}	System clock (CLK)* = IHRC/2 IHRC/4 IHRC/8 ILRC	 0 0 0	 58K	 8M 4M 2M	Hz	 V_{DD} ≧ 3.5V V_{DD} ≧ 2.5V V_{DD} ≧ 1.8V V_{DD} = 3V
V_{POR}	Power On Reset Voltage	1.7	1.8	1.9	V	
I_{OP}	Operating Current		0.51 38 72		mA uA uA	f_{SYS}=IHRC/16=1MIPS@3V f_{SYS}=ILRC=58KHz@3V f_{SYS}=EOSC=32KHz@3V
I_{PD}	Power Down Current (by ***stopsy***s command)		0.25		uA	f_{SYS}= 0Hz, V_{DD} =3.3V
I_{PS}	Power Save Current (by ***stopexe*** command) *Disable IHRC		3		uA	V_{DD} =3.3V
V_{IL}	Input low voltage for IO lines	0		0.1 V_{DD}	V	
V_{IH}	Input high voltage for IO lines	0.8 V_{DD} 0.6 V_{DD}		V_{DD} V_{DD}		PA5 other IO
I_{OL}	IO lines sink current (normal)					
	*PA0,PA3,PA4,PB2,PB5,PB6		20		mA	V_{DD}=5V, V_{OL}=0.5V
	*Others		12			
	IO lines sink current (low)					
	*PA5		11		mA	V_{DD}=5V, V_{OL}=0.5V
	*Others		4			
I_{OH}	PA5 Other IO lines drive current (normal)		0 -13		mA	V_{DD}=3.3V, V_{OH}=2.97V
	PA5 Other IO lines drive current (low)		0 -4			
V_{IN}	Input voltage	-0.3		V_{DD} + 0.3	V	
$I_{INJ\ (PIN)}$	Injected current on pin			1	mA	V_{DD} +0.3≧V_{IN}≧ -0.3
R_{PH}	Pull-high Resistance		83		KΩ	V_{DD}=5V
f_{IHRC}	Frequency of IHRC after calibration *	15.84*	16*	16.16*	MHz	V_{DD}=5V, Ta=25ºC
		15.20*		16.80*		V_{DD} =2.0V~5.5V, -40ºC <Ta<85ºC*
		13.60*		18.40*		V_{DD} =1.8V~5.5V, -40ºC <Ta<85ºC*

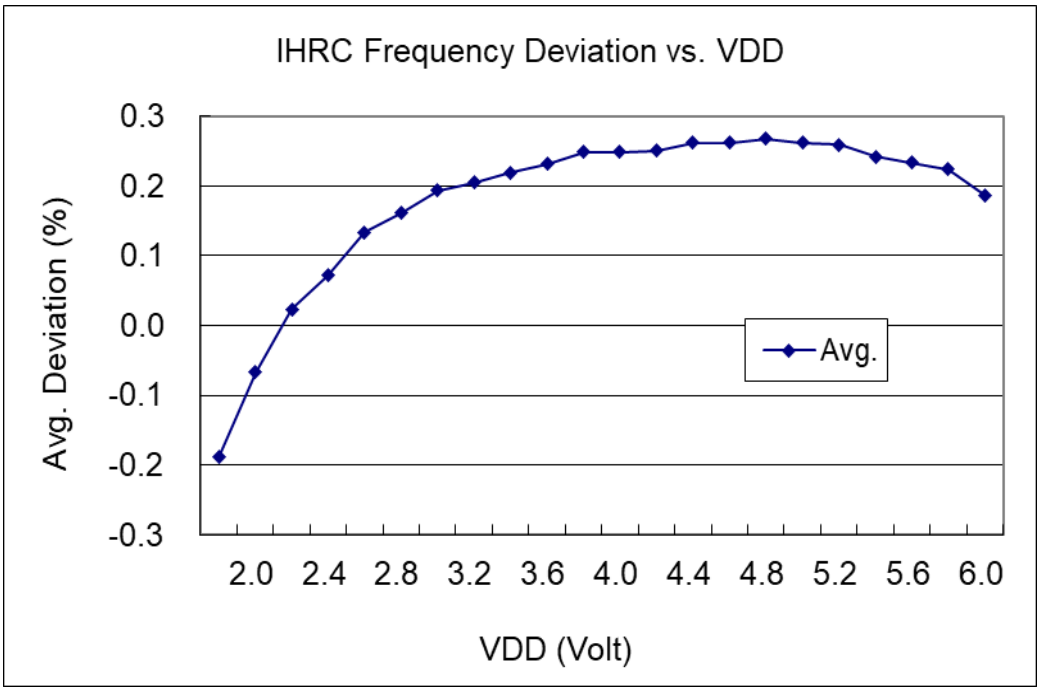
Symbol	Description	Min	Typ	Max	Unit	Conditions(Ta=25°C)
t_{INT}	Interrupt pulse width	30			ns	V_{DD} = 3.3V
V_{DR}	RAM data retention voltage*	1.5			V	In power-down mode
t_{WDT}	Watchdog timeout period		8192		T_{ILRC}	misc[1:0]=00 (default)
			16384			misc[1:0]=01
			65536			misc[1:0]=10
			262144			misc[1:0]=11
t_{SBP}	System boot-up period from power-on for Slow boot-up		56		ms	@ V_{DD} =5V
	System boot-up period from power-on for Fast boot-up		940		us	
t_{WUP}	Wake-up time period for fast wake-up		45		T_{ILRC}	Where T_{ILRC} is the time period of ILRC
	Wake-up time period for slow wake-up		3000			
t_{RST}	External reset pulse width	120			us	
CPos	Comparator offset*	-	±10	±20	mV	
CPcm	Comparator input common mode*	0		V_{DD} -1.5	V	
CPspt	Comparator response time**		100	500	ns	Both rising and falling
CPmc	Stable time to change comparator mode		2.5	7.5	us	
CPcs	Comparator current consumption		30		uA	V_{DD} = 5V

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

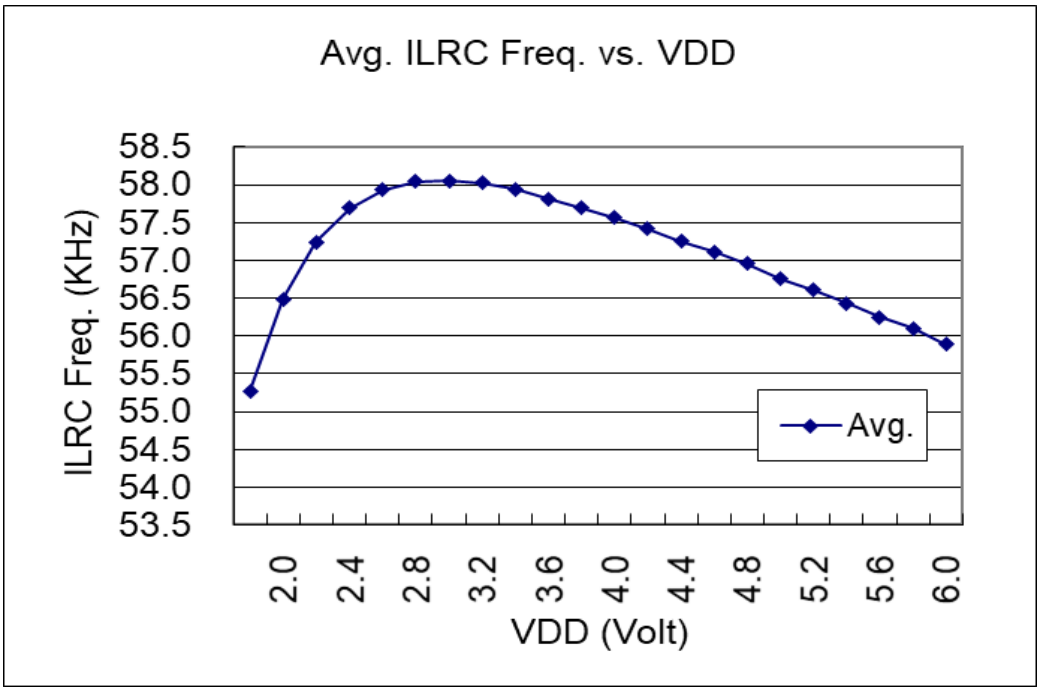
4.2. Absolute Maximum Ratings

- Supply Voltage ..... 1.8V ~ 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 …… -40°C ~ 85°C
- Storage Temperature …… -50°C ~ 125°C
- Junction Temperature …….. 150°C

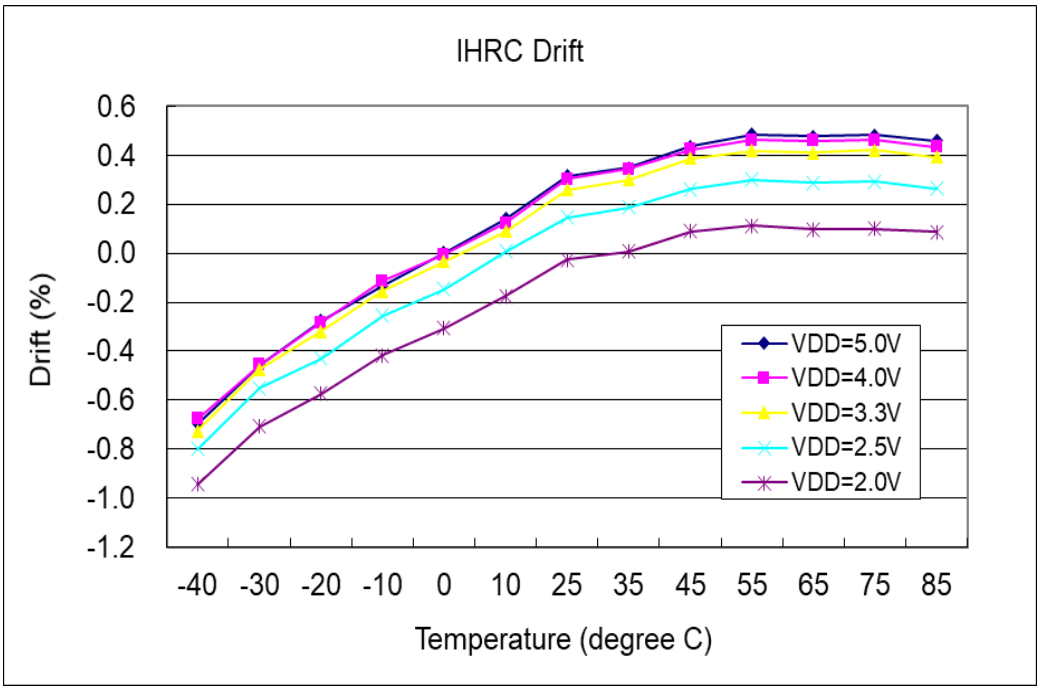
4.3. Typical IHRC Frequency vs. VDD (calibrated to 16MHz)



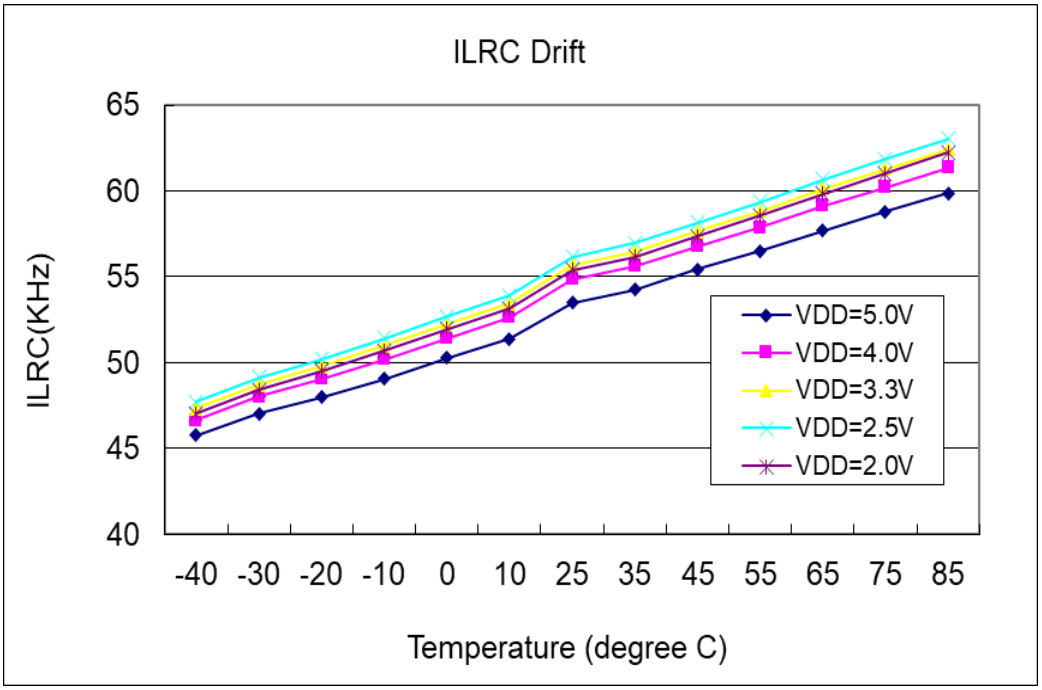
4.4. Typical ILRC Frequency vs. VDD



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



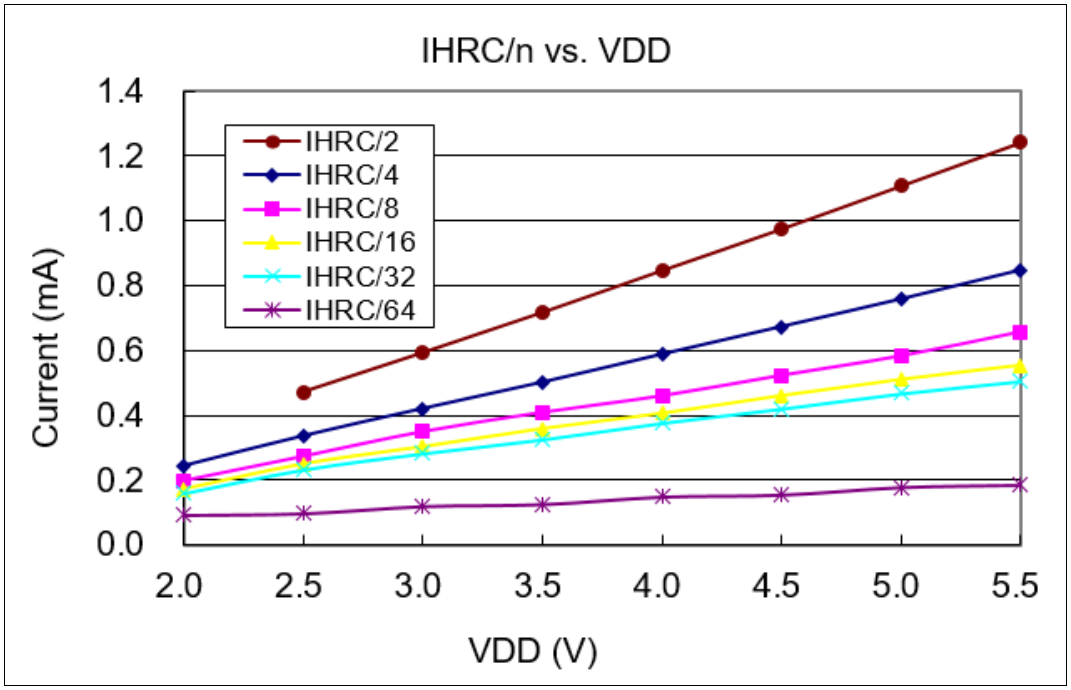
4.6. Typical ILRC Frequency vs. Temperature



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

Conditions: **ON**: IHRC; **OFF**: Bandgap, LVR, T16 modules, ILRC modules;

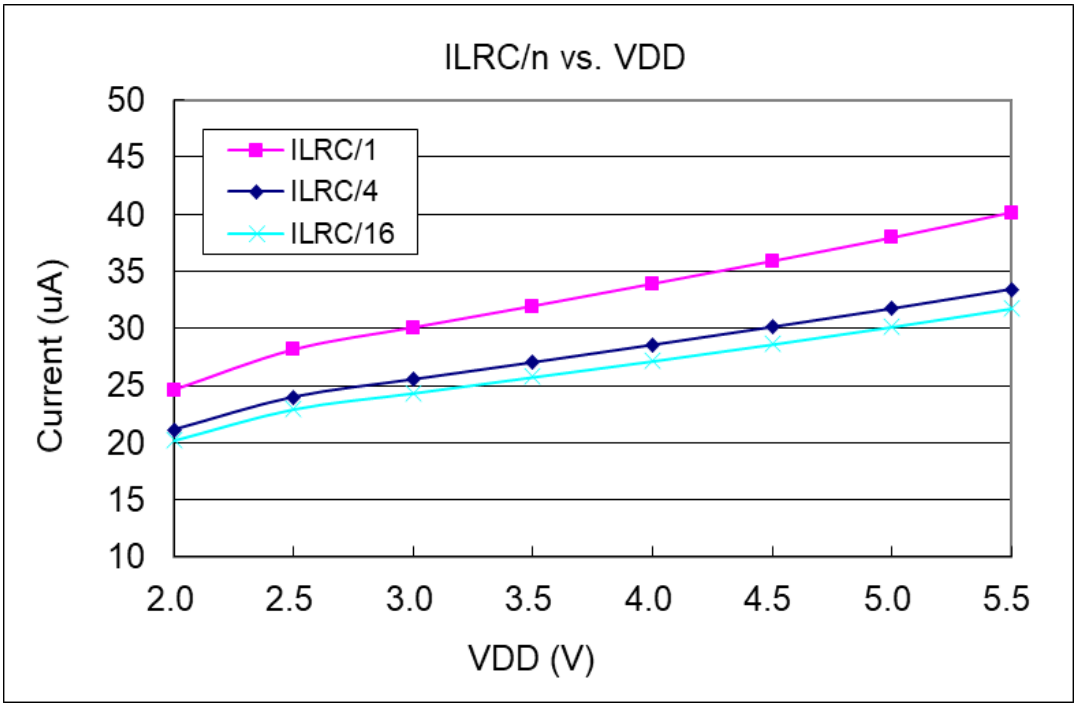
IO: PA0:0.5Hz output toggle and no loading, **others**: input and no floating



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

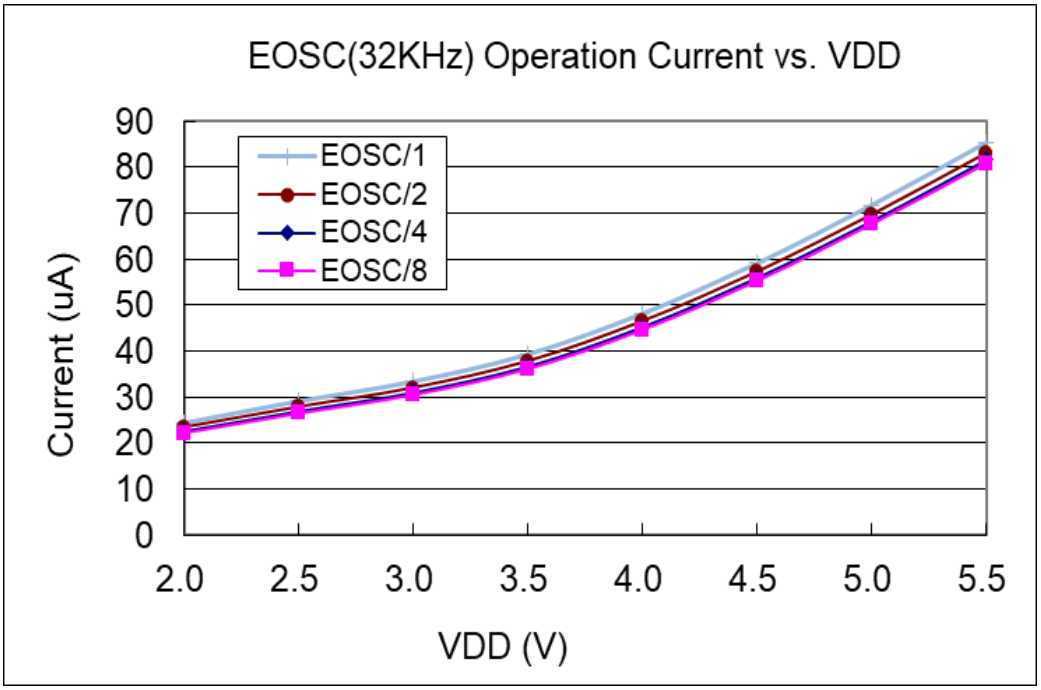
Conditions: **ON**: ILRC; **OFF**: Bandgap, LVR, T16 modules, IHRC modules;

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



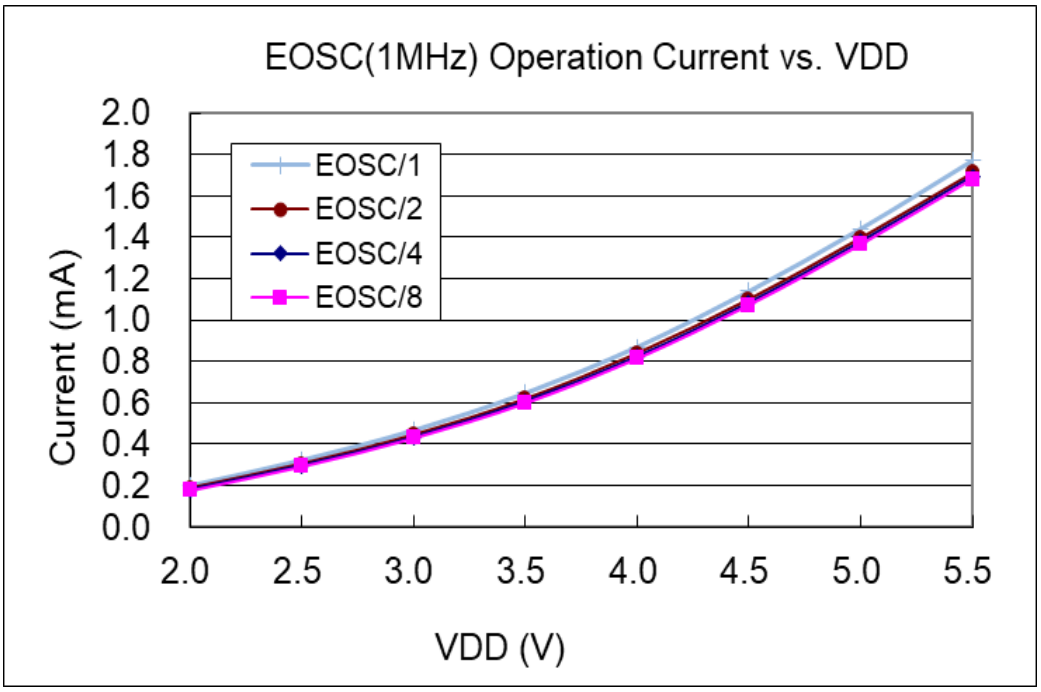
4.9. Typical Operating Current vs. VDD and CLK=32KHz EOSC / n

Conditions: **ON**: EOSC; **OFF**: Bandgap, LVR, T16 modules, IHRC, ILRC modules;
IO: PA0:0.5Hz output toggle and no loading, **others**: input and no floating



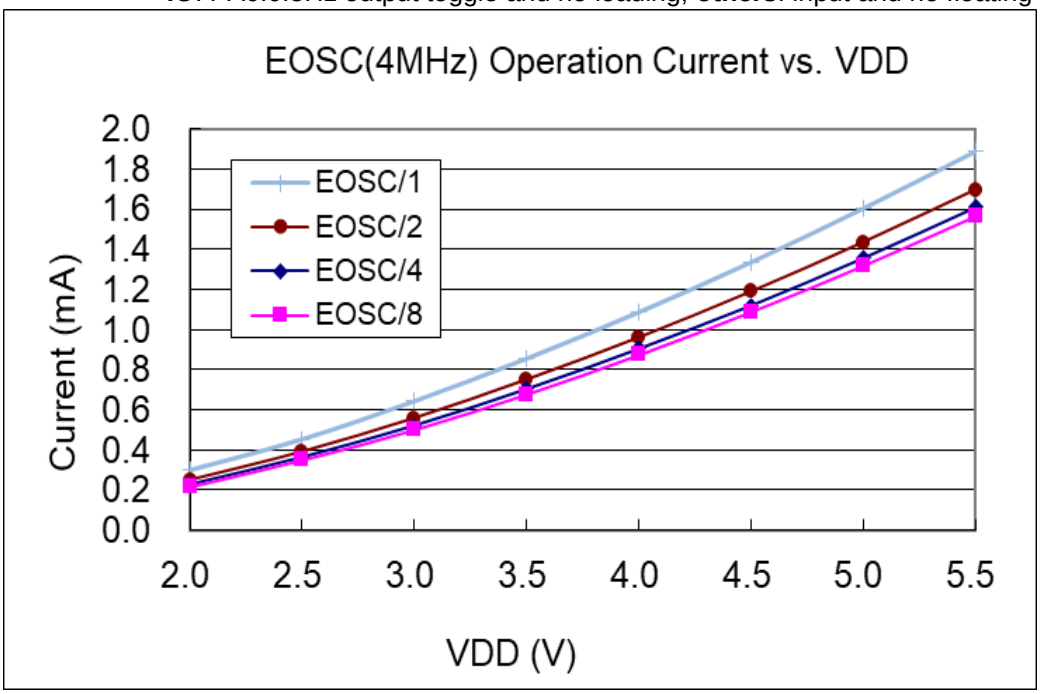
4.10. Typical Operating Current vs. VDD and CLK=1MHz EOSC / n

Conditions: **ON**: EOSC; **OFF**: Bandgap, LVR, T16 modules, IHRC, ILRC modules;
IO: PA0:0.5Hz output toggle and no loading, **others**: input and no floating

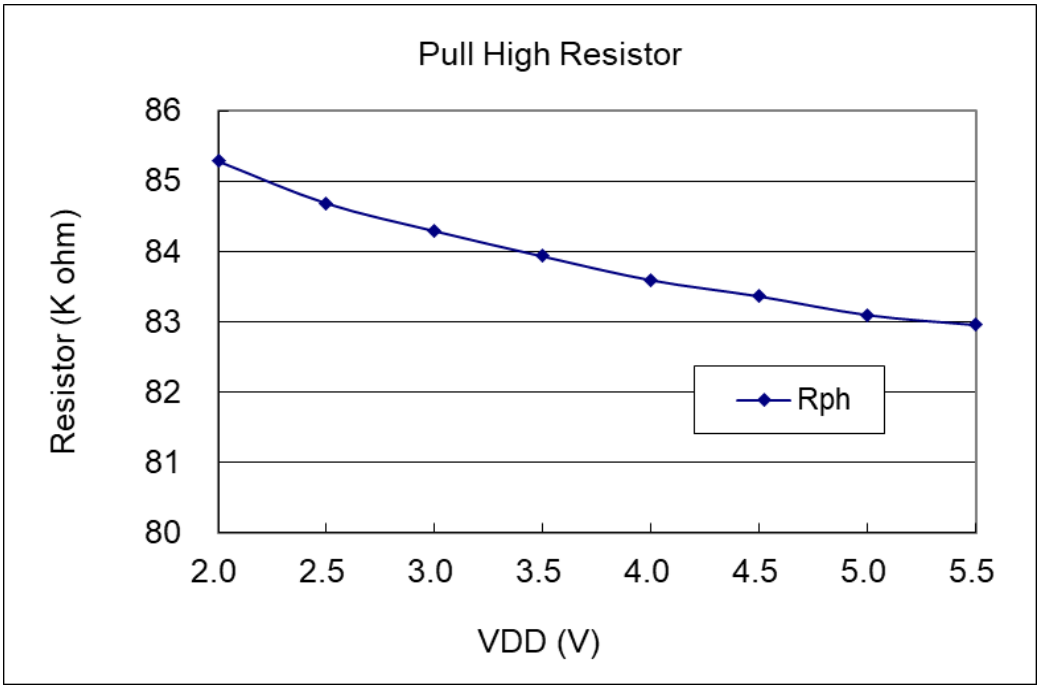


4.11. Typical Operating Current vs. VDD and CLK=4MHz EOSC / n

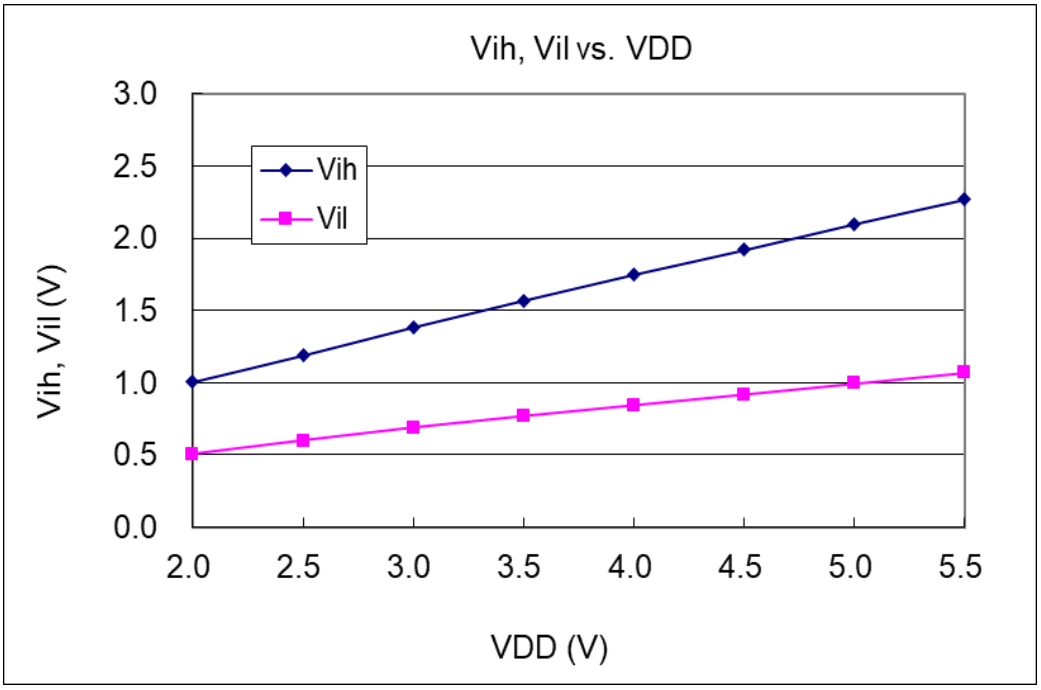
Conditions: **ON**: EOSC; **OFF**: Bandgap, LVR, T16 modules, IHRC, ILRC modules;
IO: PA0:0.5Hz output toggle and no loading, **others**: input and no floating



4.12. Typical IO pull high resistance

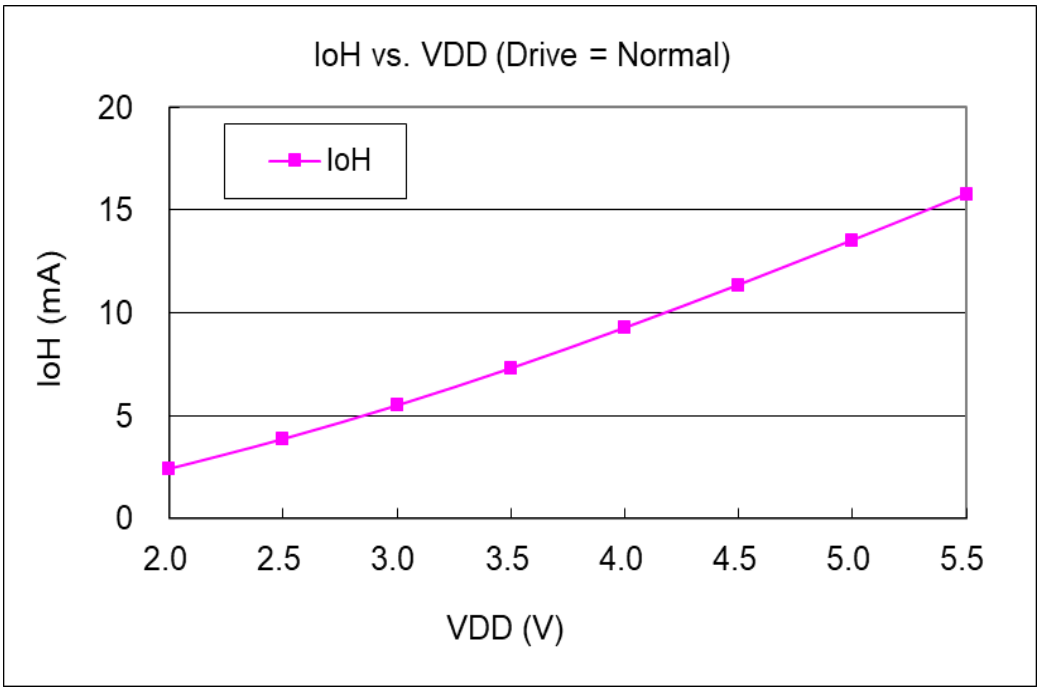


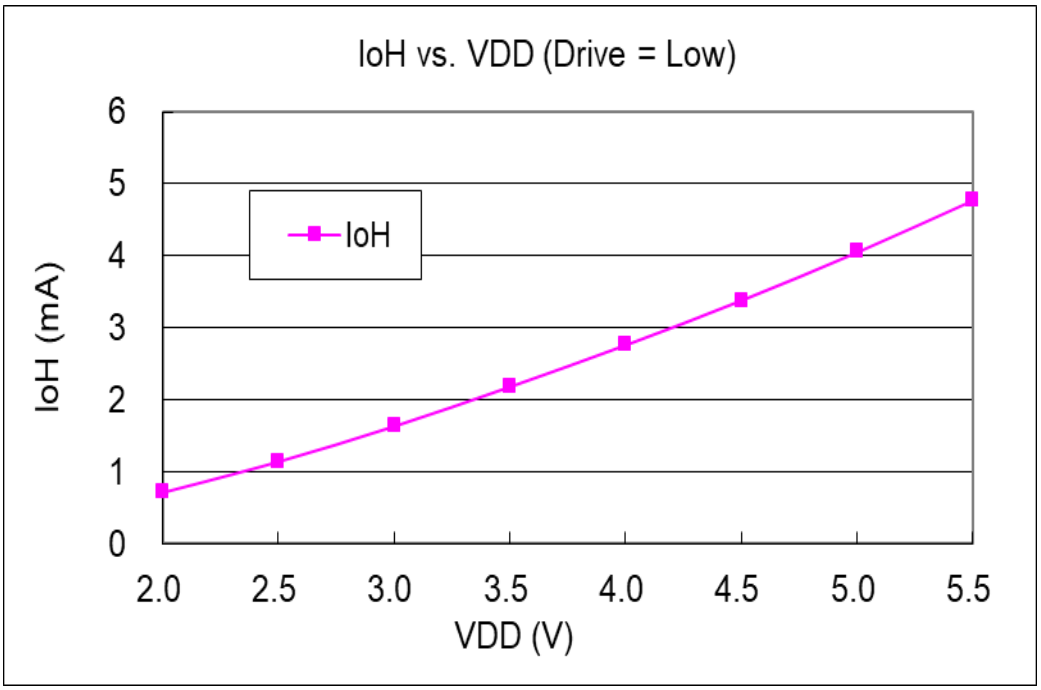
4.13. Typical IO input high/low threshold voltage (V_{IH}/V_{IL})

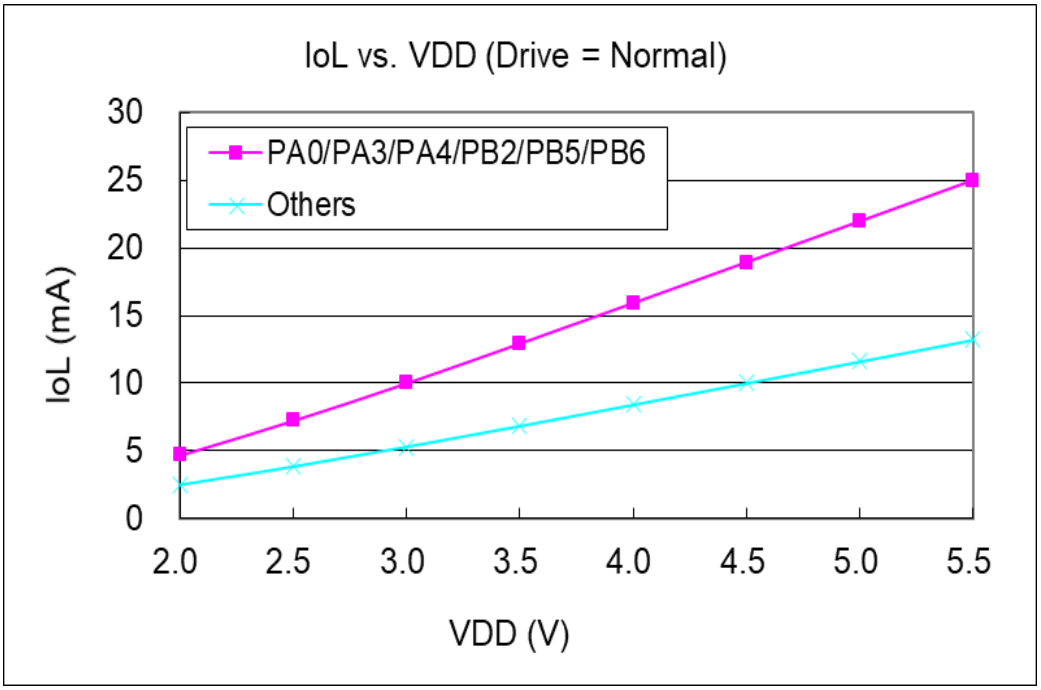


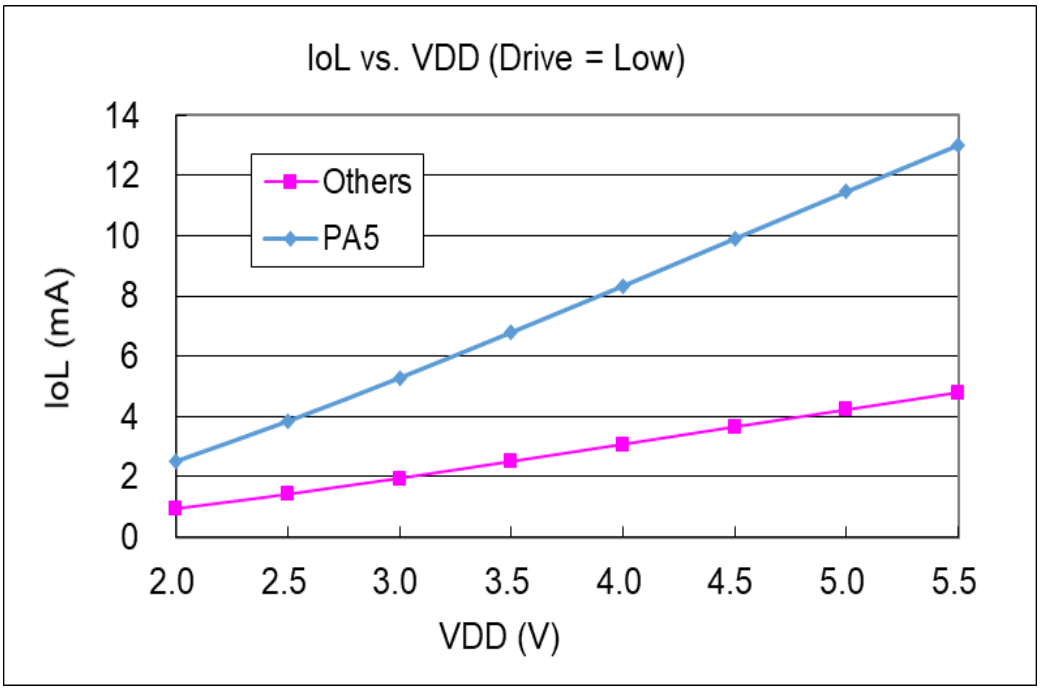
4.14. Typical IO driving current (I_{OH}) and sink current (I_{OL})

(VOH=0.9*VDD, VOL=0.1*VDD)

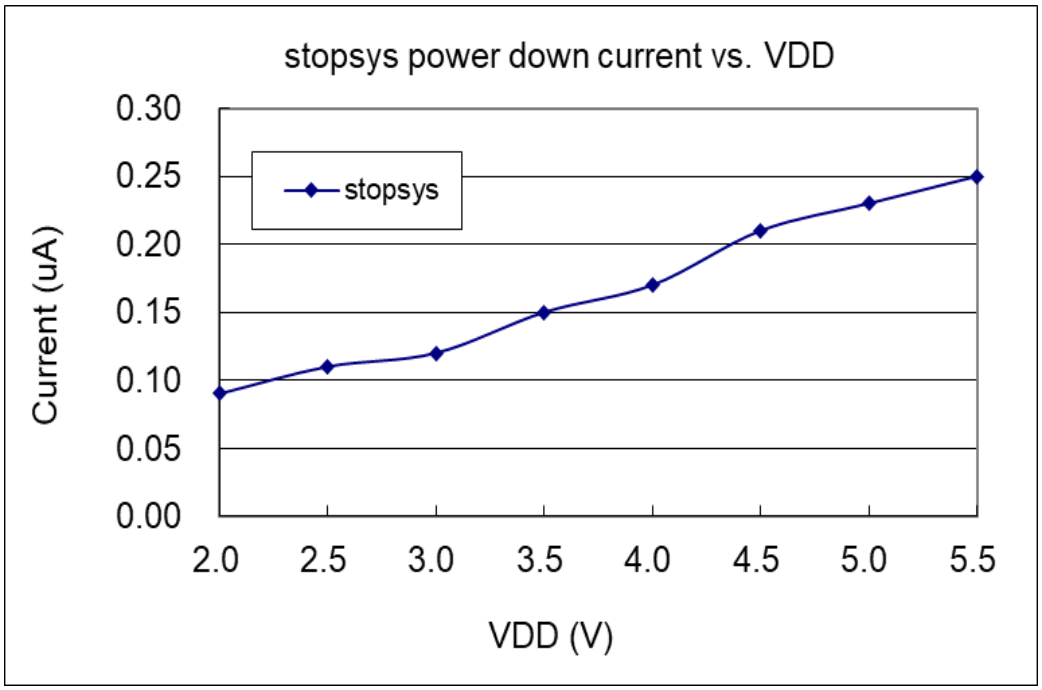


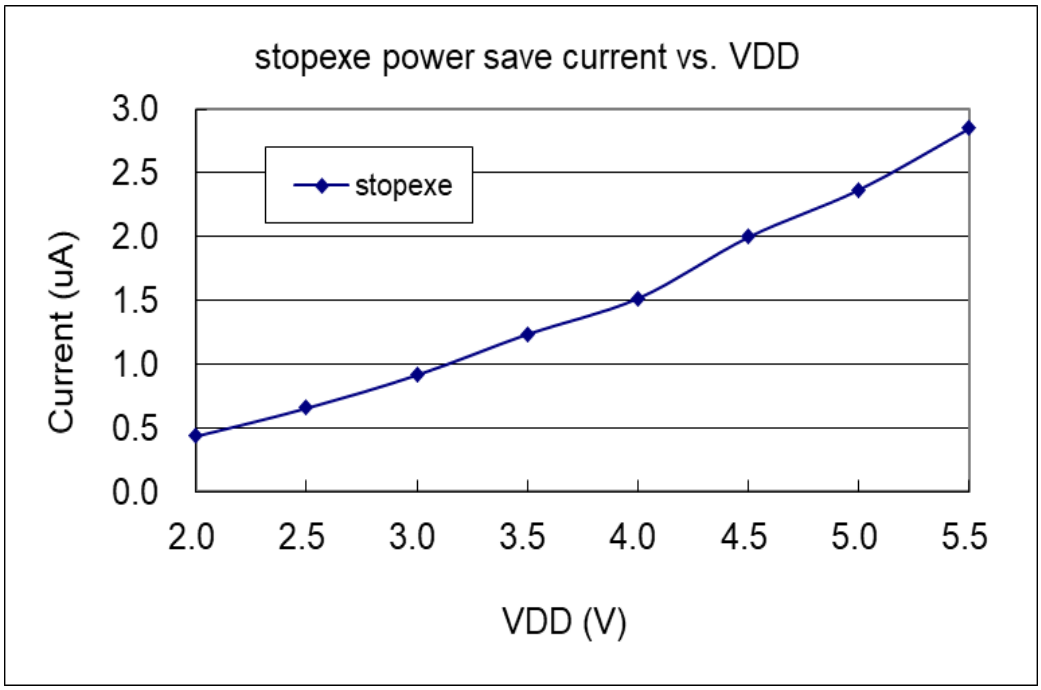






4.15. Typical power down current (IPD) and power save current (IPS)





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 0x000 is reserved for system using, so the program will start from 0x001 which is GOTO FPPA0 instruction usually. The interrupt entry is 0x010 if used, the last 16 addresses are reserved for system using, like checksum, serial number, etc. The OTP program memory for PMS154G is 2KW that is partitioned as Table 1. The OTP memory from address 0x7E8 to 0x7FF is for system using, address space from 0x002 to 0x00F and from 0x011 to 0x7E7 is user program space.

Address	Function
0x000	System Using
0x001	GOTO FPPA0 instruction
0x002	User program
•	•
0x00F	User program
0x010	Interrupt entry address
0x011	User program
•	•
0x7E7	User program
0x7E8	System Using
•	•
0x7FF	System Using

Table 1: Program Memory Organization

5.2. Boot-Up

POR (Power-On-Reset) is used to reset PMS154G when power up. The boot-up time can be optional fast or normal. Time for fast boot-up is about 45 ILRC clock cycles whereas 3000 ILRC clock cycles for normal boot-up. Customer must ensure the stability of supply voltage after power up no matter which option is chosen, the power up sequence is shown in the Fig. 1 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.

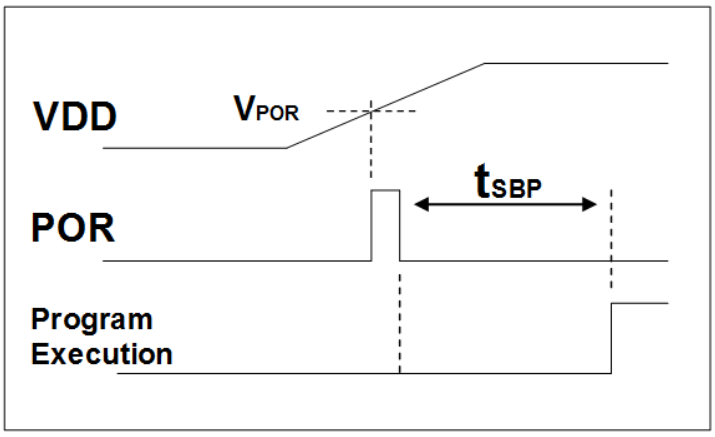
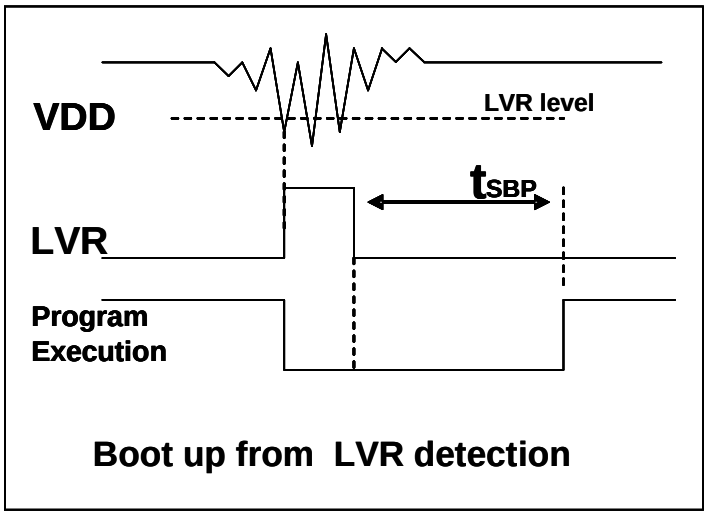
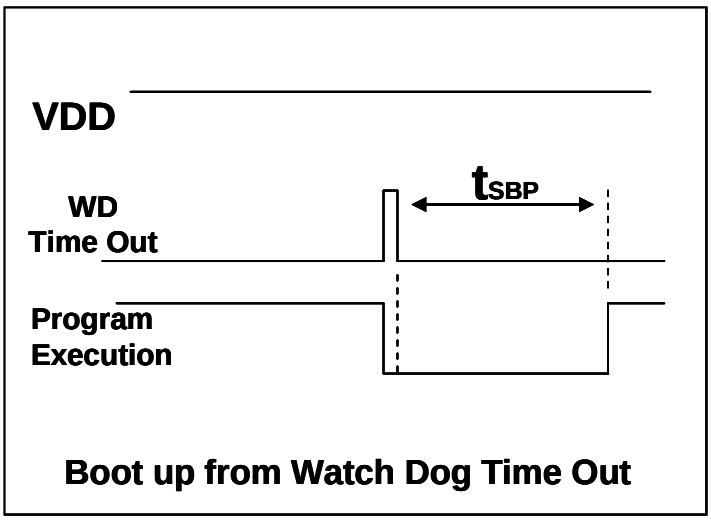
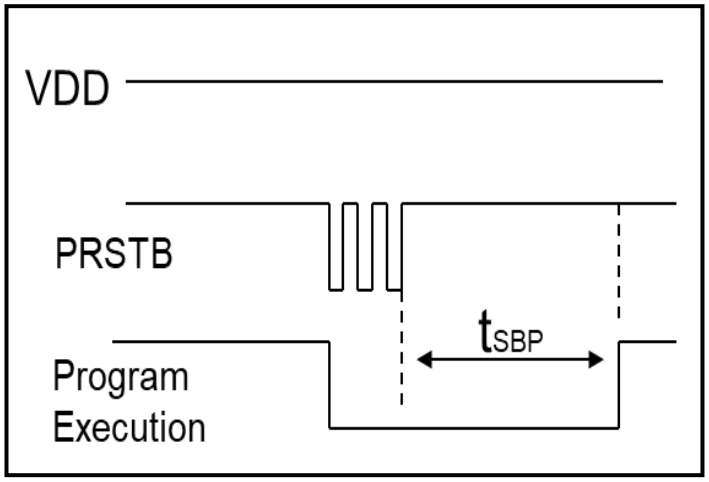


Fig. 1: Power Up Sequence

5.2.1. Timing charts for reset conditions







5.3. Data Memory – SRAM

The access of data memory can be byte or bit operation. Besides data storage, the SRAM data memory is also served as data pointer of indirect access method and the stack memory.

The stack memory is defined in the data memory. The stack pointer is defined in the stack pointer register; the depth of stack memory of each processing unit is defined by the user. The arrangement of stack memory fully flexible and can be dynamically adjusted by the user.

For indirect memory access mechanism, the data memory is used as the data pointer to address the data byte. All the data memory could be the data pointer; it's quite flexible and useful to do the indirect memory access. All the 128 bytes data memory of PMS154G can be accessed by indirect access mechanism.

5.4. Oscillator and clock

There are three oscillator circuits provided by PMS154G: external crystal oscillator (EOSC), internal high RC oscillator (IHRC) and internal low RC oscillator (ILRC), and these three oscillators are enabled or disabled by registers eoscr.7, clkmd.4 and clkmd.2 independently. User can choose one of these three oscillators as system clock source and use ***clkmd*** register to target the desired frequency as system clock to meet different applications.

Oscillator Module	Enable / Disable
EOSC	eoscr.7
IHRC	clkmd.4
ILRC	clkmd.2

Table2: Three Oscillator Circuits provided by PMS154G

5.4.1. Internal High RC oscillator and Internal Low RC oscillator

After boot-up, the IHRC and ILRC oscillators are enabled. The frequency of IHRC can be calibrated to eliminate process variation by ***ihrcr*** register; normally it is calibrated to 16MHz. 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.

5.4.2. IHRC calibration

The IHRC frequency may be different chip by chip due to manufacturing variation, PMS154G provide the IHRC frequency calibration to eliminate this variation, and this function can be selected when compiling user's program and the command will be inserted into user's program automatically. The calibration command is shown as below:

.ADJUST_IC SYSCLK=IHRC/(**p1**), IHRC=(**p2**)MHz, V_{DD}=(**p3**)V

Where,

p1=2, 4, 8, 16, 32; In order to provide different system clock.

p2=16 ~ 18; In order to calibrate the chip to different frequency, 16MHz is the usually one.

p3=1.8 ~ 5.5; In order to calibrate the chip under different supply voltage.

5.4.3. IHRC Frequency Calibration and System Clock

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

SYSCLK	CLKMD	IHRCR	Description
○ Set IHRC / 2	= 34h (IHRC / 2)	Calibrated	IHRC calibrated to 16MHz, CLK=8MHz (IHRC/2)
○ Set IHRC / 4	= 14h (IHRC / 4)	Calibrated	IHRC calibrated to 16MHz, CLK=4MHz (IHRC/4)
○ Set IHRC / 8	= 3Ch (IHRC / 8)	Calibrated	IHRC calibrated to 16MHz, CLK=2MHz (IHRC/8)
○ Set IHRC / 16	= 1Ch (IHRC / 16)	Calibrated	IHRC calibrated to 16MHz, CLK=1MHz (IHRC/16)
○ Set IHRC / 32	= 7Ch (IHRC / 32)	Calibrated	IHRC calibrated to 16MHz, CLK=0.5MHz (IHRC/32)
○ Set ILRC	= E4h (ILRC / 1)	Calibrated	IHRC calibrated to 16MHz, CLK=ILRC
○ Disable	No change	No Change	IHRC not calibrated, CLK not changed, Bandgap OFF

Table 3: Options for IHRC Frequency Calibration

Usually, .ADJUST_IC will be the first command after boot up, in order to set the target operating frequency whenever stating the system. The program code for IHRC frequency calibration is executed only one time that occurs in writing the codes into OTP memory; after then, it will not be executed again. If the different option for IHRC calibration is chosen, the system status is also different after boot. The following shows the status of PMS154G for different option:

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

After boot, CLKMD = 0x34：

- IHRC frequency is calibrated to 16MHz@V_{DD}=5V and IHRC module is enabled
- System CLK = IHRC/2 = 8MHz
- Watchdog timer is disabled, ILRC is enabled, PA5 is in input mode

(2) .ADJUST_IC SYSCLK=IHRC/4, IHRC=16MHz, V_{DD}=3.3V

After boot, CLKMD = 0x14：

- IHRC frequency is calibrated to 16MHz@V_{DD}=3.3V and IHRC module is enabled
- System CLK = IHRC/4 = 4MHz
- Watchdog timer is disabled, ILRC is enabled, PA5 is in input mode

(3) .ADJUST_IC SYSCLK=IHRC/8, IHRC=16MHz, V_{DD}=2.5V

After boot, CLKMD = 0x3C：

- IHRC frequency is calibrated to 16MHz@V_{DD}=2.5V and IHRC module is enabled
- System CLK = IHRC/8 = 2MHz
- Watchdog timer is disabled, ILRC is enabled, PA5 is in input mode

(4) .ADJUST_IC SYSCLK=IHRC/16, IHRC=16MHz, V_{DD}=2.2V

After boot, CLKMD = 0x1C：

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

(5) .ADJUST_IC SYSCLK=IHRC/32, IHRC=16MHz, V_{DD}=5V

After boot, CLKMD = 0x7C：

- IHRC frequency is calibrated to 16MHz@V_{DD}=5V and IHRC module is enabled
- System CLK = IHRC/32 = 500KHz
- Watchdog timer is disabled, ILRC is enabled, PA5 is in input mode

(6) .ADJUST_IC SYSCLK=ILRC, IHRC=16MHz, V_{DD}=5V

After boot, CLKMD = 0xE4：

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

(7) .ADJUST_IC DISABLE

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

- IHRC is not calibrated and IHRC module is disabled
- System CLK = ILRC or IHRC/64 (by Boot-up_Time)
- Watchdog timer is enabled, ILRC is enabled, PA5 is in input mode

5.4.4. External Crystal Oscillator

If crystal oscillator is used, a crystal or resonator is required between X1 and X2. Fig. 2 shows the hardware connection under this application; the range of operating frequency of crystal oscillator can be from 32 KHz to 4MHz, depending on the crystal placed on; higher frequency oscillator than 4MHz is NOT supported.

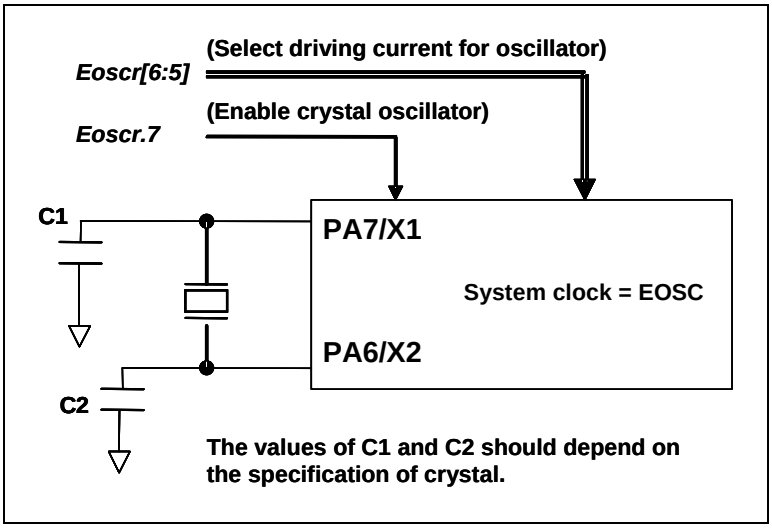


Fig. 2: Connection of crystal oscillator

Besides crystal, external capacitor and options of PMS154G should be fine-tuned in *eoscr* (0x0a) register to have good sinusoidal waveform. The *eoscr*.7 is used to enable crystal oscillator module, *eoscr*.6 and *eoscr*.5 are used to set the different driving current to meet the requirement of different frequency of crystal oscillator:

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

Table 4 shows the recommended values of C1 and C2 for different crystal oscillator; the measured start-up time under its corresponding conditions is also shown. Since the crystal or resonator had its own characteristic, the capacitors and start-up time may be slightly different for different type of crystal or resonator, please refer to its specification for proper values of C1 and C2.

Frequency	C1	C2	Measured Start-up time	Conditions
4MHz	4.7pF	4.7pF	6ms	(*eoscr*[6:5]=11)
1MHz	10pF	10pF	11ms	(*eoscr*[6:5]=10)
32KHz	22pF	22pF	450ms	(*eoscr*[6:5]=01)

Table 4: Recommend values of C1 and C2 for crystal and resonator oscillators

When using the crystal oscillator, user must pay attention to the stable time of oscillator after enabling it, the stable time of oscillator will depend on frequency, crystal type, external capacitor and supply voltage. Before switching the system to the crystal oscillator, user must make sure the oscillator is stable; the reference program is shown as below:

```
void    FPPA0 (void)
{
    . ADJUST_IC   SYSCLK=IHRC/16, IHRC=16MHz, VDD=5V
    ...
    $    EOSCR   Enable, 4Mhz;        // EOSCR = 0b111_00000;
    $ T16M    EOSC，/1，BIT13;        // while T16.Bit13 0 => 1, Intrq.T16 => 1
                                      // suppose crystal EOSC is stable
    WORD    count    =    0;
    stt16   count;
    Intrq.T16   =    0;
    while (¡ Intrq.T16)   NULL;       // count from 0x0000 to 0x2000, then trigger INTRQ.T16
    clkmd = 0xB4;                     // switch system clock to EOSC;
    clkmd.4 = 0;                      // disable IHRC
    ...
}
```

Please notice that the crystal oscillator should be fully turned off before entering the power-down mode, in order to avoid unexpected wakeup event.

5.4.5. System Clock and LVR levels

The clock source of system clock comes from IHRC, ILRC or EOSC, the hardware diagram of system clock in the PMS154G is shown as Fig. 3.

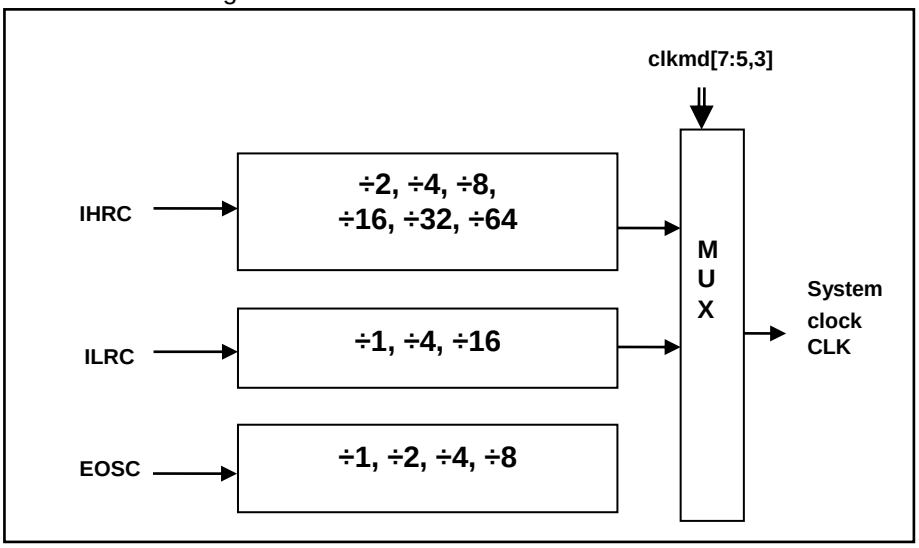


Fig. 3: Options of System Clock

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

5.4.6. System Clock Switching

After IHRC calibration, user may want to switch system clock to a new frequency or may switch system clock at any time to optimize the system performance and power consumption. Basically, the system clock of PGR431 can be switched among IHRC, ILRC and EOSC by setting the ***clkmd*** register at any time; system clock will be the new one after writing to ***clkmd*** register immediately. Please notice that the original clock module can NOT be turned off at the same time as writing command to ***clkmd*** register. The examples are shown as below and more information about clock switching, please refer to the "Help" -> "Application Note" -> "IC Introduction" -> "Register Introduction" -> CLKMD".

Case 1: Switching system clock from ILRC to IHRC/2

```
...                             //  system clock is ILRC
CLKMD.4     =   1;              //  turn on IHRC first to improve anti-interference ability
CLKMD       =   0x34 ;          //  switch to IHRC/2, ILRC CAN NOT be disabled here
//  CLKMD.2 =   0 ;             //  if need, ILRC CAN be disabled at this time
...
```

Case 2: Switching system clock from ILRC to EOSC

```
...                             //  system clock is ILRC
CLKMD       =   0xA6 ;          //  switch to IHRC, ILRC CAN NOT be disabled here
CLKMD.2     =   0 ;             //  ILRC CAN be disabled at this time
...
```

Case 3: Switching system clock from IHRC/2 to ILRC

```
...                             //  system clock is IHRC/2
CLKMD       =   0xF4 ;          //  switch to ILRC, IHRC CAN NOT be disabled here
CLKMD.4     =   0 ;             //  IHRC CAN be disabled at this time
...
```

Case 4: Switching system clock from IHRC/2 to EOSC

```
...                             //  system clock is IHRC/2
CLKMD       =   0XB0 ;          //  switch to EOSC, IHRC CAN NOT be disabled here
CLKMD.4     =   0 ;             //  IHRC CAN be disabled at this time
...
```

Case 5: Switching system clock from IHRC/2 to IHRC/4

```
...                             //  system clock is IHRC/2, ILRC is enabled here
CLKMD       =   0X14 ;          //  switch to IHRC/4
...
```

Case 6: System may hang if it is to switch clock and turn off original oscillator at the same time

```
...                             //  system clock is ILRC
CLKMD       =   0x30 ;          //  CAN NOT switch clock from ILRC to IHRC/2 and
                                        turn off ILRC oscillator at the same time
```

5.5. 16-bit Timer (Timer16)

PMS154G provide a 16-bit hardware timer (Timer16/T16) and its clock source may come from system clock (CLK), internal high RC oscillator (IHRC), internal low RC oscillator (ILRC), external crystal oscillator (EOSC), PA0 or PA4. Before sending clock to the 16-bit counter, a pre-scaling logic with divided-by-1, 4, 16 or 64 is selectable for wide range counting. The 16-bit counter performs up-counting operation only, the counter initial values can be stored from data memory by issuing the ***stt16*** instruction and the counting values can be loaded to data memory by issuing the ***ldt16*** instruction. The interrupt request from Timer16 will be triggered by the selected bit which comes from bit[15:8] of this 16-bit counter, rising edge or falling edge can be optional chosen by register ***integs.4***. The hardware diagram of Timer16 is shown as Fig. 4.

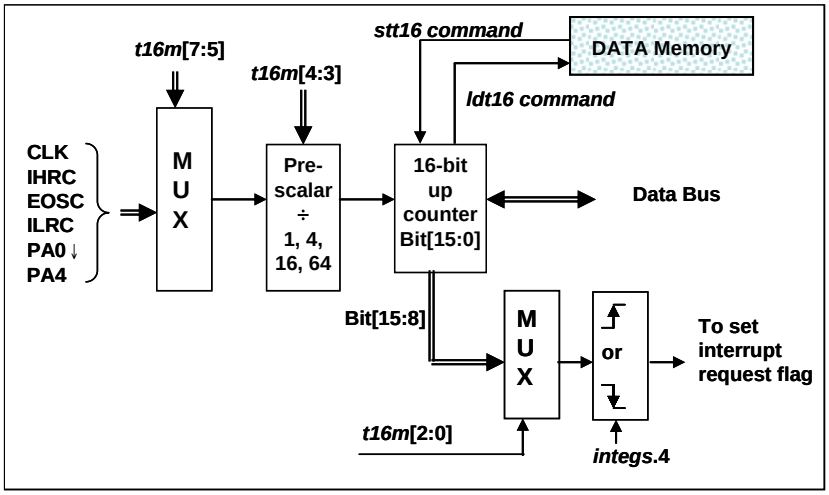


Fig. 4: Hardware diagram of Timer16

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, EOSC, ILRC, PA0_F      // 1st par.
$ 4~3:   /1, /4, /16, /64                                     // 2nd par.
$ 2~0:   BIT8, BIT9, BIT10, BIT11, BIT12, BIT13, BIT14, BIT15 // 3rd par.
```

User can 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
```

5.6. Watchdog Timer

The watchdog timer (WDT) is a counter with clock coming from ILRC. WDT can be cleared by power-on-reset or by command ***wdreset*** at any time. There are four different timeout periods of watchdog timer can be chosen by setting the ***misc*** register, it is:

- 8k ILRC clock period when misc[1:0]=00 (default)
- 16k ILRC clock period when misc[1:0]=01
- 64k ILRC clock period when misc[1:0]=10
- 256k ILRC clock period when 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, PMS154G will be reset to restart the program execution. The relative timing diagram of watchdog timer is shown as Fig. 5.

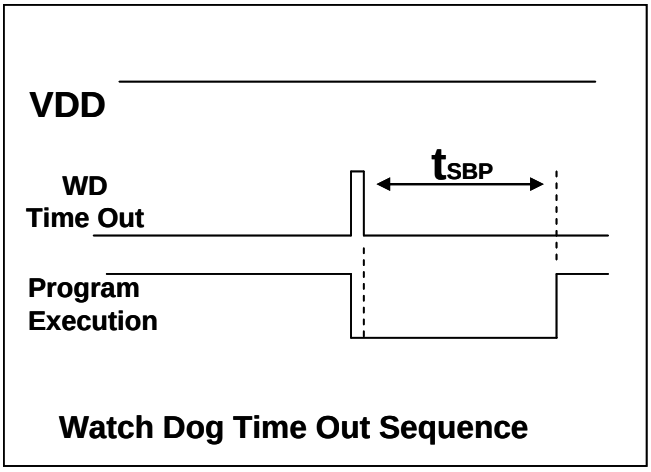


Fig. 5: Sequence of Watch Dog Time Out

5.7. Interrupt Controller

The hardware diagram of interrupt controller is shown as Fig. 6. There are total 7 interrupt sources for PMS154G: PA0, PB0, Timer16, Comparator, Timer2, Timer3, PWMG0. Among them, every interrupt request line to CPU has its own corresponding interrupt control bit to enable or disable it. 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.

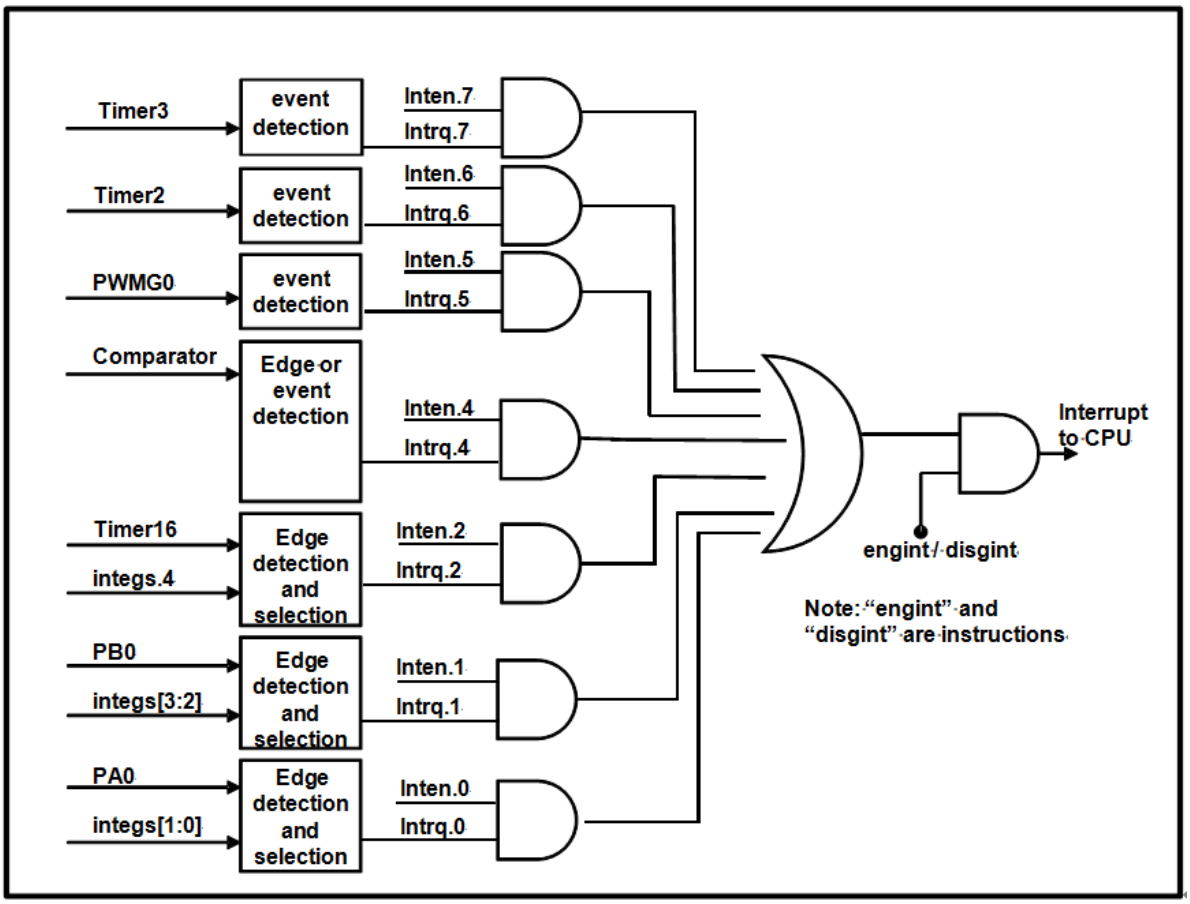


Fig. 6: Hardware diagram of Interrupt controller

The stack memory for interrupt is shared with data memory and its address is specified by stack register ***sp.*** Since the program counter is 16 bits width, the bit 0 of stack register ***sp*** should be kept 0. Moreover, user can use ***pushaf*** / ***popaf*** instructions to store or restore the values of ***ACC*** and ***flag*** register ***to*** / ***from*** stack memory. Since the stack memory is shared with data memory, the stack position and level are arranged by the compiler in Mini-C project. When defining the stack level in ASM project, users should arrange their locations carefully to prevent address conflicts.

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. And so on, two bytes stack memory is for ***pushaf***. For interrupt operation, the following sample program shows how to handle the interrupt, noticing that it needs four bytes stack memory to handle interrupt and ***pushaf***.

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

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

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

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

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

5.8. Power-Save and Power-Down

There are three operational modes defined by hardware: ON mode, Power-Save mode and Power-Down modes. ON mode is the state of normal operation with all functions ON, Power-Save mode ("***stopexe***") is the state to reduce operating current and CPU keeps ready to continue, Power-Down mode ("***stopsys***") is used to save power deeply. Therefore, Power-Save mode is used in the system which needs low operating power with wake-up occasionally and Power-Down mode is used in the system which needs power down deeply with seldom wake-up. Table 5 shows the differences in oscillator modules between Power-Save mode ("***stopexe***") and Power-Down mode ("***stopsys***").

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

Table 5: Differences in oscillator modules between STOPSYS and STOPEXE

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

Using "***stopexe***" instruction to enter the Power-Save mode, only system clock is disabled, remaining all the oscillator modules be active. For CPU, it stops executing; however, for Timer16, counter keep counting if its clock source is not the system clock. The wake-up sources for "***stopexe***" can be IO-toggle or Timer16 counts to set values when the clock source of Timer16 is IHRC or ILRC modules,or wakeup by comparator when setting GPCC.7=1 and GPCS.6=1 to enable the comparator wakeup 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 shown below:

- IHRC and EOSC oscillator modules: No change, keep active if it was enabled
- ILRC oscillator modules: must remain enabled, need to start with ILRC when be wakening up
- System clock: Disable, therefore, CPU stops execution
- OTP memory is turned off
- Timer counter: Stop counting if the system clock is selected or the corresponding oscillator module is disabled; otherwise, it keeps counting. (The Timer contains TM16, TM2, TM3, PWMG0, PWMG1, and PWMG2.)
- Wake-up sources:
  a. IO toggle wake-up: IO toggling in digital input mode (*PxC* bit is 0 and *PxDIER* bit is 1).
  b. Timer wake-up: If the clock source of Timer is not the SYSCLK, the system will be wake-up when the Timer counter reaches the set value.
  c. 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

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

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

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

5.8.2. Power-Down mode ("*stopsys*")

Power-Down mode is the state of deeply power-saving with turning off all the oscillator modules. By using the "*stopsys*" instruction, this chip will be put on Power-Down mode directly. It is recommending to set GPCC.7=0 to disable the comparator before the command "stopsys". The internal low frequency RC oscillator must be enabled before entering the Power-Down mode, means that bit 2 of register ***clkmd*** (0x03) must be set to high before issuing "***stopsys***" command in order to resume the system when wakeup. The following shows the internal status of PMS154G 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: IO toggle in digital mode (PxDIER bit is 1)

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

```
CMKMD   =   0xF4;     //   Change clock from IHRC to ILRC, disable watchdog timer
CLKMD.4 =   0;        //   disable IHRC
...
while (1)
{
        STOPSYS;      //   enter power-down
        if (…) break; //   if wakeup happen and check OK, then return to high speed,
                      //   else stay in power-down mode again.
}
CLKMD   =   0x34;     //   Change clock from ILRC to IHRC/2
```

5.8.3. Wake-up

After entering the Power-Down or Power-Save modes, the PMS154G can be resumed to normal operation by toggling IO pins, Wake-up from timer is available for Power-Save mode ONLY. Table 6 shows the differences in wake-up sources between STOPSYS and STOPEXE.

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

Table 6: Differences in wake-up sources between Power-Save mode and Power-Down mode

When using the IO pins to wake-up the PMS154G, registers ***pxdier*** should be properly set to enable the wake-up function for every corresponding pin. The time for normal wake-up is about 3,000 ILRC clocks counting from wake-up event; fast wake-up can be selected to reduce the wake-up time by ***misc*** register, and the time for fast wake-up is about 45 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	45 * T_{ILRC}, Where T_{ILRC} is the time period of ILRC
STOPEXE suspend or STOPSYS suspend	Normal wake-up	3000 * T_{ILRC}, Where T_{ILRC} is the clock period of ILRC

Table 7: Wake-up time of toggling IO pin

Please notice that when Code Option is set to Fast boot-up, 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***.

5.9. IO Pins

Other than PA5, all the pins can be independently set into two states output or input by configuring the data registers (***pa/pb***), control registers (***pac/pbc***) and pull-high registers (***paph/pbph***). All these pins have Schmitt-trigger input buffer and output driver with CMOS level. When it is set to output low, the pull-high resistor is turned off automatically. If user wants to read the pin state, please notice that it should be set to input mode before reading the data port; if user reads the data port when it is set to output mode, the reading data comes from data register, NOT from IO pad. As an example, Table 7 shows the configuration table of bit 0 of port A. The hardware diagram of IO buffer is also shown as Fig. 8.

pa.0	***pac.0***	***paph.0***	**Description**
X	0	0	Input without pull-high resistor
X	0	1	Input with pull-high resistor
0	1	X	Output low without pull-high resistor(pull-high resistor turns off automatically)
1	1	0	Output high without pull-high resistor
1	1	1	Output high with pull-high resistor

Table 8: PA0 Configuration Table

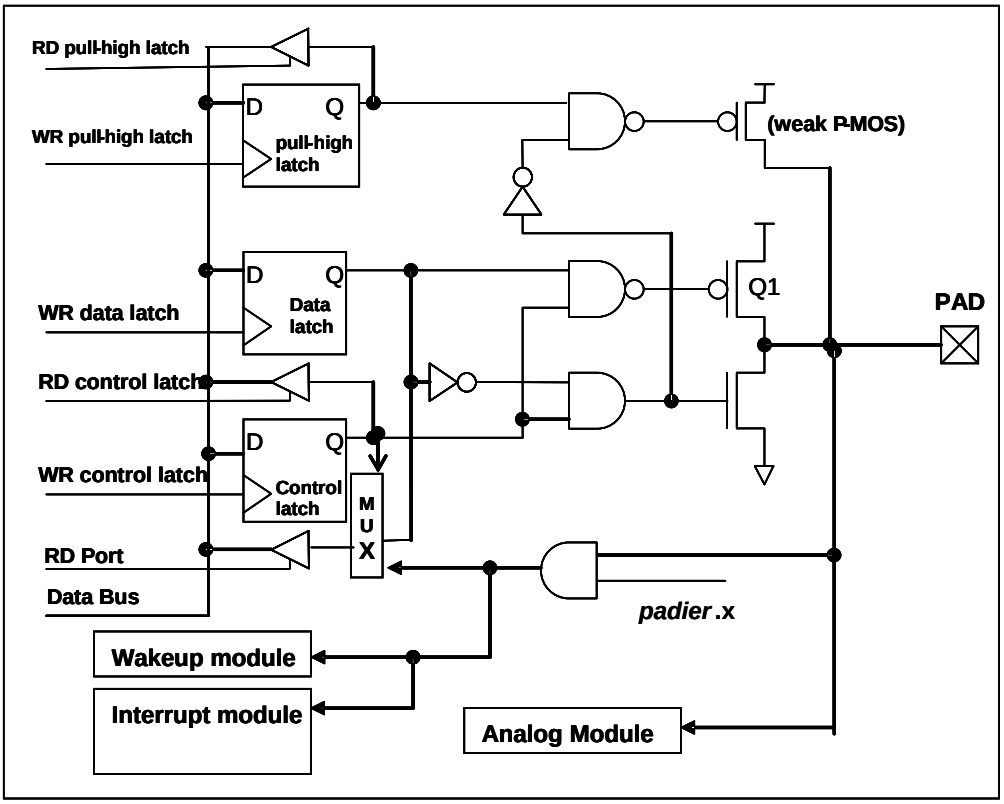


Fig. 7: Hardware diagram of IO buffer

Most IOs can be adjusted their Driving or Sinking current capability to Normal or Low by code option ***Drive.***

Other than PA5, all the IO pins have the same structure; PA5 can be open-drain ONLY when setting to output mode (without Q1). When PMS154G 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 ***pxdier*** to high. The same reason, ***padier.0*** should be set to high when PA0 is used as external interrupt pin.

5.10. Reset

5.10.1. Reset

There are many causes to reset the PMS154G, once reset is asserted, all the registers in PMS154G 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.

5.10.2. LVR reset

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

5.11. VDD/2 Bias Voltage Generator

This function can be enabled by ***misc.4*** and code option ***LCD2***. To use this function, user must select ***PB0_A034*** for ***LCD2*** and set ***misc.4*** to 1 in the program. Those pins which are defined to output VDD/2 voltage are PB0, PA0, PA4 and PA3 during input mode, being used as COM function for LCD application. If user wants to output V_{DD}, VDD/2, GND three levels voltage, the corresponding pins must be set to output-high for V_{DD}, enabling VDD/2 bias voltage with input mode for VDD/2, and output-low for GND correspondingly, Fig.8 shows how to use this function.

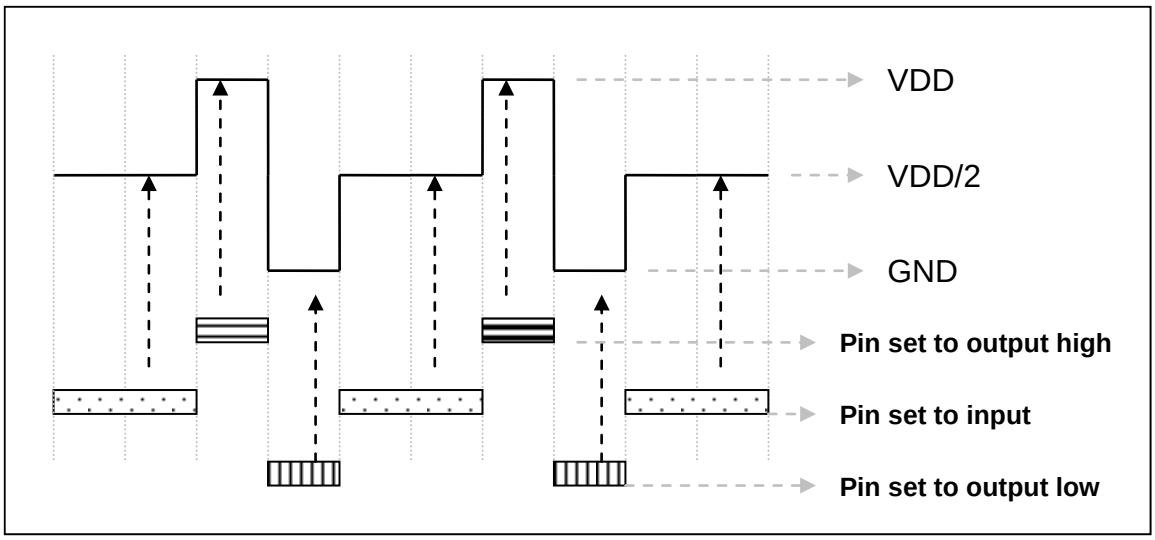


Fig. 8: Using VDD/2 bias voltage generator

5.12. Comparator

One hardware comparator is built inside the PMS154G; Fig. 9 shows its hardware diagram. It can compare signals between two pins or with either internal reference voltage $V_{internal\ R}$ or internal bandgap reference voltage. The two signals to be compared, one is the plus input and the other one is the minus input. For the minus input of comparator can be PA3, PA4, Internal bandgap 1.20V, PB6, PB7 or $V_{internal\ R}$ selected by bit [3:1] of ***gpcc*** register, and the plus input of comparator can be PA4 or $V_{internal\ R}$ selected by bit 0 of gpcc register.

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

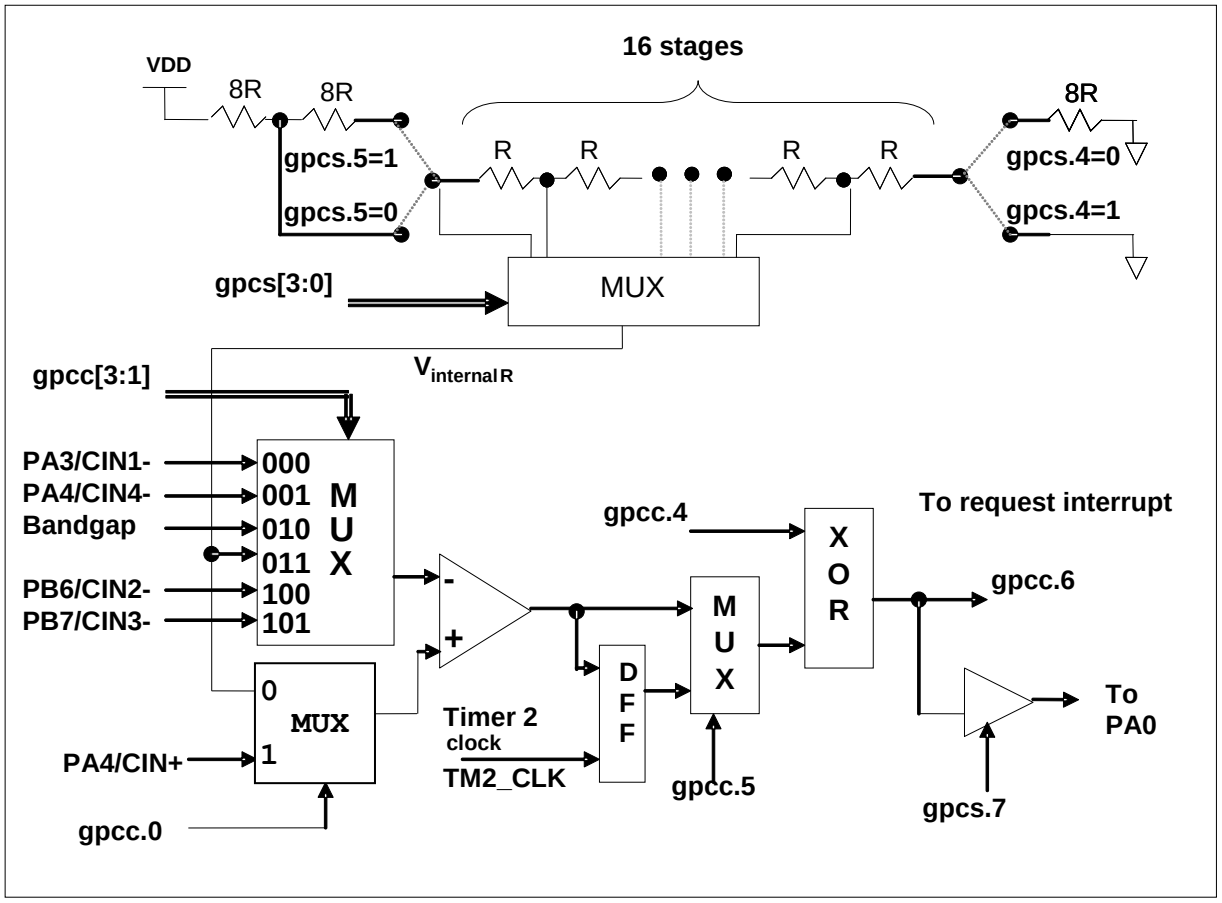


Fig. 9: Hardware diagram of comparator

5.12.1 Internal reference voltage ($V_{internal R}$)

The internal reference voltage $V_{internal R}$ is built by series resistance to provide different level of reference voltage, bit 4 and bit 5 of ***gpcs*** register are used to select the maximum and minimum values of $V_{internal R}$ and bit [3:0] of ***gpcs*** register are used to select one of the voltage level which is deivided-by-16 from the defined maximum level to minimum level. Fig. 10 to Fig. 13 shows four conditions to have different reference voltage $V_{internal R}$. By setting the ***gpcs*** register, the internal reference voltage $V_{internal R}$ can be ranged from (1/32)*V_{DD} to (3/4)*V_{DD}.

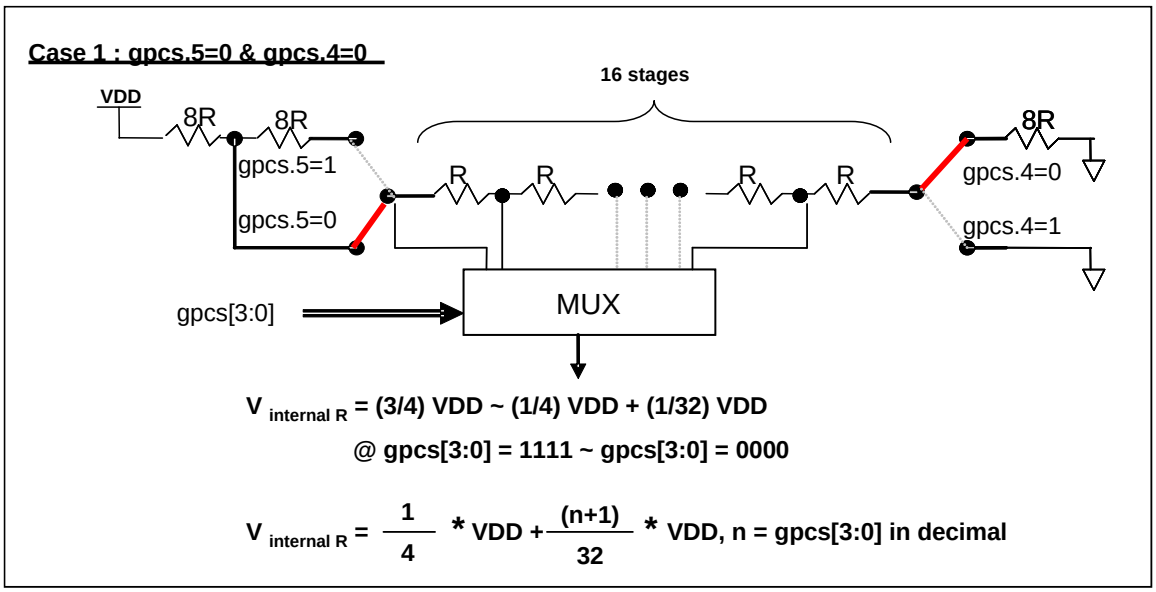


Fig. 10: $V_{internal\ R}$ hardware connection if gpcs.5=0 and gpcs.4=0

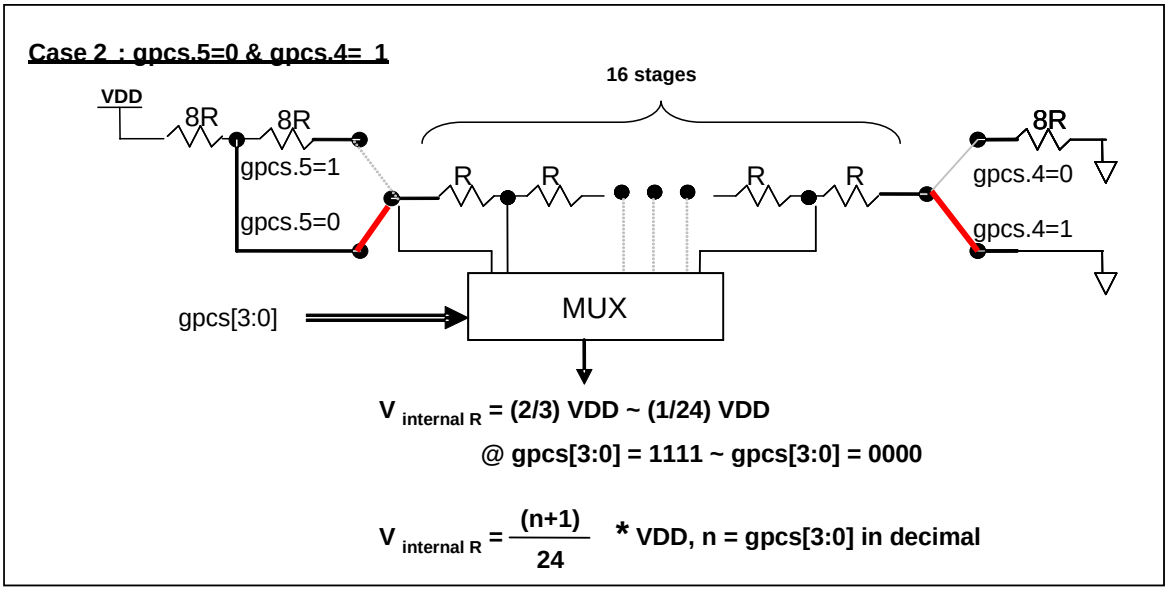


Fig. 11: $V_{internal\ R}$ hardware connection if gpcs.5=0 and gpcs.4=1

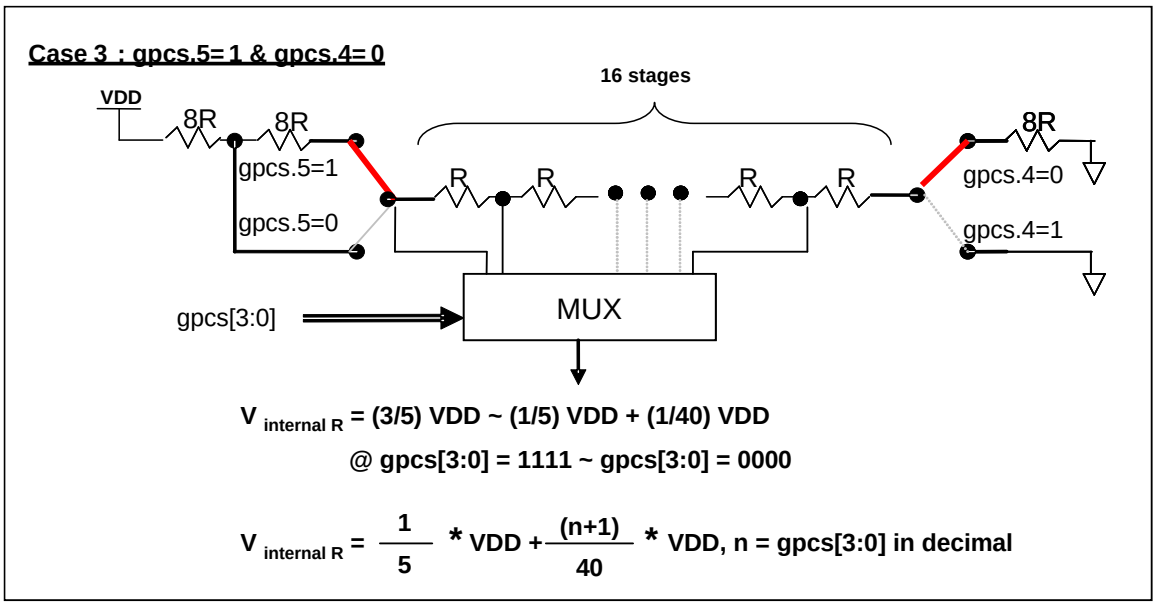


Fig. 12: $V_{internal\ R}$ hardware connection if gpcs.5=1 and gpcs.4=0

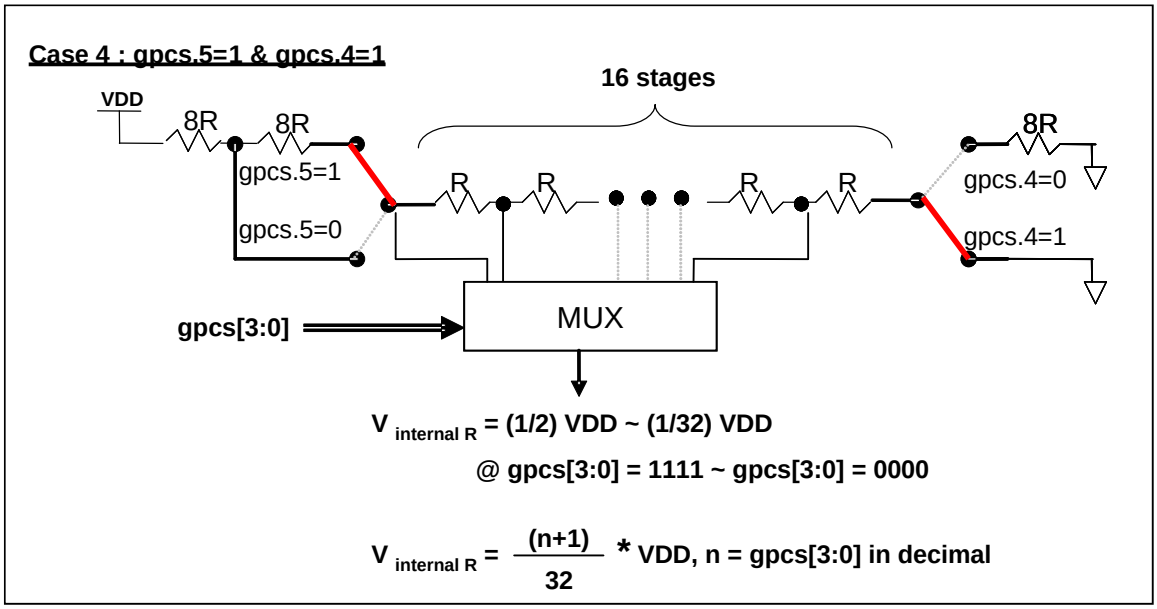


Fig. 13: $V_{internal\ R}$ hardware connection if gpcs.5=1 and gpcs.4=1

5.12.2. 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    = 0b0_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_10_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  = 0bxxxx_0_xxx;       // 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;
```

Note: When selecting output to PA0 output, GPCS will affect the PA3 output function in ICE. Though the IC is fine, be careful to avoid this error during emulation.

5.12.3. Using the comparator and bandgap 1.20V

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

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

Case 1:

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

5.13. 8-bit Timer with PWM generation (Timer2, Timer3)

Two 8-bit hardware timers (Timer2/TM2, Timer3/TM3) with PWM generation are implemented in the PMS154G, Timer2 is used as the example to describe its function due to these two 8-bit timers are the same. Please refer to Fig. 14 shown its hardware diagram, the clock sources of Timer2 may come from system clock, internal high RC oscillator (IHRC) or, internal low RC oscillator (ILRC), external crystal oscillator (EOSC), PA0, PA4, PB0 or comparator. Bit[7:4] of register tm2c are used to select the clock source of Timer2. Please notice that if IHRC is selected for Timer2 clock source, the clock sent to Timer2 will keep running when using ICE in halt state. According to the setting of register tm2c[3:2], Timer2 output can be selectively output to PB2, PA3 or PB4(Timer3 count output can be selected as PB5, PB6 or PB7). At this point, regardless of whether PX.x is the input or output state, Timer2(or Timer3) signal will be forced to output. A clock pre-scaling module is provided with divided-by-1, 4, 16, and 64 options, controlled by bit [6:5] of tm2s register; one scaling module with divided-by-1~32 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 or duty of PWM. There are two operating modes for Timer2: period mode and PWM mode; period mode is used to generate periodical output waveform or interrupt event; PWM mode is used to generate PWM output waveform with optional 6-bit or 8-bit PWM resolution, Fig. 15 shows the timing diagram of Timer2 for both period mode and PWM mode.

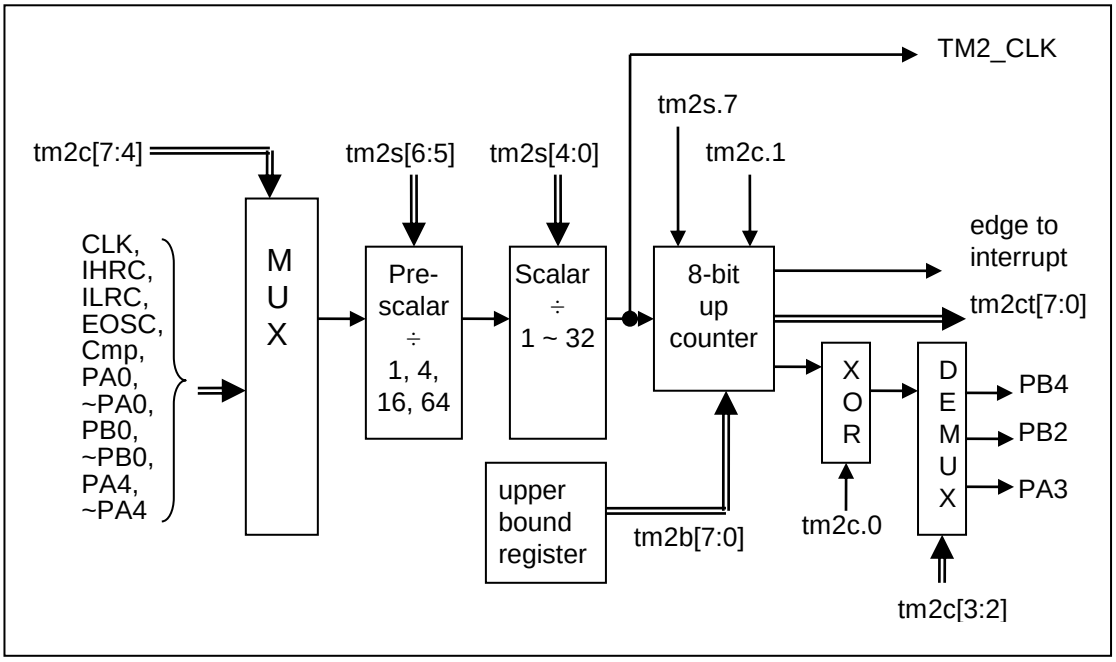
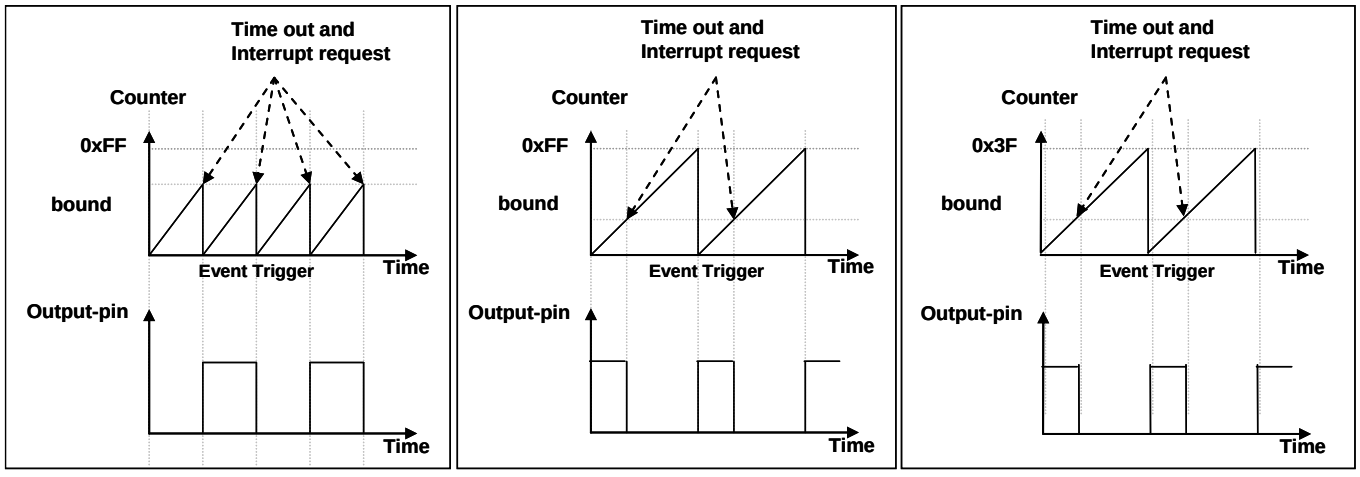


Fig. 14: Timer2 hardware diagram



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

Fig. 15: Timing diagram of Timer2 in period mode and PWM mode (tm2c.1=1)

5.13.1. Using the Timer2 to generate periodical waveform

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

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

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

Example 1:

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

Example 2:

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

Example 3:

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

Example 4:

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

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

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

5.13.2. Using the Timer2 to generate 8-bit PWM waveform

If 8-bit PWM mode is selected, it should set ***tm2c***[1]=1 and ***tm2s***[7]=0, the frequency and duty cycle of output waveform can be summarized as below:

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

Duty of Output = [(K＋1) ÷ 256]×100%

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

Example 1:

tm2c = 0b0001_1010, Y=8MHz
tm2b = 0b0111_1111, K=127
tm2s = 0b0_00_00000, S1=1, S2=0
➔ frequency of output = 8MHz ÷ (256 × 1 × (0+1)) = 31.25KHz
➔ duty of output = [(127+1) ÷ 256] × 100% = 50%

Example 2:

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

Example 3:

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

Example 4:

tm2c = 0b0001_1010, Y=8MHz
tm2b = 0b0000_1001, K = 9
tm2s = 0b0_00_00000, S1=1, S2=0
➔ frequency of output = 8MHz ÷ (256 × 1 × (0+1)) = 31.25KHz
➔ duty of output = [(9+1) ÷ 256] × 100% = 3.9%

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

```
void    FPPA0 (void)
{
    .ADJUST_IC    SYSCLK=IHRC/2, IHRC=16MHz, VDD=5V
    wdreset;
    tm2ct = 0x00;
    tm2b = 0x7f;
    tm2s = 0b0_00_00001;        //    8-bit PWM, pre-scalar = 1, scalar = 2
    tm2c = 0b0001_10_1_0;       //    system clock, output=PA3, PWM mode
    while(1)
    {
        nop;
    }
}
```

5.13.3. Using the Timer2 to generate 6-bit PWM waveform

If 6-bit PWM mode is selected, it should set ***tm2c***[1]=1 and ***tm2s***[7]=1, the frequency and duty cycle of output waveform can be summarized as below:

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

Duty of Output = [(K＋1) ÷ 64] × 100%

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

Example 1:

tm2c = 0b0001_1010, Y=8MHz
tm2b = 0b0001_1111, K=31
tm2s = 0b1_00_00000, S1=1, S2=0
➔ frequency of output = 8MHz ÷ (64 × 1 × (0+1)) = 125KHz
➔ duty = [(31+1) ÷ 64] × 100% = 50%

Example 2:

tm2c = 0b0001_1010, Y=8MHz
tm2b = 0b0001_1111, K=31
tm2s = 0b1_11_11111, S1=64, S2=31
➔ frequency of output = 8MHz ÷ (64 × 64 × (31+1)) = 61.03Hz
➔ duty of output = [(31+1) ÷ 64] × 100% = 50%

Example 3:

tm2c = 0b0001_1010, Y=8MHz
tm2b = 0b0011_1111, K=63
tm2s = 0b1_00_00000, S1=1, S2=0
➔ frequency of output = 8MHz ÷ (64 × 1 × (0+1)) = 125KHz
➔ duty of output = [(63+1) ÷ 64] × 100% = 100%

Example 4:

tm2c = 0b0001_1010, Y=8MHz
tm2b = 0b0000_0000, K=0
tm2s = 0b1_00_00000, S1=1, S2=0
➔ Frequency = 8MHz ÷ (64 × 1 × (0+1)) = 125KHz
➔ Duty = [(0+1) ÷ 64] × 100% =1.5%

5.14. 11-bit PWM generation

Three 11-bit hardware PWM generators (PWMG0, PWMG1 & PWMG2) are implemented in the PMS154G. PWMG0 is used as the example to describe its functions due to all of them are almost the same.

Their individual outputs are listed as below:

- PWMG0 – PA0, PB4, PB5
- PWMG1 – PA4, PB6, PB7

  PWMG2 – PA3, PB2, PB3, PA5 (Note: PA5 open drain output only, users need to enable the internal pull-high resistor or add an external pull-high resistor when using, and ICE does not support PA5 PWM function.)

5.14.1. PWM Waveform

A PWM output waveform (Fig. 16) has a time-base (T_{Period} = Time of Period) and a time with output high level (Duty Cycle). The frequency of the PWM output is the inverse of the period ($f_{PWM} = 1/T_{Period}$), the resolution of the PWM is the clock count numbers for one period (N bits resolution, $2^N \times T_{clock} = T_{Period}$).

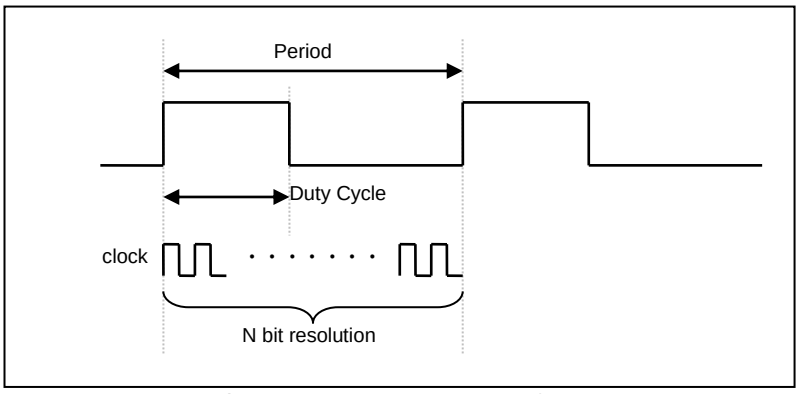


Fig. 16: PWM Output Waveform

5.14.2. Hardware and Timing Diagram

Fig. 17 shows the hardware diagram of 11-bit Timer. The clock source can be IHRC or system clock. Depending on the setting of register PWMC, PWM can be optionally output to PA0, PB4 or PB5. At this point, PWM signal will be forced to output regardless of whether PX.x is the input or output state. The period of PWM waveform is defined in the PWM upper bond high and low registers, the duty cycle of PWM waveform is defined in the PWM duty high and low registers.

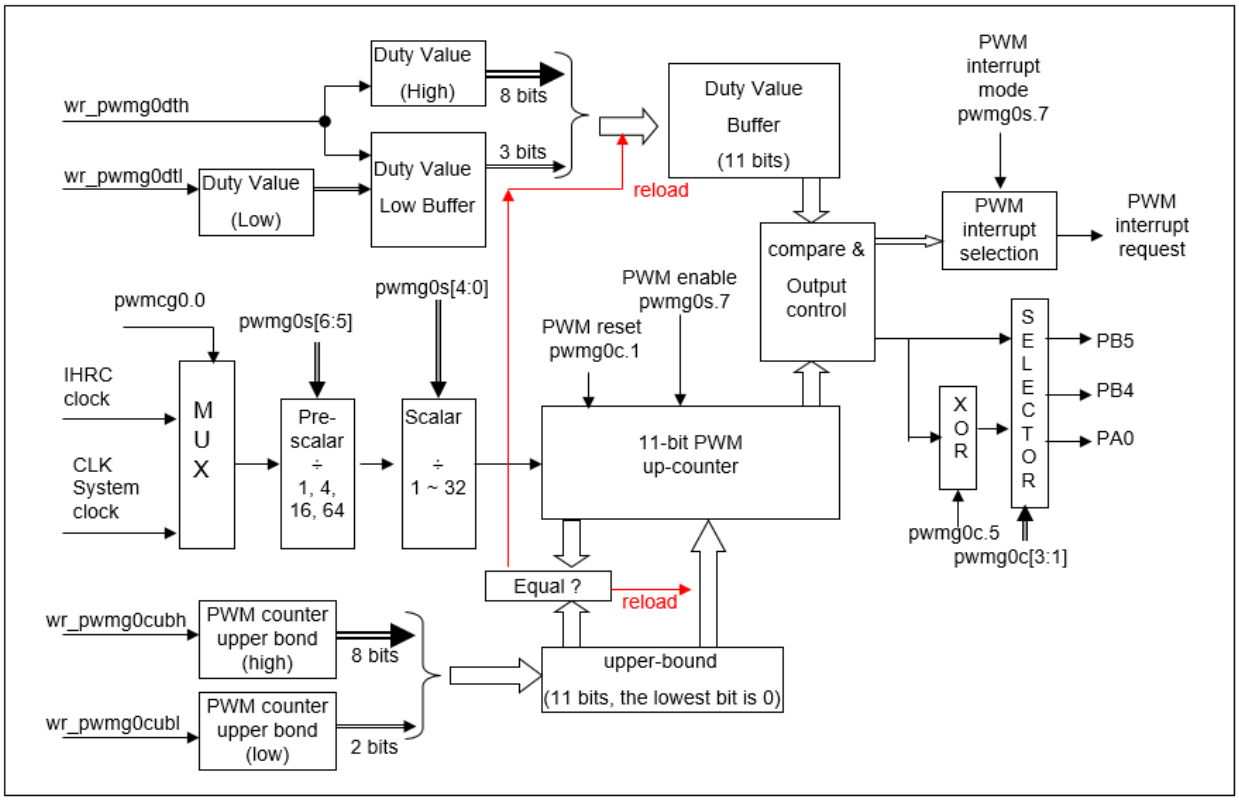


Fig. 17: Hardware Diagram of 11-bit PWM Generator 0 (PWMG0)

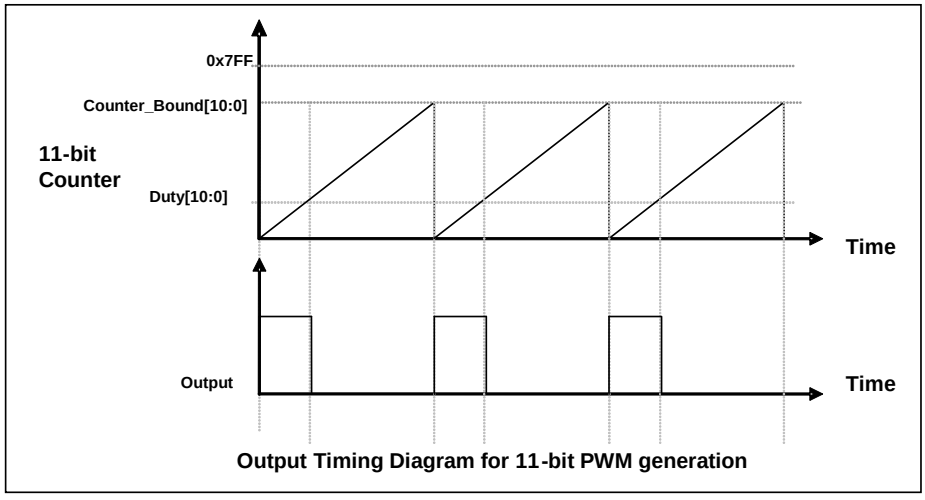


Fig. 18: Output Timing Diagram of 11-bit PWM Generator

5.14.3. Equations for 11-bit PWM Generator

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

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

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

Where, **P** = *PWMGxS* [6:5] : pre-scalar (**P** = 1, 4, 16, 64)

K = *PWMGxS* [4:0] : scalar in decimal (**K** =0 ~ 31)

DB10_1 = Duty_Bound[10:1] = {*PWMGxDTH*[7:0], *PWMGxDTL*[7:6]}, duty bound

DB0 = Duty_Bound[0] = *PWMGxDTL*[5]

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

5.14.4. Complementary PWM with Dead Zones

Users can use two 11bit PWM generators to output two complementary PWM waveforms with dead zones. Take PWMG0 output PWM0, PWMG1 output PWM1 as an example, the program reference is as follows.

In addition, Timer2 and Timer3 can also output 8-bit PWM waveforms with complementary dead zones of two bands. The principle is similar to this, and it will not be described in detail.

```
#define  dead_zone_R  2          //   Control dead-time before rising edge of PWM1
#define  dead_zone_F  3          //   Control dead-time after falling edge of PWM1

void     FPPA0 (void)
{
  .ADJUST_IC     SYSCLK=IHRC/16, IHRC=16MHz, VDD=5V;
  //.....
  Byte   duty     =   60;         //   Represents the duty cycle of PWM0
  Byte   _duty    =   100 - duty; //   Represents the duty cycle of PWM1

  //************** Set the counter upper bound and duty cycle **************
  PWMG0DTL     =   0x00;
  PWMG0DTH     =   _duty;
  PWMG0CUBL    =   0x00;
  PWMG0CUBH    =   100;

  PWMG1DTL     =   0x00;
  PWMG1DTH     =   _duty - dead_zone_F;
  // Use duty cycle to adjust the dead-time after the falling edge of PWM1
  PWMG1CUBL    =   0x00;
  PWMG1CUBH    =   100;
  // The above values are assigned before enable PWM output
```

```
//******************** PWM out control **********************
$ PWMG0C Enable,Inverse,PA0,SYSCLK;          //PWMG0 output the PWM0 waveform to PA0
$ PWMG0S INTR_AT_DUTY,/1,/1;

.delay dead_zone_R;  // Use delay to adjust the dead-time before the rising edge of PWM1

$ PWMG1C Enable, PA4, SYSCLK;      //PWMG1 output the PWM1 waveform to PA4
$ PWMG1S INTR_AT_DUTY, /1, /1;

//***** Note: for the output control part of the program, the code sequence can not be moved ******//

While(1)
      {  nop;       }
}
```

The PWM0 / PWM1 waveform obtained by the above program is shown in Fig. 19.

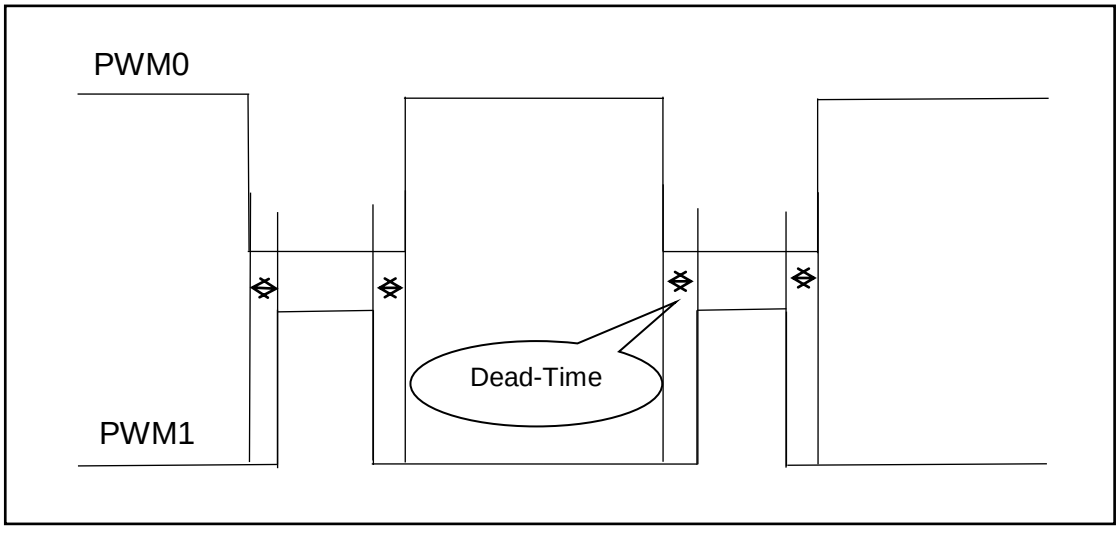


Fig. 19: Two complementary PWM waveforms with dead zones

Users can modify the **dead_zone_R** and **dead_zone_F** values in the program to adjust the dead-time. Table 9 provides data corresponding to different dead-time for users' reference. Where, if dead-time = 4us, then there are dead zones of 4us before and after PWM1 high level.

dead-time (us)	dead_zone_R	dead_zone_F
4 (minimum)	0	2
6	2	3
8	4	4
10	6	5
12	8	6
14	10	7

Table 9: The value of dead-time for reference

Dead_zone_R and **dead_zone_F** need to work together to get an ideal dead-time. If user wants to adjust other dead-time, please note that **dead_zone_R** and **dead_zone_F** need to meet the following criteria:

dead_zone_R	dead_zone_F
1 / 2 / 3	> 1
4 / 5 / 6 / 7	> 2
8 / 9	> 3
...	...

6. IO Registers

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

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

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

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

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

Bit	Reset	R/W	Description	
7 – 5	111	R/W	System clock selection	
			Type 0, clkmd[3]=0	Type 1, clkmd[3]=1
			000: IHRC/4 001: IHRC/2 010: reserved 011: EOSC/4 100: EOSC/2 101: EOSC 110: ILRC/4 111: ILRC (default)	000: IHRC/16 001: IHRC/8 010: ILRC/16 (ICE does NOT Support.) 011: IHRC/32 100: IHRC/64 101: EOSC/8 Others: reserved
4	1	R/W	IHRC oscillator Enable. 0 / 1: disable / enable	
3	0	R/W	Clock Type Select. This bit is used to select the clock type in bit [7:5]. 0 / 1: Type 0 / Type 1	
2	1	R/W	ILRC Enable. 0 / 1: disable / enable If ILRC is disabled, watchdog timer is also disabled.	
1	1	R/W	Watch Dog Enable. 0 / 1: disable / enable	
0	0	R/W	Pin PA5/PRSTB function. 0 / 1: PA5 / PRSTB.	

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

Bit	Reset	R/W	Description
7	-	R/W	Enable interrupt from Timer3. 0 / 1: disable / enable.
6	-	R/W	Enable interrupt from Timer2. 0 / 1: disable / enable.
5	-	R/W	Enable interrupt from PWMG0. 0 / 1: disable / enable.
4	-	R/W	Enable interrupt from comparator. 0 / 1: disable / enable.
3	-	R/W	Reserved.
2	-	R/W	Enable interrupt from Timer16 overflow. 0 / 1: disable / enable.
1	-	R/W	Enable interrupt from PB0. 0 / 1: disable / enable.
0	-	R/W	Enable interrupt from PA0. 0 / 1: disable / enable.

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

Bit	Reset	R/W	Description
7	-	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 PWMG0, 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	-	R/W	Interrupt Request from pin PB0, this bit is set by hardware and cleared by software. 0 / 1: No request / Request
0	-	R/W	Interrupt Request from pin PA0, this bit is set by hardware and cleared by software. 0 / 1: No request / Request

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

Bit	Reset	R/W	Description
7 – 5	000	R/W	Timer Clock source selection 000: Timer 16 is disabled 001: CLK (system clock) 010: reserved 011: PA4 falling edge (from external pin) 100: IHRC 101: EOSC 110: ILRC 111: PA0 falling edge (from external pin)
4 – 3	00	R/W	Internal clock divider. 00: /1 01: /4 10: /16 11: /64
2 – 0	000	R/W	Interrupt source selection. Interrupt event happens when selected bit is changed. 0 : bit 8 of Timer16 1 : bit 9 of Timer16 2 : bit 10 of Timer16 3 : bit 11 of Timer16 4 : bit 12 of Timer16 5 : bit 13 of Timer16 6 : bit 14 of Timer16 7 : bit 15 of Timer16

6.7. External Oscillator setting Register (*eoscr, write only*), IO address = 0x0a

Bit	Reset	R/W	Description
7	0	WO	Enable external crystal oscillator. 0 / 1 : Disable / Enable
6 – 5	00	WO	External crystal oscillator selection. 00 : reserved 01 : Low driving capability, for lower frequency, ex: 32KHz crystal oscillator 10 : Middle driving capability, for middle frequency, ex: 1MHz crystal oscillator 11 : High driving capability, for higher frequency, ex: 4MHz crystal oscillator
4 – 1	-	-	Reserved. Please keep 0 for future compatibility.
0	0	WO	Power-down the Bandgap and LVR hardware modules. 0 / 1: normal / power-down. Note: If bandgap be disabled, there will only ILRC/T16/TM2/TM3 and I/O function can be used.

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

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

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

Bit	Reset	R/W	Description
7 – 3	11111	WO	Enable PA7~PA3 digital input and wake up event. 1 / 0 : enable / disable. When this bit is “0”, the function is disable to wake up from PA7~PA3.
2 – 1	-	-	Reserved. (keep 00 for future compatibility)
0	1	WO	Enable PA0 digital input, wake up event and interrupt request. 1 / 0 : enable / disable. When this bit is “0”, the function is disable wake up from PA0 and interrupt request from this pin.

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

Bit	Reset	R/W	Description
7 – 1	0xFF	WO	Enable PB7~PB1 digital input and wake up event. 1 / 0 : enable / disable. When this bit is “0”, the function is disable wake up from PB7~PB1.
0		WO	Enable PB0 digital input, wake up event and interrupt request. 1 / 0 : enable / disable. When this bit is “0”, the function is disable wake up from PB0 and interrupt request from this pin.

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

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

6.12. Port A Control Registers (*pac*), IO address = 0x11

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

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

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

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

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

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

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

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

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

6.17. MISC Register (*misc*), IO address = 0x08

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 3000 ILRC clocks (Not for fast boot-up) 1: Fast wake up. The wake-up time is 45 ILRC clocks.

Bit	Reset	R/W	Description
4	0	WO	Enable VDD/2 bias voltage generator 0 / 1 : Disable / Enable (ICE cannot be dynamically switched) If Code Option selects LCD output, but MISC.4 does not set to 1, then the VDD/2 bias cannot be output on the IC. However, the emulator is always OK. Two above phenomena are different.
3	-	-	Reserved.
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

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

Bit	Reset	R/W	Description
7 – 4	0000	R/W	Timer2 clock selection. 0000 : disable 0001 : CLK 0010 : IHRC 0011 : EOSC 0100 : ILRC 0101 : comparator output 1000 : PA0 (rising edge) 1001 : ~PA0 (falling edge) 1010 : PB0 (rising edge) 1011 : ~PB0 (falling edge) 1100 : PA4 (rising edge) 1101 : ~PA4 (falling edge) Others: reserved **Notice:** In ICE mode and IHRC is selected for Timer2 clock, the clock sent to Timer2 does NOT be stopped, Timer2 will keep counting when ICE is in halt state.
3 – 2	00	R/W	Timer2 output selection. 00 : disable 01 : PB2 10 : PA3 11 : PB4
1	0	R/W	Timer2 mode selection. 0 / 1 : period mode / PWM mode
0	0	R/W	Enable to inverse the polarity of Timer2 output. 0 / 1: disable / enable

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

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

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

Bit	Reset	R/W	Description
7	0	WO	PWM resolution selection. 0 : 8-bit 1 : 6-bit
6 – 5	00	WO	Timer2 clock pre-scalar. 00 : ÷ 1 01 : ÷ 4 10 : ÷ 16 11 : ÷ 64
4 – 0	00000	WO	Timer2 clock scalar.

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

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

6.22. Timer3 Control Register (*tm3c*), IO address = 0x32

Bit	Reset	R/W	Description
7 – 4	0000	R/W	Timer3 clock selection. 0000 : disable 0001 : CLK 0010 : IHRC 0011 : EOSC 0100 : ILRC 0101 : comparator output 1000 : PA0 (rising edge) 1001 : ~PA0 (falling edge) 1010 : PB0 (rising edge) 1011 : ~PB0 (falling edge) 1100 : PA4 (rising edge) 1101 : ~PA4 (falling edge) Others: reserved **Notice:** In ICE mode and IHRC is selected for Timer3 clock, the clock sent to Timer3 does NOT be stopped, Timer3 will keep counting when ICE is in halt state.
3 – 2	00	R/W	Timer3 output selection. 00 : disable 01 : PB5 10 : PB6 11 : PB7

Bit	Reset	R/W	Description
1	0	R/W	Timer3 mode selection. 0 / 1 : period mode / PWM mode
0	0	R/W	Enable to inverse the polarity of Timer3 output. 0 / 1: disable / enable

6.23. Timer3 Counter Register (*tm3ct*), IO address = 0x33

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

6.24. Timer3 Scalar Register *(tm3s*), IO address = 0x34

Bit	Reset	R/W	Description
7	0	WO	PWM resolution selection. 0 : 8-bit 1 : 6-bit
6 – 5	00	WO	Timer3 clock pre-scalar. 00 : ÷ 1 01 : ÷ 4 10 : ÷ 16 11 : ÷ 64
4 – 0	00000	WO	Timer3 clock scalar.

6.25. Timer3 Bound Register (*tm3b*), IO address = 0x35

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

6.26. Comparator Control Register (*gpcc*), IO address = 0x18

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 input pins to be digital disable to prevent IO leakage.
6	-	RO	Comparator result 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.

Bit	Reset	R/W	Description
3 – 1	000	R/W	Selection the minus input (-) of comparator. 000 : PA3 001 : PA4 010 : Internal 1.20 volt bandgap reference voltage (not suitable for the comparator wake-up function) 011 : $V_{internal\ R}$ 100 : PB6 101: PB7 11X: reserved
0	0	R/W	Selection the plus input (+) of comparator. 0 : $V_{internal\ R}$ 1 : PA4

6.27. Comparator Selection Register (*gpcs*), IO address = 0x19

Bit	Reset	R/W	Description
7	0	WO	Comparator output enable (to PA0). 0 / 1 : disable / enable (Please avoid this situation: GPCS will affect the PA3 output function when selecting output to PA0 output in ICE.)
6	-	WO	Wakeup by comparator 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)

6.28. PWMG0 control Register (*pwmg0c*), IO address = 0x20

Bit	Reset	R/W	Description
7	0	R/W	Enable PWMG0 generator. 0 / 1: disable / enable
6	-	RO	Output status of PWMG0 generator.
5	0	R/W	Enable to inverse the polarity of PWMG0 generator output. 0 / 1: disable / enable.
4	0	R/W	PWMG0 counter reset. Writing “1” to clear PWMG0 counter.
3 – 1	0	R/W	Select PWM output pin for PWMG0. 000: none 001: PB5 011: PA0 100: PB4 Others: reserved

Bit	Reset	R/W	Description
0	0	R/W	Clock source of PWMG0 generator. 0：SYSCLK，1：IHRC

6.29. PWMG0 Scalar Register (*pwmg0s*), IO address = 0x21

Bit	Reset	R/W	Description
7	0	WO	PWMG0 interrupt mode 0: Generate interrupt when counter matches the duty value 1: Generate interrupt when counter is 0
6 – 5	0	WO	PWMG0 clock pre-scalar 00 : ÷1 01 : ÷4 10 : ÷16 11 : ÷64
4 – 0	0	WO	PWMG0 clock divider

6.30. PWMG0 Counter Upper Bound High Register (*pwmg0cubh*), IO address = 0x24

Bit	Reset	R/W	Description
7 – 0	-	WO	Bit[10:3] of PWMG0 counter upper bound.

6.31. PWMG0 Counter Upper Bound Low Register (*pwmg0cubl*), IO address = 0x25

Bit	Reset	R/W	Description
7 – 6	-	WO	Bit[2:1] of PWMG0 counter upper bound.
5 – 0	-	-	Reserved

6.32. PWMG0 Duty Value High Register (*pwmg0dth*), IO address = 0x22

Bit	Reset	R/W	Description
7 – 0	-	WO	Duty values bit[10:3] of PWMG0.

6.33. PWMG0 Duty Value Low Register (*pwmg0dtl*), IO address = 0x23

Bit	Reset	R/W	Description
7 – 5	-	WO	Duty values bit [2:0] of PWMG0.
4 – 0	-	-	Reserved

Note: It's necessary to write PWMG0 Duty_Value Low Register before writing PWMG0 Duty_Value High Register.

6.34. PWMG1 control Register (*pwmg1c*), IO address = 0x26

Bit	Reset	R/W	Description
7	0	R/W	Enable PWMG1. 0 / 1 : disable / enable.
6	-	RO	Output of PWMG1.
5	0	R/W	Enable to inverse the polarity of PWMG1 output. 0 / 1 : disable / enable.
4	0	R/W	PWMG1 counter reset. Writing "1" to clear PWMG1 counter and this bit will be self clear to 0 after counter reset.

Bit	Reset	R/W	Description
3 – 1	0	R/W	Select PWMG1 output pin. 000: none 001: PB6 011: PA4 100: PB7 Others: reserved
0	0	R/W	Clock source of PWMG1. 0：SYSCLK，1：IHRC

6.35. PWMG1 Scalar Register (*pwmg1s*), IO address = 0x27

Bit	Reset	R/W	Description
7	0	WO	PWMG1 interrupt mode. 0: Generate interrupt when counter matches the duty value 1: Generate interrupt when counter is 0.
6 – 5	0	WO	PWMG1 clock pre-scalar. 00 : ÷1 01 : ÷4 10 : ÷16 11 : ÷64
4 – 0	0	WO	PWMG1 clock divider.

6.36. PWMG1 Counter Upper Bound High Register (*pwmg1cubh*), IO address = 0x2a

Bit	Reset	R/W	Description
7 – 0	0x00	WO	Bit[10:3] of PWMG1 counter upper bound.

6.37. PWMG1 Counter Upper Bound Low Register (*pwmg1cubl*), IO address = 0x2b

Bit	Reset	R/W	Description
7 – 6	000	WO	Bit[2:1] of PWMG1 counter upper bound.
5 – 0	-	-	Reserved

6.38. PWMG1 Duty Value High Register (*pwmg1dth*), IO address = 0x28

Bit	Reset	R/W	Description
7 – 0	0x00	WO	Duty values bit[10:3] of PWMG1.

6.39. PWMG1 Duty Value Low Register (*pwmg1dtl*), IO address = 0x29

Bit	Reset	R/W	Description
7 – 5	000	WO	Duty values bit[2:0] of PWMG1.
4 – 0	-	-	Reserved

Note: It's necessary to write PWMG1 Duty_Value Low Register before writing PWMG1 Duty_Value High Register.

6.40. PWMG2 control Register (*pwmg2c*), IO address = 0x2c

Bit	Reset	R/W	Description
7	0	R/W	Enable PWMG2. 0 / 1 : disable / enable.
6	-	RO	Output of PWMG2.
5	0	R/W	Enable to inverse the polarity of PWMG2 output. 0 / 1 : disable / enable.
4	0	R/W	PWMG2 counter reset. Writing “1” to clear PWMG2 counter and this bit will be self clear to 0 after counter reset.
3 – 1	0	R/W	Select PWMG2 output pin. 000: disable 001: PB3 011: PA3 100: PB2 101: PA5 (ICE does NOT Support.) Others: reserved
0	0	R/W	Clock source of PWMG2. 0：SYSCLK，1：IHRC

6.41. PWMG2 Scalar Register (*pwmg2s*), IO address = 0x2d

Bit	Reset	R/W	Description
7	0	WO	PWMG2 interrupt mode. 0: Generate interrupt when counter matches the duty value 1: Generate interrupt when counter is 0.
6 – 5	0	WO	PWMG2 clock pre-scalar. 00 : ÷1 01 : ÷4 10 : ÷16 11 : ÷64
4 – 0	0	WO	PWMG2 clock divider.

6.42. PWMG2 Counter Upper Bound High Register (*pwmg2cubh*), IO address = 0x30

Bit	Reset	R/W	Description
7 – 0	0x00	WO	Bit[10:3] of PWMG2 counter upper bound.

6.43. PWMG2 Counter Upper Bound Low Register (*pwmg2cubl*), IO address = 0x31

Bit	Reset	R/W	Description
7 – 6	000	WO	Bit[2:1] of PWMG2 counter upper bound.
5 – 0	-	-	Reserved

6.44. PWMG2 Duty Value High Register (*pwmg2dth*), IO address = 0x2e

Bit	Reset	R/W	Description
7 – 0	0x00	WO	Duty values bit[10:3] of PWMG2.

6.45. PWMG2 Duty Value Low Register (*pwmg2dtl*), IO address = 0x2f

Bit	Reset	R/W	Description
7 – 5	000	WO	Duty values bit[2:0] of PWMG2.
4 – 0	-	-	Reserved

Note: It’s necessary to write PWMG2 Duty_Value Low Register before writing PWMG2 Duty_Value High Register.

7. Instructions

Symbol	Description
ACC	Accumulator (Abbreviation of accumulator)
a	Accumulator (Symbol of accumulator in program)
sp	Stack pointer
flag	ACC status flag register
I	Immediate data
&	Logical AND
|	Logical OR
←	Movement
^	Exclusive logic OR
+	Add
–	Subtraction
~	NOT (logical complement, 1's complement)
∓	NEG (2's complement)
OV	Overflow (The operational result is out of range in signed 2's complement number system)
Z	Zero (If the result of ALU operation is zero, this bit is set to 1)
C	Carry (The operational result is to have carry out for addition or to borrow carry for subtraction in unsigned number system)
AC	Auxiliary Carry (If there is a carry out from low nibble after the result of ALU operation, this bit is set to 1)
IO.n	The bit of register
M.n,	Only addressed in 0~0x3F (0~63) is allowed

7.1. Data Transfer Instructions

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

ldt16 ***word***	Move 16-bit counting values in Timer16 to memory in word.
	Example: ldt16 word;
	Result: word ← 16-bit timer
	Affected flags: 『N』Z 『N』C 『N』AC 『N』OV
	
	Application Example:
	--- 	
	word T16val ; // declare a RAM word
	…
	clear lb@ T16val ; // clear T16val (LSB)
	clear hb@ T16val ; // clear T16val (MSB)
	stt16 T16val ; // initial T16 with 0
	…
	set1 t16m.5 ; // enable Timer16
	…
	set0 t16m.5 ; // disable Timer 16
	ldt16 T16val ; // save the T16 counting value to T16val
	….
	--- 	

stt16 word	Store 16-bit data from memory in word to Timer16.
	Example: *stt16* word;
	Result: 16-bit timer ←word
	Affected flags: 『N』Z 『N』C 『N』AC 『N』OV
	
	Application Example:
	--- 	
	word T16val ; // declare a RAM word
	…
	mov a, 0x34 ;
	mov lb@ T16val , a ; // move 0x34 to T16val (LSB)
	mov a, 0x12 ;
	mov hb@ T16val , a ; // move 0x12 to T16val (MSB)
	stt16 T16val ; // initial T16 with 0x1234
	…
	--- 	

idxm a, index	Move data from specified memory to ACC by indirect method. It needs 2T to execute this instruction.
	Example: idxm a, index;
	Result: a ← [index], where index is declared by word.
	Affected flags: 『N』Z 『N』C 『N』AC 『N』OV
	
	Application Example:
	--- 	
	word RAMIndex ; // declare a RAM pointer
	…
	mov a, 0x5B ; // assign pointer to an address (LSB)
	mov lb@RAMIndex, a ; // save pointer to RAM (LSB)
	mov a, 0x00 ; // assign 0x00 to an address (MSB), should be 0
	mov hb@RAMIndex, a ; // save pointer to RAM (MSB)
	…
	idxm a, RAMIndex ; // move memory data in address 0x5B to ACC
	--- 	

Idxm **index, a**	Move data from ACC to specified memory by indirect method. It needs 2T to execute this instruction.
	Example: *idxm* index, a;
	Result: [index] ← a; where index is declared by word.
	Affected flags: 『N』Z 『N』C 『N』AC 『N』OV
	
	Application Example:
	--- 	
	word RAMIndex ; // declare a RAM pointer
	…
	mov a, 0x5B ; // assign pointer to an address (LSB)
	mov lb@RAMIndex, a ; // save pointer to RAM (LSB)
	mov a, 0x00 ; // assign 0x00 to an address (MSB), should be 0
	mov hb@RAMIndex, a ; // save pointer to RAM (MSB)
	…
	mov a, 0xA5 ;
	idxm RAMIndex, a ; // move 0xA5 to memory in address 0x5B
	--- 	

xch M	Exchange data between ACC and memory Example: xch MEM ; Result: MEM ← a , a ← MEM Affected flags: 『N』Z 『N』C 『N』AC 『N』OV

pushaf	Move the ACC and flag register to memory that address specified in the stack pointer.
	Example: pushaf;
	Result: [sp] ← {flag, ACC};
	sp ← sp + 2 ;
	Affected flags: 『N』Z 『N』C 『N』AC 『N』OV
	
	Application Example:
	--- 	
	.romadr 0x10 ; // ISR entry address
	pushaf ; // put ACC and flag into stack memory
	… // ISR program
	… // ISR program
	popaf ; // restore ACC and flag from stack memory
	reti ;
	--- 	

popaf	Restore ACC and flag from the memory which address is specified in the stack pointer. Example: popaf; Result: sp ← sp - 2 ; {Flag, ACC} ← [sp] ; Affected flags: 『Y』Z 『Y』C 『Y』AC 『Y』OV

7.2. Arithmetic Operation Instructions

add **a, I**	Add immediate data with ACC, then put result into ACC Example: *add* a, 0x0f ; Result: a ← a + 0fh Affected flags: 『Y』Z 『Y』C 『Y』AC 『Y』OV
add **a, M**	Add data in memory with ACC, then put result into ACC Example: *add* a, MEM ; Result: a ← a + MEM Affected flags: 『Y』Z 『Y』C 『Y』AC 『Y』OV
add **M, a**	Add data in memory with ACC, then put result into memory Example: *add* MEM, a; Result: MEM ← a + MEM Affected flags: 『Y』Z 『Y』C 『Y』AC 『Y』OV
addc **a, M**	Add data in memory with ACC and carry bit, then put result into ACC Example: *addc* a, MEM ; Result: a ← a + MEM + C Affected flags: 『Y』Z 『Y』C 『Y』AC 『Y』OV
addc **M, a**	Add data in memory with ACC and carry bit, then put result into memory Example: *addc* MEM, a ; Result: MEM ← a + MEM + C Affected flags: 『Y』Z 『Y』C 『Y』AC 『Y』OV
addc **a**	Add carry with ACC, then put result into ACC Example: *addc* a ; Result: a ← a + C Affected flags: 『Y』Z 『Y』C 『Y』AC 『Y』OV
addc **M**	Add carry with memory, then put result into memory Example: *addc* MEM ; Result: MEM ← MEM + C Affected flags: 『Y』Z 『Y』C 『Y』AC 『Y』OV
nadd a, M	Add negative logic (2's complement) of ACC with memory Example: nadd a, MEM ; Result: a ← ∓a + MEM Affected flags: 『Y』Z 『Y』C 『Y』AC 『Y』OV ICE doesn't support
nadd M, a	Add negative logic (2's complement) of memory with ACC Example: nadd MEM, a ; Result: MEM ← ∓MEM + a Affected flags: 『Y』Z 『Y』C 『Y』AC 『Y』OV ICE doesn't support
sub **a, I**	Subtraction immediate data from ACC, then put result into ACC. Example: *sub* a, 0x0f; Result: a ← a - 0fh (a + [2's complement of 0fh]) Affected flags: 『Y』Z 『Y』C 『Y』AC 『Y』OV
sub **a, M**	Subtraction data in memory from ACC, then put result into ACC Example: *sub* a, MEM ; Result: a ← a - MEM (a + [2's complement of M]) Affected flags: 『Y』Z 『Y』C 『Y』AC 『Y』OV

sub **M, a**	Subtraction data in ACC from memory, then put result into memory Example: *sub* MEM, a; Result: MEM ← MEM - a (MEM + [2's complement of a]) Affected flags: 『Y』Z 『Y』C 『Y』AC 『Y』OV
subc **a, M**	Subtraction data in memory and carry from ACC, then put result into ACC Example: *subc* a, MEM; Result: a ← a – MEM - C Affected flags: 『Y』Z 『Y』C 『Y』AC 『Y』OV
subc **M, a**	Subtraction ACC and carry bit from memory, then put result into memory Example: *subc* MEM, a ; Result: MEM ← MEM – a - C Affected flags: 『Y』Z 『Y』C 『Y』AC 『Y』OV
subc **a**	Subtraction carry from ACC, then put result into ACC Example: *subc* a; Result: a ← a - C Affected flags: 『Y』Z 『Y』C 『Y』AC 『Y』OV
subc **M**	Subtraction carry from the content of memory, then put result into memory Example: *subc* MEM; Result: MEM ← MEM - C Affected flags: 『Y』Z 『Y』C 『Y』AC 『Y』OV
inc **M**	Increment the content of memory Example: *inc* MEM ; Result: MEM ← MEM + 1 Affected flags: 『Y』Z 『Y』C 『Y』AC 『Y』OV
dec **M**	Decrement the content of memory Example: *dec* MEM; Result: MEM ← MEM - 1 Affected flags: 『Y』Z 『Y』C 『Y』AC 『Y』OV
clear **M**	Clear the content of memory Example: *clear* MEM ; Result: MEM ← 0 Affected flags: 『N』Z 『N』C 『N』AC 『N』OV

7.3. Shift Operation Instructions

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

7.4. Logic Operation Instructions

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

xor **IO, a**	Perform logic XOR on ACC and IO register, then put result into IO register
	Example: *xor pa, a ;*
	Result: pa ← a ^ pa ; // pa is the data register of port A
	Affected flags: 『N』 Z 『N』 C 『N』 AC 『N』 OV

xor **a, M**	Perform logic XOR on ACC and memory, then put result into ACC Example: *xor* a, MEM ; Result: a ← a ^ RAM10 Affected flags: 『Y』 Z 『N』 C 『N』 AC 『N』 OV
xor **M, a**	Perform logic XOR on ACC and memory, then put result into memory Example: *xor* MEM, a ; Result: MEM ← a ^ MEM Affected flags: 『Y』 Z 『N』 C 『N』 AC 『N』 OV

not **a**	Perform 1's complement (logical complement) of ACC
	Example: *not* a ;
	Result: a ← ~a
	Affected flags: 『Y』Z 『N』C 『N』AC 『N』OV
	Application Example:
	--- 	
	mov a, 0x38 ; // ACC=0X38
	not a ; // ACC=0XC7
	--- 	

not **M**	Perform 1's complement (logical complement) of memory
	Example: *not* MEM ;
	Result: MEM ← ~MEM
	Affected flags: 『Y』Z 『N』C 『N』AC 『N』OV
	
	Application Example:
	--- 	
	mov a, 0x38 ;
	mov mem, a ; // mem = 0x38
	not mem ; // mem = 0xC7
	--- 	

neg **a**	Perform 2's complement of ACC
	Example: *neg* a;
	Result: a ← ∓a
	Affected flags: 『Y』Z 『N』C 『N』AC 『N』OV
	
	Application Example:
	--- 	
	mov a, 0x38 ; // ACC=0X38
	neg a ; // ACC=0XC8
	--- 	

neg **M**	Perform 2's complement of memory
	Example: *neg* MEM;
	Result: MEM ← ∓MEM
	Affected flags: 『Y』Z 『N』C 『N』AC 『N』OV
	
	Application Example:
	--- 	
	mov a, 0x38 ;
	mov mem, a ; // mem = 0x38
	not mem ; // mem = 0xC8
	--- 	

comp a, M	Compare ACC with the content of memory
	Example: comp a, MEM;
	Result: Flag will be changed by regarding as (a - MEM)
	Affected flags: 『Y』Z 『Y』C 『Y』AC 『Y』OV
	Application Example:
	--- 	
	mov a, 0x38 ;
	mov mem, a ;
	comp a, mem ; // Z flag is set
	mov a, 0x42 ;
	mov mem, a ;
	mov a, 0x38 ;
	comp a, mem ; // C flag is set
	ICE doesn't support

comp M, a	Compare ACC with the content of memory Example: comp MEM, a; Result: Flag will be changed by regarding as (MEM - a) Affected flags: 『Y』Z 『Y』C 『Y』AC 『Y』OV ICE doesn't support

7.5 Bit Operation Instructions

set0 **IO.n**	Set bit n of IO port to low Example: *set0* pa.5 ; Result: set bit 5 of port A to low Affected flags: 『N』Z 『N』C 『N』AC 『N』OV
set1 **IO.n**	Set bit n of IO port to high Example: *set1* pa.5 ; Result: set bit 5 of port A to high Affected flags: 『N』Z 『N』C 『N』AC 『N』OV
set0 **M.n**	Set bit n of memory to low Example: *set0* MEM.5 ; Result: set bit 5 of MEM to low Affected flags: 『N』Z 『N』C 『N』AC 『N』OV
set1 **M.n**	Set bit n of memory to high Example: *set1* MEM.5 ; Result: set bit 5 of MEM to high Affected flags: 『N』Z 『N』C 『N』AC 『N』OV

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

7.6. Conditional Operation Instructions

***ceqsn* a, I**	Compare ACC with immediate data and skip next instruction if both are equal. Flag will be changed like as (a ← a - I) Example: *ceqsn* a, 0x55 ; *inc* MEM ; *goto* error ; Result: If a=0x55, then “goto error”; otherwise, “inc MEM”. Affected flags: 『Y』Z 『Y』C 『Y』AC 『Y』OV
***ceqsn* a, M**	Compare ACC with memory and skip next instruction if both are equal. Flag will be changed like as (a ← a - M) Example: *ceqsn* a, MEM; Result: If a=MEM, skip next instruction Affected flags: 『Y』Z 『Y』C 『Y』AC 『Y』OV
cneqsn a, M	Compare ACC with memory and skip next instruction if both are not equal. Flag will be changed like as (a ← a - M) Example: *cneqsn* a, MEM; Result: If a≠MEM, skip next instruction

	Affected flags: 『Y』Z 『Y』C 『Y』AC 『Y』OV
cneqsn a, I	Compare ACC with immediate data and skip next instruction if both are no equal. Flag will be changed like as (a ← a - I) Example: *cneqsn* a,0x55 ; *inc* MEM ; *goto* error ; Result: If a≠0x55, then “goto error”; Otherwise, “inc MEM”. Affected flags: 『Y』Z 『Y』C 『Y』AC 『Y』OV
***t0sn* IO.n**	Check IO bit and skip next instruction if it’s low Example: *t0sn* pa.5; Result: If bit 5 of port A is low, skip next instruction Affected flags: 『N』Z 『N』C 『N』AC 『N』OV
***t1sn* IO.n**	Check IO bit and skip next instruction if it’s high Example: *t1sn* pa.5 ; Result: If bit 5 of port A is high, skip next instruction Affected flags: 『N』Z 『N』C 『N』AC 『N』OV
***t0sn* M.n**	Check memory bit and skip next instruction if it’s low Example: *t0sn* MEM.5 ; Result: If bit 5 of MEM is low, then skip next instruction Affected flags: 『N』Z 『N』C 『N』AC 『N』OV
***t1sn* M.n**	Check memory bit and skip next instruction if it’s high EX: *t1sn* MEM.5 ; Result: If bit 5 of MEM is high, then skip next instruction Affected flags: 『N』Z 『N』C 『N』AC 『N』OV
***izsn* a**	Increment ACC and skip next instruction if ACC is zero Example: *izsn* a; Result: a ← a + 1,skip next instruction if a = 0 Affected flags: 『Y』Z 『Y』C 『Y』AC 『Y』OV
***dzsn* a**	Decrement ACC and skip next instruction if ACC is zero Example: *dzsn* a; Result: A ← A - 1,skip next instruction if a = 0 Affected flags: 『Y』Z 『Y』C 『Y』AC 『Y』OV
***izsn* M**	Increment memory and skip next instruction if memory is zero Example: *izsn* MEM; Result: MEM ← MEM + 1, skip next instruction if MEM= 0 Affected flags: 『Y』Z 『Y』C 『Y』AC 『Y』OV
***dzsn* M**	Decrement memory and skip next instruction if memory is zero Example: *dzsn* MEM; Result: MEM ← MEM - 1, skip next instruction if MEM = 0 Affected flags: 『Y』Z 『Y』C 『Y』AC 『Y』OV

7.7. System control Instructions

call **label**	Function call, address can be full range address space Example: *call* function1; Result: [sp] ← pc + 1 pc ← function1 sp ← sp + 2 Affected flags: 『N』Z 『N』C 『N』AC 『N』OV
goto **label**	Go to specific address which can be full range address space Example: *goto* error; Result: Go to error and execute program. Affected flags: 『N』Z 『N』C 『N』AC 『N』OV
ret **I**	Place immediate data to ACC, then return Example: *ret* 0x55; Result: A ← 55h ret ; Affected flags: 『N』Z 『N』C 『N』AC 『N』OV
ret	Return to program which had function call Example: ret; Result: sp ← sp – 2 pc ← [sp] Affected flags: 『N』Z 『N』C 『N』AC 『N』OV
reti	Return to programfrom interrupt service routine. After this command is executed, global interrupt is enabled automatically. Example: reti; Affected flags: 『N』Z 『N』C 『N』AC 『N』OV
nop	No operation Example: nop; Result: nothing changed Affected flags: 『N』Z 『N』C 『N』AC 『N』OV
wdreset	Reset Watchdog timer. Example: wdreset ; Result: Reset Watchdog timer. Affected flags: 『N』Z 『N』C 『N』AC 『N』OV

pcadd a	Next program counter is current program counter plus ACC.
	Example: pcadd a;
	Result: pc ← pc + a
	Affected flags: 『N』Z 『N』C 『N』AC 『N』OV
	Application Example:
	---------- 	
	…
	mov a, 0x02 ;
	pcadd a ; // PC <- PC+2
	goto err1 ;
	goto correct ; // jump here
	goto err2 ;
	goto err3 ;
	…
	correct: // jump here
	…
	---------- 	

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

7.8. Summary of Instructions Execution Cycle

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

7.9. Summary of affected flags by Instructions

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

7.10. BIT definition

Bit defined: Only addressed at 0x00 ~ 0x3F.

8. Code Options

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
LVR	4.0V	Select LVR = 4.0V
	3.5V	Select LVR = 3.5V
	3.0V	Select LVR = 3.0V
	2.75V	Select LVR = 2.75V
	2.5V	Select LVR = 2.5V
	2.2V	Select LVR = 2.2V
	2.0V	**Select LVR = 2.0V**
	1.8V	Select LVR = 1.8V
Boot-up_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
Drive	Low	IO Low driving and sinking current
	Normal	**IO Normal driving and sinking current**
LCD2 (please refer to MISC.4)	**Disable**	**VDD/2 bias voltage generator disabled, PB0 PA[0,3,4] are normal IO pins**
	PB0_A034	VDD/2 bias voltage generator enabled, PB0 PA[0,3,4] are VDD/2 if input mode
Comparator_Edge	**All_Edge**	**The comparator will trigger an interrupt on the rising edge or falling edge**
	Rising_Edge	The comparator will trigger an interrupt on the rising edge
	Falling_Edge	The comparator will trigger an interrupt on the falling edge

Note: The **Bolded** options are the default options.

9. Special Notes

This chapter is to remind user who use PMS154G series IC in order to avoid frequent errors upon operation.

9.1. Using IC

9.1.1. IO pin usage and setting

(1) IO pin as digital input

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

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

- ◆ Configure IO pin as input
- ◆ Set corresponding bit to “1” in PXDIER
- ◆ For those IO pins of PA that are not used, PADIER[1:2] should be set low in order to prevent them from leakage.

(3) PA5 is set to be output pin

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

(4) PA5 is set to be PRSTB input pin

- ◆ Configure PA5 as input
- ◆ Set CLKMD.0=1 to enable PA5 as PRSTB input pin

(5) PA5 is set to be input pin and to connect with a push button or a switch by a long wire

- ◆ Needs to put a >33Ω resistor in between PA5 and the long wire
- ◆ Avoid using PA5 as input in such application.

(6) PA7 and PA6 as external crystal oscillator

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

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

9.1.2. 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.

◆ Example : Switch system clock from ILRC to IHRC/2

```
CLKMD   =   0x36;          // switch to IHRC, ILRC cannot 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. 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. TIMER16 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 Calibration

(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	≧ 1.8V	≧ 1.8V
4MHz	≧ 2.5V	≧ 2.5V
8MHz	≧ 3.5V	≧ 3.5V

Table 9: LVR setting for reference

(1) The setting of LVR (1.8V ~ 4.0V) 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.

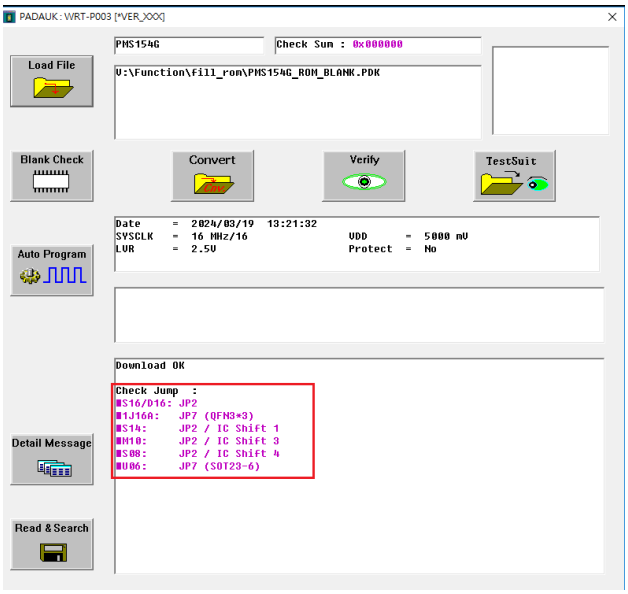
(3) The LVR function will be invalid when IC in stopexe or stopsys mode.

9.1.8. Program writing

Please use PDK5S-P-003Bx to program. PDK3S-P-002 or older versions do not support programming PMS154G.

There are 6 signals for programming PMS154G: PA3, PA4, PA5, PA6, V_{DD}, and GND.

Jumper connection: Please follow the instruction inside the writer 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 11V.
  (2) V_{DD} may be higher than 8V, and its maximum current may reach about 20mA.
  (3) All other signal pins level (except GND) are the same as V_{DD}.

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

Important Cautions：

- **You MUST follow the instructions on APN004 and APN011 for programming IC on the handler.**
- **Connecting a 0.01uF capacitor between VDD and GND at the handler port to the IC is always good for suppressing disturbance. But please DO NOT connect with > 0.01uF capacitor, otherwise, programming mode may be fail.**

9.2. Using ICE

(1) It is recommended to use 5S-I-S01/2(B) for emulation of PMS154G.

(2) 5S-I-S01/2(B) supports PMS154G 1-FPPA MCU emulation work, the following items should be noted when using 5S-I-S01/2(B) to emulate PMS154G:

- 5S-I-S01/2(B) doesn't support the instruction NADD/COMP.
- 5S-I-S01/2(B) doesn't support SYSCLK=ILRC/16.
- 5S-I-S01/2(B) doesn't support the dynamic setting of function ***misc.4*** (Only fix to 0 or 1).
- 5S-I-S01/2(B) doesn't support the function ***Tm2.gpcrs***/***Tm3.gpcrs.***
- The PA3 output function will be affected when GPCS selects output to PA0 output.
- 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.
- 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.
- When using 5S-I-S01/2(B) for simulation, changing the value of tm2ct/tm3ct will affect the duty during timer2/timer3 period mode. But it will not be affected for the actual IC.
- With 5S-I-S01/2(B) simulation, when fast wake-up is enabled, the watchdog overflow reset time becomes very short. Actually, it has not effect on IC.
- The power-down command Stopsys does not support the comparator wake-up function. When using 5S-I-S01 / 2 (B), it should be noted that the comparator enable should be set to the off state before entering the power-down mode. If the enable state is turned on, the comparator will be mistakenly awakened.
- Fast Wakeup time is different from 5S-I-S01/2(B): 128 SysClk, PMS154G: 45 ILRC.
- Watch dog time out period is different from 5S-I-S01/2:

WDT period	5S-I-S01/2(B)	PMS154G
misc[1:0]=00	2048 * T_{ILRC}	8k * T_{ILRC}
misc[1:0]=01	4096 * T_{ILRC}	16k * T_{ILRC}
misc[1:0]=10	16384 * T_{ILRC}	64k * T_{ILRC}
misc[1:0]=11	256 * T_{ILRC}	256k * T_{ILRC}