# PMS163

# 13-ch Touch Keys OTP MCU with 12-bit ADC

***Datasheet***

***Version 0.05 – Mar. 17, 2023***

**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 which 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 |
|---|---|---|
| 0.02 | 2022/07/15 | 1. Amend Section 5.14, 6.33, 6.34<br>2. Other known details bug correct. |
| 0.03 | 2022/10/13 | 1. Added bypass mode description to touch section.<br>2. Amend MISC2, EOSCR register description.<br>3. Other known details bug correct. |
| 0.04 | 2023/02/14 | 1. Amend "IMPORTANT NOTICE"<br>2. Added Section 6.37 and 6.38<br>3. Amend Section 5.15<br>4. Other known details bug correct. |
| 0.05 | 2023/03/17 | 1. Added SOP16C<br>2. Amend Section 9.13 |

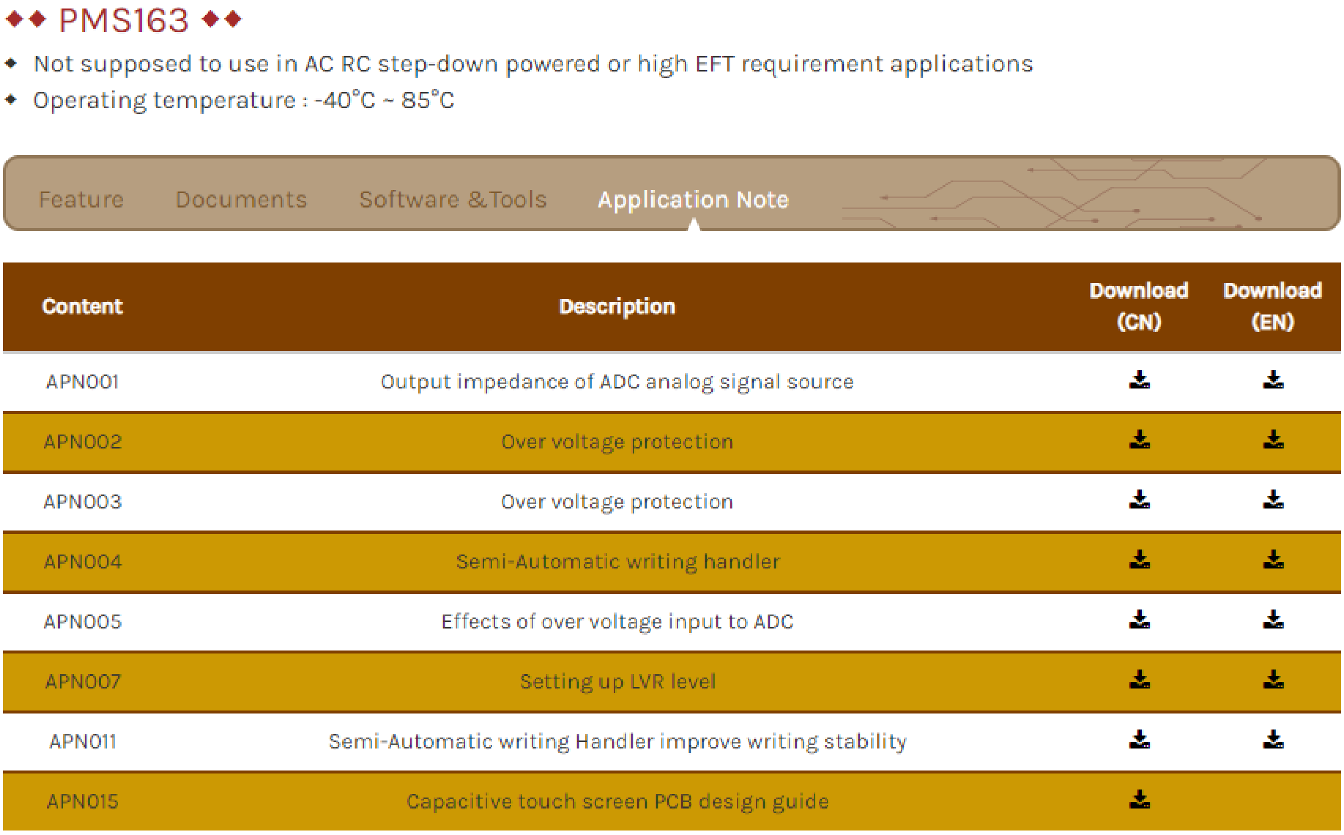
## Usage Warning

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.

http://www.padauk.com.tw/en/product/show.aspx?num=156&kw=PMS163

(The following picture are for reference only.)



◆◆ PMS163 ◆◆

- Not supposed to use in AC RC step-down powered or high EFT requirement applications
- Operating temperature : -40°C ~ 85°C

| Content | Description | Download (CN) | Download (EN) |
|---|---|---|---|
| APN001 | Output impedance of ADC analog signal source | ⬇ | ⬇ |
| APN002 | Over voltage protection | ⬇ | ⬇ |
| APN003 | Over voltage protection | ⬇ | ⬇ |
| APN004 | Semi-Automatic writing handler | ⬇ | ⬇ |
| APN005 | Effects of over voltage input to ADC | ⬇ | ⬇ |
| APN007 | Setting up LVR level | ⬇ | ⬇ |
| APN011 | Semi-Automatic writing Handler improve writing stability | ⬇ | ⬇ |
| APN015 | Capacitive touch screen PCB design guide | ⬇ | |

# 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

- 2.5KW OTP program memory
- 160 Bytes data RAM
- Maximum 13 IO pins can be selected as TOUCH PAD individually
- One hardware 16-bit timer
- Two hardware 8-bit timer with PWM generation
- Three hardware 11-bit PWM generators (PWMG0, PWMG1 & PWMG2)
- One hardware comparator
- Provide 1T 8x8 hardware multiplier
- 14 IO pins with optional pull-high/pull-low resistor
- Every IO pin can be configured to enable wake-up function
- PA6/PA7/PB7 support large sink current: 80mA
- Bandgap circuit to provide 1.2V Bandgap voltage
- Up to 12-channel 12-bit resolution ADC with one channel comes from internal Bandgap reference voltage or 0.25*$V_{DD}$
- Provide ADC reference high voltage: external input, internal $V_{DD}$, Bandgap(1.20V), 4V, 3V, 2V
- Clock sources: internal high RC oscillator(IHRC) and internal low RC oscillator(ILRC)
- For every wake-up enabled IO, two optional wake-up speed are supported: normal and fast
- Sixteen Levels of LVR reset: 4.5V, 4.0V, 3.75V, 3.5V, 3.3V, 3.15V, 3.0V, 2.7V, 2.5V, 2.4V, 2.3V, 2.2V, 2.1V, 2.0V, 1.9V, 1.8V
- Four selectable external interrupt pin
- Internal LDO for touch noise immunity
- One low-power clock (NILRC) wake-up stopsys regularly

## 1.3. CPU Features

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

## 1.4. Ordering/ Package Information

- PMS163-U06: SOT23-6 (60mil)
- PMS163-S08: SOP8 (150mil)
- PMS163-M10: MSOP10 (118mil)
- PMS163-4N10: DFN3*3-10P (0.5pitch)
- PMS163-S14: SOP14 (150mil)
- PMS163-S16A:SOP16A (150mil)
- PMS163-S16B:SOP16B (150mil)
- PMS163-S16C:SOP16C (150mil)
- PMS163-2J16A: QFN4*4-16P (0.65pitch)
- PMS163-1J16A: QFN3*3-16P (0.5pitch))

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

## 2. General Description and Block Diagram

The PMS163 is an ADC-Type, fully static, OTP-based 8 bit CMOS Touch-Key MCU; it employs RISC architecture and most the instructions are executed in one cycle except that few instructions are two cycles that handle indirect memory access.

A maximum 13-ch touch keys controller is built inside PMS163. Besides, PMS163 also includes 2.5KW OTP program memory, 160 bytes data SRAM, one hardware 16-bit timer, two are 8-bit timers with PWM generation, and three hardware 11-bit timers with PWM generation are also included.

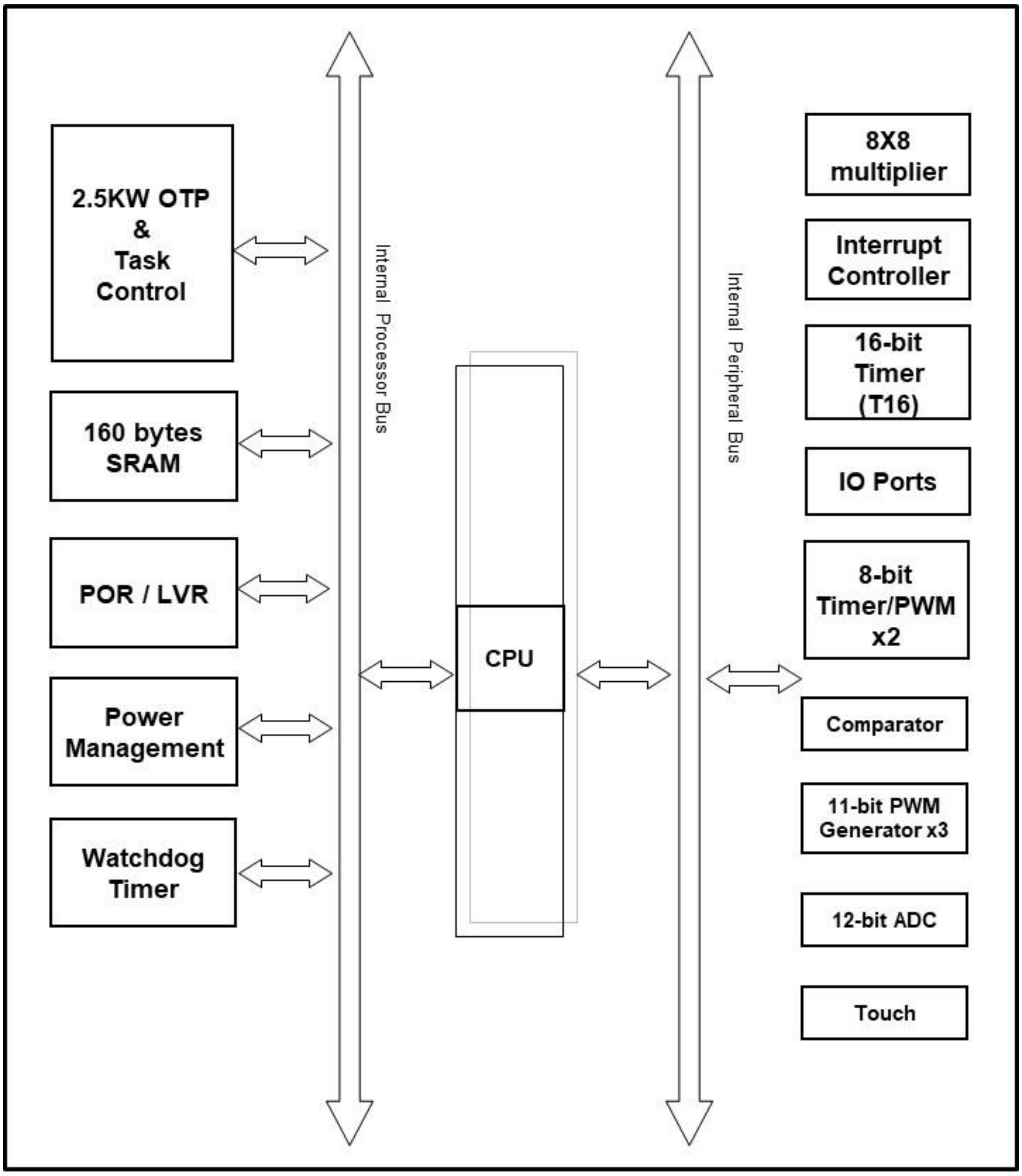
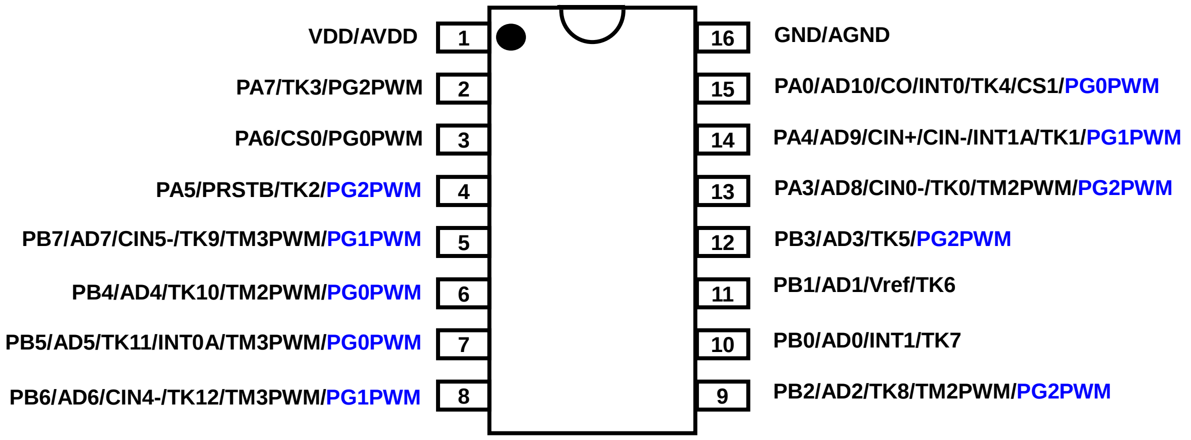
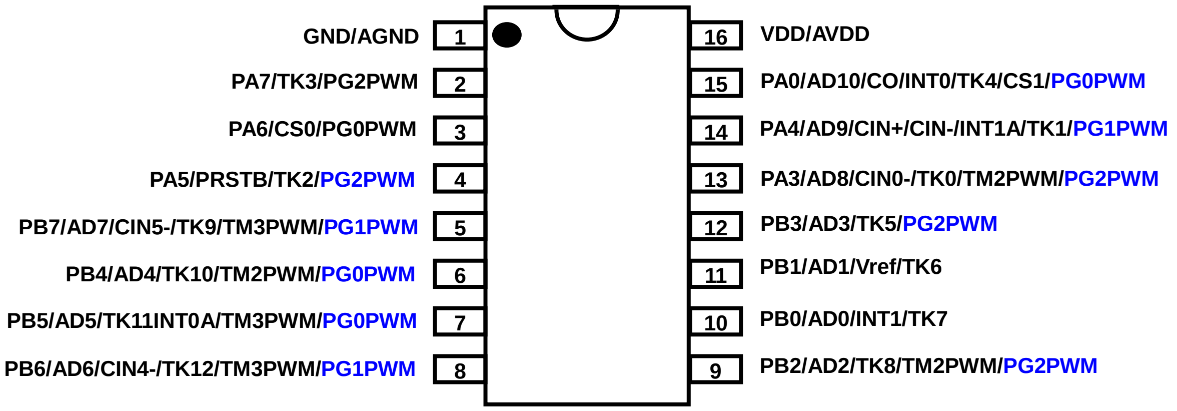


Fig. 1: PMS163 Block Diagram

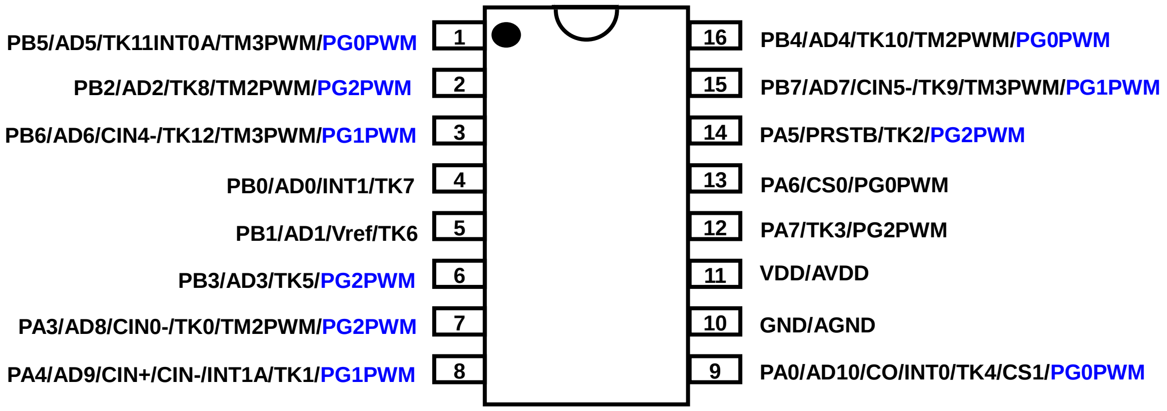
## 3. Pin Definition and Functional Description


VDD/AVDD 1
PA7/TK3/PG2PWM 2
PA6/CS0/PG0PWM 3
PA5/PRSTB/TK2/PG2PWM 4
PB7/AD7/CIN5-/TK9/TM3PWM/PG1PWM 5
PB4/AD4/TK10/TM2PWM/PG0PWM 6
PB5/AD5/TK11/INT0A/TM3PWM/PG0PWM 7
PB6/AD6/CIN4-/TK12/TM3PWM/PG1PWM 8
16 GND/AGND
15 PA0/AD10/CO/INT0/TK4/CS1/PG0PWM
14 PA4/AD9/CIN+/CIN-/INT1A/TK1/PG1PWM
13 PA3/AD8/CIN0-/TK0/TM2PWM/PG2PWM
12 PB3/AD3/TK5/PG2PWM
11 PB1/AD1/Vref/TK6
10 PB0/AD0/INT1/TK7
9 PB2/AD2/TK8/TM2PWM/PG2PWM

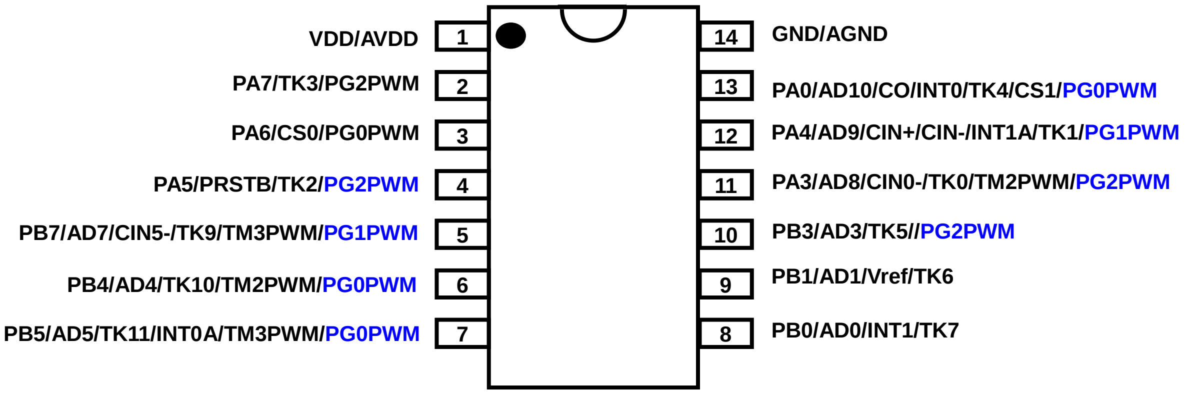
PMS163-S16A (SOP16A-150mil)


GND/AGND 1
PA7/TK3/PG2PWM 2
PA6/CS0/PG0PWM 3
PA5/PRSTB/TK2/PG2PWM 4
PB7/AD7/CIN5-/TK9/TM3PWM/PG1PWM 5
PB4/AD4/TK10/TM2PWM/PG0PWM 6
PB5/AD5/TK11INT0A/TM3PWM/PG0PWM 7
PB6/AD6/CIN4-/TK12/TM3PWM/PG1PWM 8
16 VDD/AVDD
15 PA0/AD10/CO/INT0/TK4/CS1/PG0PWM
14 PA4/AD9/CIN+/CIN-/INT1A/TK1/PG1PWM
13 PA3/AD8/CIN0-/TK0/TM2PWM/PG2PWM
12 PB3/AD3/TK5/PG2PWM
11 PB1/AD1/Vref/TK6
10 PB0/AD0/INT1/TK7
9 PB2/AD2/TK8/TM2PWM/PG2PWM

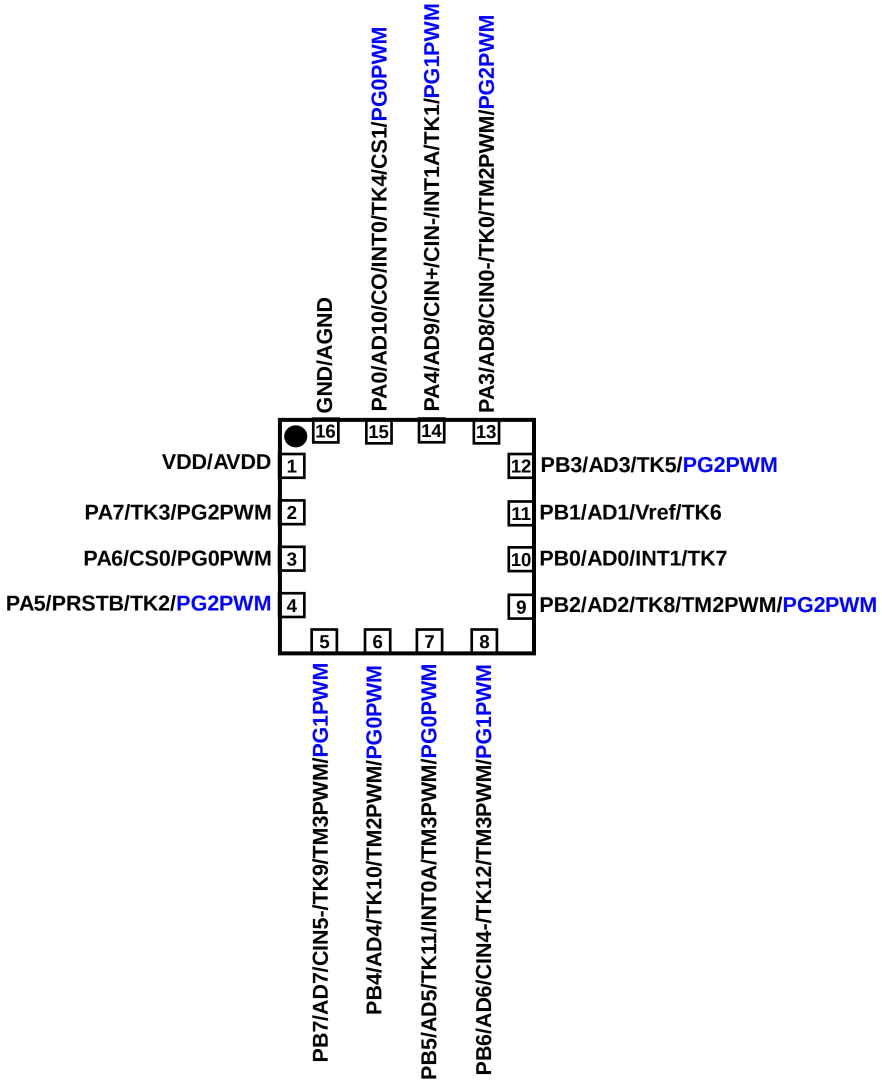
PMS163 -S16B (SOP16B-150mil)


PB5/AD5/TK11INT0A/TM3PWM/PG0PWM 1
PB2/AD2/TK8/TM2PWM/PG2PWM 2
PB6/AD6/CIN4-/TK12/TM3PWM/PG1PWM 3
PB0/AD0/INT1/TK7 4
PB1/AD1/Vref/TK6 5
PB3/AD3/TK5/PG2PWM 6
PA3/AD8/CIN0-/TK0/TM2PWM/PG2PWM 7
PA4/AD9/CIN+/CIN-/INT1A/TK1/PG1PWM 8
16 PB4/AD4/TK10/TM2PWM/PG0PWM
15 PB7/AD7/CIN5-/TK9/TM3PWM/PG1PWM
14 PA5/PRSTB/TK2/PG2PWM
13 PA6/CS0/PG0PWM
12 PA7/TK3/PG2PWM
11 VDD/AVDD
10 GND/AGND
9 PA0/AD10/CO/INT0/TK4/CS1/PG0PWM

PMS163 -S16C (SOP16C-150mil)

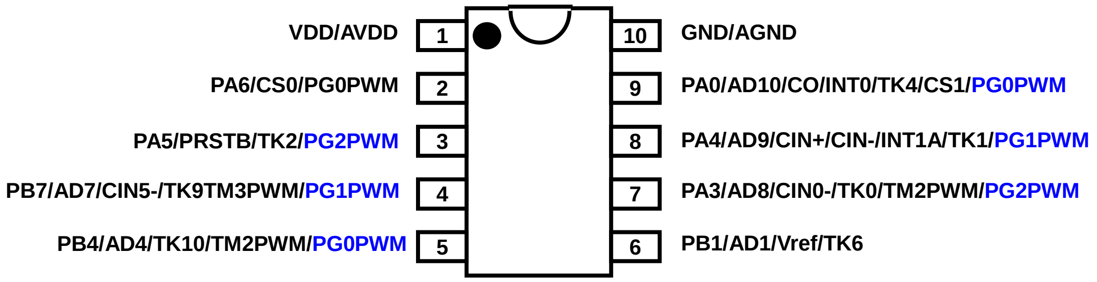

VDD/AVDD
1
PA7/TK3/PG2PWM
2
PA6/CS0/PG0PWM
3
PA5/PRSTB/TK2/PG2PWM
4
PB7/AD7/CIN5-/TK9/TM3PWM/PG1PWM
5
PB4/AD4/TK10/TM2PWM/PG0PWM
6
PB5/AD5/TK11/INT0A/TM3PWM/PG0PWM
7
14
GND/AGND
13
PA0/AD10/CO/INT0/TK4/CS1/PG0PWM
12
PA4/AD9/CIN+/CIN-/INT1A/TK1/PG1PWM
11
PA3/AD8/CIN0-/TK0/TM2PWM/PG2PWM
10
PB3/AD3/TK5//PG2PWM
9
PB1/AD1/Vref/TK6
8
PB0/AD0/INT1/TK7

PMS163 -S14 (SOP14-150mil)

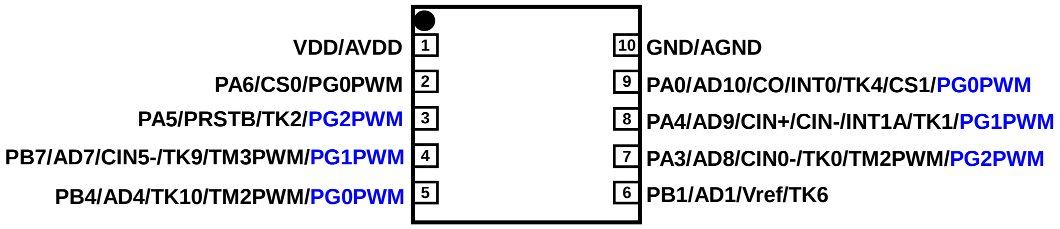

GND/AGND
PA0/AD10/CO/INT0/TK4/CS1/PG0PWM
PA4/AD9/CIN+/CIN-/INT1A/TK1/PG1PWM
PA3/AD8/CIN0-/TK0/TM2PWM/PG2PWM
16
15
14
13
VDD/AVDD
1
PA7/TK3/PG2PWM
2
PA6/CS0/PG0PWM
3
PA5/PRSTB/TK2/PG2PWM
4
12
PB3/AD3/TK5/PG2PWM
11
PB1/AD1/Vref/TK6
10
PB0/AD0/INT1/TK7
9
PB2/AD2/TK8/TM2PWM/PG2PWM
5
6
7
8
PB7/AD7/CIN5-/TK9/TM3PWM/PG1PWM
PB4/AD4/TK10/TM2PWM/PG0PWM
PB5/AD5/TK11/INT0A/TM3PWM/PG0PWM
PB6/AD6/CIN4-/TK12/TM3PWM/PG1PWM

PMS163 -2J16A (QFN4*4-16P-0.65pitch)

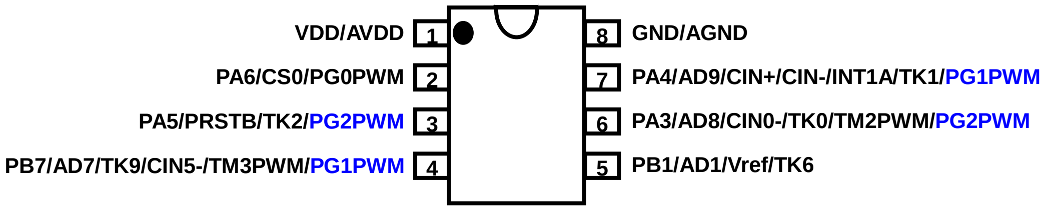
PMS163 -1J16A (QFN3*3-16P-0.5pitch)


VDD/AVDD
1
PA6/CS0/PG0PWM
2
PA5/PRSTB/TK2/PG2PWM
3
PB7/AD7/CIN5-/TK9TM3PWM/PG1PWM
4
PB4/AD4/TK10/TM2PWM/PG0PWM
5
10
GND/AGND
9
PA0/AD10/CO/INT0/TK4/CS1/PG0PWM
8
PA4/AD9/CIN+/CIN-/INT1A/TK1/PG1PWM
7
PA3/AD8/CIN0-/TK0/TM2PWM/PG2PWM
6
PB1/AD1/Vref/TK6

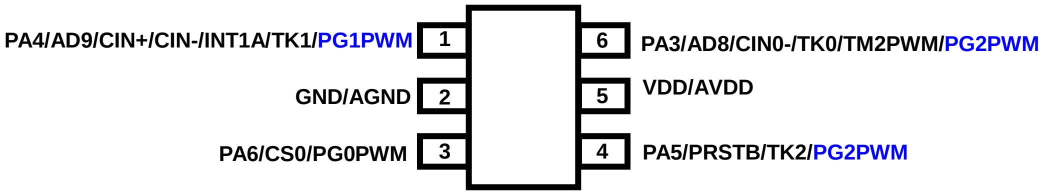
PMS163 -M10 (MSOP10-118mil)


VDD/AVDD
1
PA6/CS0/PG0PWM
2
PA5/PRSTB/TK2/PG2PWM
3
PB7/AD7/CIN5-/TK9/TM3PWM/PG1PWM
4
PB4/AD4/TK10/TM2PWM/PG0PWM
5
10
GND/AGND
9
PA0/AD10/CO/INT0/TK4/CS1/PG0PWM
8
PA4/AD9/CIN+/CIN-/INT1A/TK1/PG1PWM
7
PA3/AD8/CIN0-/TK0/TM2PWM/PG2PWM
6
PB1/AD1/Vref/TK6

PMS163 -4N10 (DFN3*3-10P-0.5pitch)


VDD/AVDD
1
PA6/CS0/PG0PWM
2
PA5/PRSTB/TK2/PG2PWM
3
PB7/AD7/TK9/CIN5-/TM3PWM/PG1PWM
4
8
GND/AGND
7
PA4/AD9/CIN+/CIN-/INT1A/TK1/PG1PWM
6
PA3/AD8/CIN0-/TK0/TM2PWM/PG2PWM
5
PB1/AD1/Vref/TK6

PMS163 -S08 (SOP8-150mil)


PA4/AD9/CIN+/CIN-/INT1A/TK1/PG1PWM
1
GND/AGND
2
PA6/CS0/PG0PWM
3
6
PA3/AD8/CIN0-/TK0/TM2PWM/PG2PWM
5
VDD/AVDD
4
PA5/PRSTB/TK2/PG2PWM

PMS163 -U06 (SOT23-6 60mil)

## Pin Description

| Pin Name | Pin Type &<br>Buffer Type | Description |
|---|---|---|
| PA7 /<br>PG2PWM /<br>TK3 | IO<br>ST /<br>CMOS | The functions of this pin can be:<br>(1) Bit 7 of port A. It can be configured as digital input or two-state output, with pull-high resistor.<br>(2) Output of 11-bit PWM generator PWMG2.<br>(3) Touch Key 3.<br>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 /<br>PG0PWM /<br>CS0 | IO<br>ST /<br>CMOS | The functions of this pin can be:<br>(1) Bit 6 of port A. It can be configured as digital input or two-state output, with pull-high resistor.<br>(2) Output of 11-bit PWM generator PWMG0.<br>(3) External Capacitor 0<br>When this pin is configured as CS0, the input function of this pin is disabled to prevent leakage current regardless of the setting of the bit 6 of register ***padier.***. |
| PA5 /<br>PRSTB /<br>PG2PWM /<br>TK2 | IO<br>ST /<br>CMOS | The functions of this pin can be:<br>(1) Bit 5 of port A. It can be configured as digital input or two-state output,with pull-high resistor.<br>(2) Hardware reset.<br>(3) Output of 11-bit PWM generator PWMG2<br>(4) Touch Key 2<br>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 /<br>AD9 /<br>CIN+ /<br>CIN1- /<br>INT1A /<br>PG1PWM<br>TK1 | IO<br>ST /<br>CMOS /<br>Analog | The functions of this pin can be:<br>(1) Bit 4 of port A. It can be configured as digital input or two-state output, with pull-high resistor by software independently<br>(2) Channel 9 of ADC analog input<br>(3) Plus input source of comparator<br>(4) Minus input source 1 of comparator<br>(5) External interrupt line 1A. It can be used as an external interrupt line 1. Both rising edge and falling edge are accepted to request interrupt service and configurable by register setting<br>(6) Output of 11-bit PWM generator PWMG1<br>(7) Touch Key 1<br>When this pin is configured as analog input, please use bit 4 of register ***padier*** to disable the digital input to prevent current leakage. The bit 4 of ***padier*** register can be set to "0" to disable digital input; wake-up from power-down by toggling this pin is also disabled. |

| Pin Name | Pin Type & Buffer Type | Description |
|---|---|---|
| PA3 /<br>AD8 /<br>CIN0- /<br>TM2PWM /<br>PG2PWM /<br>TK0 | IO<br>ST /<br>CMOS /<br>Analog | The functions of this pin can be:<br>(1) Bit 3 of port A. It can be configured as digital input or two-state output, with pull-high resistor independently by software<br>(2) Channel 8 of ADC analog input<br>(3) Minus input source 0 of comparator<br>(4) PWM output from Timer2<br>(5) Output of 11-bit PWM generator PWMG2<br>(6) Touch Key 0<br>When this pin is configured as analog input, please use bit 3 of register ***padier*** to disable the digital input to prevent current leakage. The bit 3 of ***padier*** register can be set to "0" to disable digital input; wake-up from power-down by toggling this pin is also disabled. |
| PA0 /<br>AD10 /<br>CO /<br>PG0PWM /<br>INT0 /<br>TK4 /<br>CS1 | IO<br>ST /<br>CMOS /<br>Analog | The functions of this pin can be:<br>(1) Bit 0 of port A. It can be configured as digital input or two-state output, with pull-high resistor independently by software<br>(2) Channel 10 of ADC analog input<br>(3) Output of comparator<br>(4) Output of 11-bit PWM generator PWMG0<br>(5) External interrupt line 0. It can be used as an external interrupt line 0. Both rising edge and falling edge are accepted to request interrupt service and configurable by register setting<br>(6) External Capacitor 1<br>The bit 0 of ***padier*** register can be set to "0" to disable wake-up from power-down by toggling this pin. |
| PB7 /<br>AD7 /<br>CIN5- /<br>TM3PWM /<br>PG1PWM /<br>TK9 | IO<br>ST /<br>CMOS /<br>Analog | The functions of this pin can be:<br>(1) Bit 7 of port B. It can be configured as digital input or two-state output, with pull-high resistor independently by software<br>(2) Channel 7 of ADC analog input<br>(3) Minus input source 5 of comparator<br>(4) PWM output from Timer3<br>(5) Output of 11-bit PWM generator PWMG1<br>(6) Touch Key 9<br>When this pin is configured as analog input, please use bit 7 of register ***pbdier*** to disable the digital input to prevent current leakage. The bit 7 of ***pbdier*** register can be set to "0" to disable digital input; wake-up from power-down by toggling this pin is also disabled. |

| Pin Name | Pin Type & Buffer Type | Description |
|---|---|---|
| PB6 /<br>AD6 /<br>CIN4- /<br>TM3PWM /<br>PG1PWM /<br>TK12 | IO<br>ST /<br>CMOS /<br>Analog | The functions of this pin can be:<br>(1) Bit 6 of port B. It can be configured as digital input or two-state output, with pull-high resistor independently by software<br>(2) Channel 6 of ADC analog input<br>(3) Minus input source 4 of comparator.<br>(4) PWM output from Timer3<br>(5) Output of 11-bit PWM generator PWMG1<br>(6) Touch Key 12<br>When this pin is configured as analog input, please use bit 6 of register ***pbdier*** to disable the digital input to prevent current leakage. The bit 6 of ***pbdier*** register can be set to “0” to disable digital input; wake-up from power-down by toggling this pin is also disabled. |
| PB5 /<br>AD5 /<br>TM3PWM /<br>PG0PWM /<br>INT0A /<br>TK11 | IO<br>ST /<br>CMOS /<br>Analog | The functions of this pin can be:<br>(1) Bit 5 of port B. It can be configured as digital input or two-state output, with pull-high resistor independently by software<br>(2) Channel 5 of ADC analog input<br>(3) PWM output from Timer3<br>(4) Output of 11-bit PWM generator PWMG0.<br>(5) External interrupt line 0A. It can be used as an external interrupt line 0. Both rising edge and falling edge are accepted to request interrupt service and configurable by register setting.<br>(6) Touch Key 11<br>When this pin is configured as analog input, please use bit 5 of register ***pbdier*** to disable the digital input to prevent current leakage. The bit 5 of ***pbdier*** register can be set to “0” to disable digital input; wake-up from power-down by toggling this pin is also disabled. |
| PB4 /<br>AD4 /<br>TM2PWM /<br>PG0PWM /<br>TK10 | IO<br>ST /<br>CMOS /<br>Analog | The functions of this pin can be:<br>(1) Bit 4 of port B. It can be configured as digital input or two-state output, with pull-high resistor independently by software<br>(2) Channel 4 of ADC analog input<br>(3) PWM output from Timer2<br>(4) Output of 11-bit PWM generator PWMG0.<br>(5) Touch Key 10<br>When this pin is configured as analog input, please use bit 4 of register pbdier to disable the digital input to prevent current leakage. The bit 4 of pbdier register can be set to “0” to disable digital input; wake-up from power-down by toggling this pin is also disabled. |

| Pin Name | Pin Type & Buffer Type | Description |
|---|---|---|
| PB3 /<br>AD3 /<br>PG2PWM /<br>TK5 | IO<br>ST /<br>CMOS /<br>Analog | The functions of this pin can be:<br>(1) Bit 3 of port B. It can be configured as digital input or two-state output, with pull-high resistor independently by software<br>(2) Channel 3 of ADC analog input<br>(3) Output of 11-bit PWM generator PWMG2<br>(4) Touch Key 5<br>When this pin is configured as analog input, please use bit 3 of register ***pbdier*** to disable the digital input to prevent current leakage. The bit 3 of ***pbdier*** register can be set to "0" to disable digital input; wake-up from power-down by toggling this pin is also disabled. |
| PB2 /<br>AD2 /<br>TM2PWM /<br>PG2PWM /<br>TK8 | IO<br>ST /<br>CMOS /<br>Analog | The functions of this pin can be:<br>(1) Bit 2 of port B. It can be configured as digital input or two-state output, with pull-high resistor independently by software<br>(2) Channel 2 of ADC analog input<br>(3) PWM output from Timer2<br>(4) Output of 11-bit PWM generator PWMG2<br>(5) Touch Key 8<br>When this pin is configured as analog input, please use bit 2 of register ***pbdier*** to disable the digital input to prevent current leakage. The bit 2 of ***pbdier*** register can be set to "0" to disable digital input; wake-up from power-down by toggling this pin is also disabled. |
| PB1 /<br>AD1 /<br>Vref /<br>TK6 | IO<br>ST /<br>CMOS /<br>Analog | The functions of this pin can be:<br>(1) Bit 1 of port B. It can be configured as digital input or two-state output, with pull-high resistor independently by software<br>(2) Channel 1 of ADC analog input<br>(3) External reference high voltage for ADC.<br>(4) Touch Key 6<br>When this pin is configured as analog input, please use bit 1 of register ***pbdier*** to disable the digital input to prevent current leakage. The bit 1 of ***pbdier*** register can be set to "0" to disable digital input; wake-up from power-down by toggling this pin is also disabled. |

| Pin Name | Pin Type & Buffer Type | Description |
|---|---|---|
| PB0 /<br>AD0 /<br>INT1 /<br>TK7 | IO<br>ST /<br>CMOS /<br>Analog | The functions of this pin can be:<br>(1) Bit 0 of port B. It can be configured as analog input, digital input or two-state output, with pull-high resistor independently by software.<br>(2) Channel 0 of ADC analog input.<br>(3) External interrupt line 1. It can be used as an external interrupt line 1. Both rising edge and falling edge are accepted to request interrupt service and configurable by register setting.<br>(4) Touch Key 7<br>When this pin acts as analog input, please use bit 0 of register ***pbdier*** to disable the digital input to prevent current leakage. If bit 0 of ***pbdier*** register is set to “0” to disable digital input, wake-up from power-down by toggling this pin is also disabled. |
| VDD/<br>AVDD | VDD /<br>AVDD | VDD: digital positive power<br>AVDD: Analog positive power<br>VDD is the IC power supply while AVDD is the ADC power supply. AVDD and VDD are double bonding internally and they have the same external pin. |
| GND /<br>AGND | GND /<br>AGND | GND: digital negative power<br>AGND: Analog negative power<br>GND is the IC ground pin while AGND is the ADC ground pin. AGND and GND are double bonding internally and they have the same external pin. |
| Notes: **IO**: Input/Output; **ST**: Schmitt Trigger input; **OD**: Open Drain; **Analog**: Analog input pin<br>**CMOS**: CMOS voltage level | | |

# 4. Device Characteristics

## 4.1. DC/AC Characteristics

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

| Symbol | Description | Min. | Typ | Max. | Unit | Conditions |
|---|---|---|---|---|---|---|
| $V_{DD}$ | Operating Voltage | 2.2[#] | | 5.5 | V | [#] Subject to LVR tolerance |
| LVR% | Low Voltage Reset Tolerance | -5 | | 5 | % | |
| $f_{SYS}$ | System clock (CLK)* =<br>IHRC/2<br>IHRC/4<br>IHRC/8<br>ILRC | <br>0<br>0<br>0<br> | <br><br><br><br>68K | <br>8M<br>4M<br>2M<br> | Hz | <br>$V_{DD}$ ≧ 3.5V<br>$V_{DD}$ ≧ 2.5V<br>$V_{DD}$ ≧ 2.2V<br>$V_{DD}$ =5V |
| $I_{OP}$ | Operating Current | | 0.61<br>44 | | mA<br>uA | $f_{SYS}$=IHRC/16=1MIPS@5.0V<br>$f_{SYS}$=ILRC=68KHz@5.0V |
| $I_{PD}$ | Power Down Current<br>(by *stopsys* command) | | 0.2<br>0.1<br>0.5<br>0.21 | | uA | $V_{DD}$ =5V<br>$V_{DD}$ =3V<br>$V_{DD}$ =5V, NILRC Enable<br>$V_{DD}$ =3V, NILRC Enable |
| $I_{PS}$ | Power Save Current<br>(by *stopexe* command)<br>*Disable IHRC | | 3.2<br>1.2<br>3.5<br>1.3 | | uA | $V_{DD}$ =5V<br>$V_{DD}$ =3V<br>VDD =5V, NILRC Enable<br>VDD =3V, NILRC Enable |
| $V_{IL}$ | Input low voltage for IO lines | 0 | | 0.1 $V_{DD}$ | V | |
| $V_{IH}$ | Input high voltage for IO lines | 0.7 $V_{DD}$ | | $V_{DD}$ | V | |
| $I_{OL}$ | IO lines sink current (Normal)<br>PB0,PB1,PB3<br>PA6,PA7<br>Others IO<br>IO lines sink current (Strong)<br>PB0,PB1,PB3<br>PA6,PA7,PB7<br>PB4<br>Others IO | | <br>14<br>78<br>22<br><br>14<br>78<br>40<br>22 | | mA | $V_{DD}$=5.0V, $V_{OL}$=0.5V |
| $I_{OH}$ | IO lines drive current (Normal)<br>IO lines drive current (Strong)<br>PB4,PB7<br>Others IO | | 12<br><br>29<br>12 | | mA | $V_{DD}$=5.0V, $V_{OH}$=4.5V |
| $V_{IN}$ | Input voltage | -0.3 | | $V_{DD}$+0.3 | V | |
| $I_{INJ\ (PIN)}$ | Injected current on pin | | 1 | 40 | uA | $V_{DD}$ +0.3≧$V_{IN}$≧ -0.3 |
| $R_{PH}$ | Pull-high Resistance | | 63 | | KΩ | $V_{DD}$=5.0V |
| $R_{PL}$ | Pull-low Resistance | | 63 | | KΩ | $V_{DD}$=5.0V |

| Symbol | Description | Min. | Typ | Max. | Unit | Conditions |
|---|---|---|---|---|---|---|
| $f_{IHRC}$ | Frequency of IHRC after calibration ***** | 15.76* | 16* | 16.24* | MHz | 25$^{o}$C, $V_{DD}$ =2.2V~5.5V |
| | | 15.20* | 16* | 16.80* | | $V_{DD}$ =2.2V~5.5V,<br>-40$^{o}$C <Ta<85$^{o}$C* |
| | | 14.60* | 16* | 17.40* | | $V_{DD}$ =2.0V~5.5V,<br>-40$^{o}$C <Ta<85$^{o}$C |
| $f_{ILRC}$ | Frequency of ILRC * | | 68 | | KHz | $V_{DD}$ = 5.0V |
| $f_{NILRC}$ | Frequency of NILRC * | | 18 | | KHz | $V_{DD}$ = 5.0V |
| $t_{INT}$ | Interrupt pulse width | 30 | | | ns | $V_{DD}$ = 5.0V |
| $t_{WDT}$ | Watchdog timeout period | | 8192 | | ILRC clock period | misc[1:0]=00 (default) |
| | | | 16384 | | | misc[1:0]=01 |
| | | | 65536 | | | misc[1:0]=10 |
| | | | 262144 | | | misc[1:0]=11 |
| $t_{WUP}$ | 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 | | | |
| $t_{SBP}$ | System boot-up period from power-on | | 45 | | ms | @ $V_{DD}$ =5V |
| $t_{RST}$ | External reset pulse width | 120 | | | us | @ $V_{DD}$ =5V |
| $V_{AD}$ | AD Input Voltage | 0 | | $V_{DD}$ | V | |
| ADrs | ADC resolution | | | 12 | bit | |
| ADcs | ADC current consumption | | 0.9<br>0.8 | | mA | @5V<br>@3V |
| ADclk | ADC clock period | | 2 | | us | 2.2V ~ 5.5V |
| $t_{ADCONV}$ | ADC conversion time<br>($T_{ADCLK}$ is the period of the selected AD conversion clock) | | 16 | | $T_{ADCLK}$ | 12-bit resolution |
| AD DNL | ADC Differential NonLinearity | | ±2* | | LSB | |
| AD INL | ADC Integral NonLinearity | | ±4* | | LSB | |
| ADos | ADC offset* | | 2 | | mV | @ $V_{DD}$ =3V |
| $V_{REFH}$ | ADC reference high voltage | | | | | @ $V_{DD}$ =5V, 25 $^{o}$C |
| | 4V | 3.90 | 4 | 4.10 | | |
| | 3V | 2.93 | 3 | 3.07 | | |
| | 2V | 1.95 | 2 | 2.05 | | |

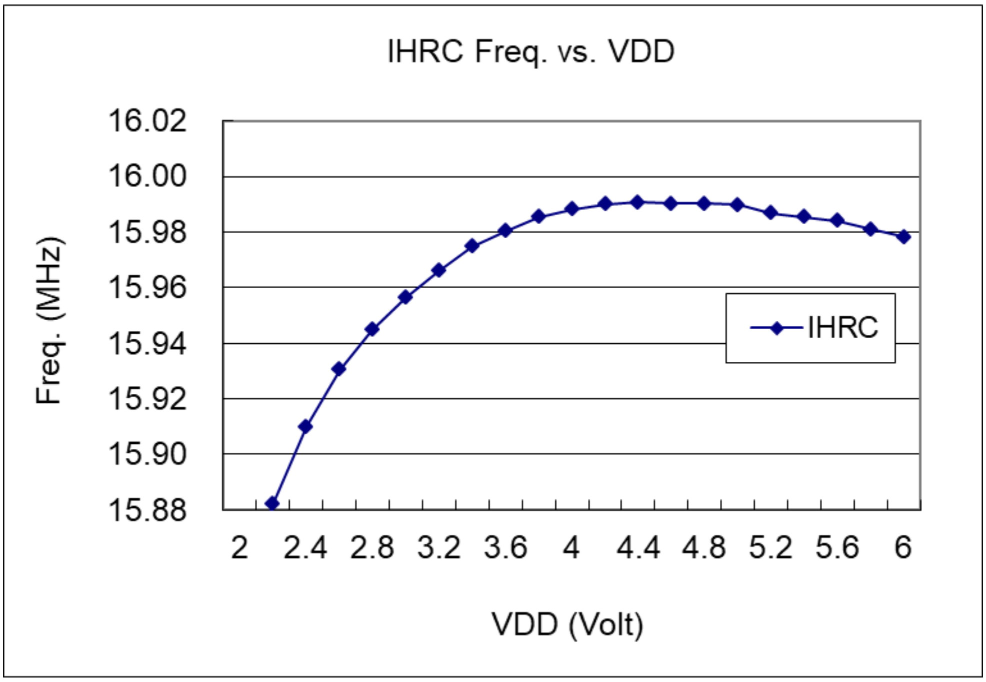
*These parameters are for design reference, not tested for every chip.

*The characteristic diagrams are the actual measured values. Considering the influence of production drift and other factors, the data in the table are within the safety range of the actual measured values.

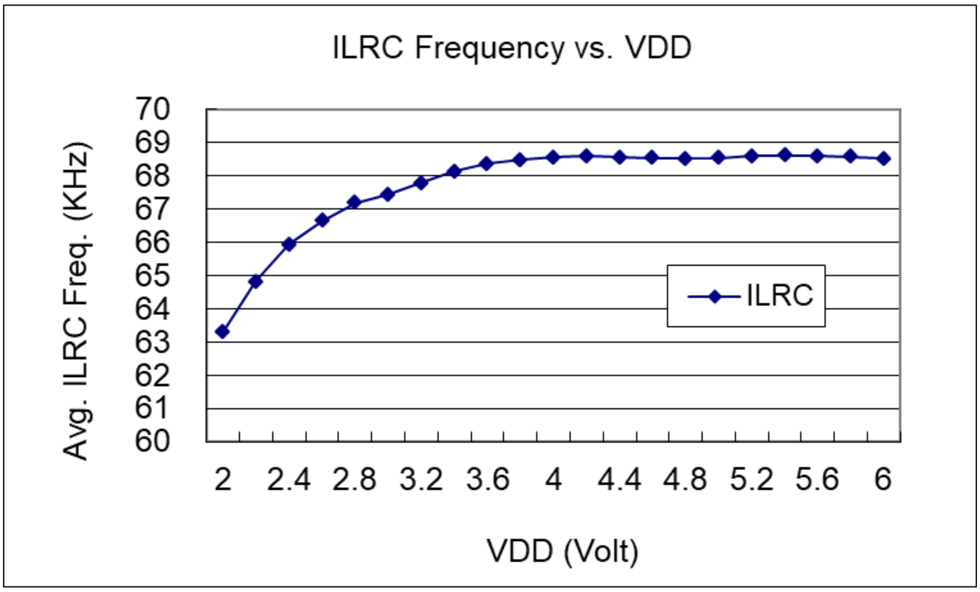
### 4.2. Absolute Maximum Ratings

- Supply Voltage .......................................... 2.2V ~ 5.5V (Maximum Rating: 5.5V) for Touch application
  *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

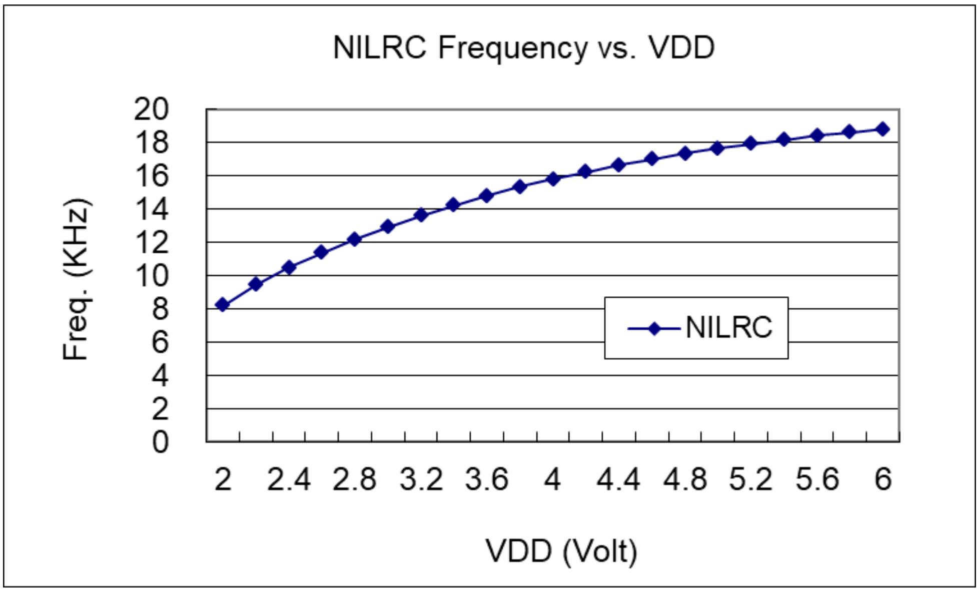
### 4.3. Typical IHRC Frequency vs. VDD (calibrated to 16MHz)



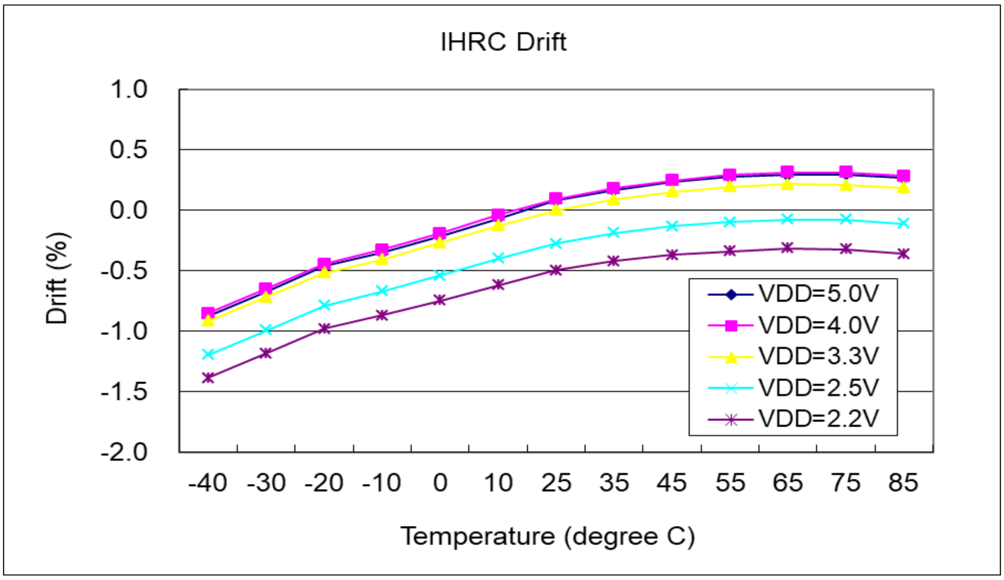
### 4.4. Typical ILRC Frequency vs. VDD



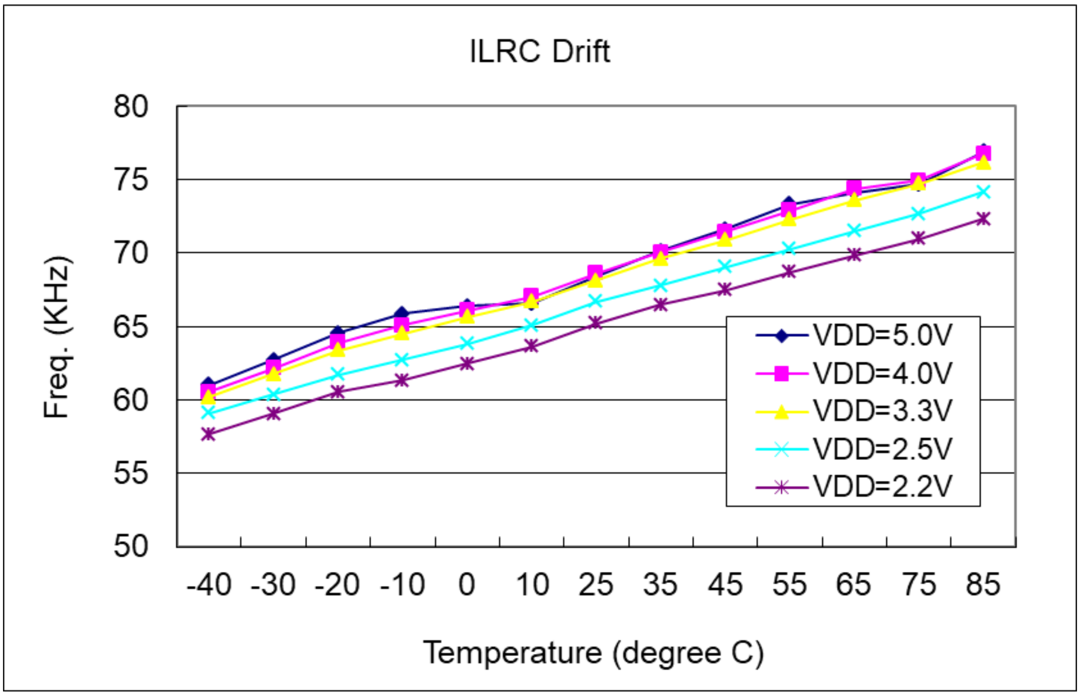
### 4.5. Typical NILRC Frequency vs. VDD



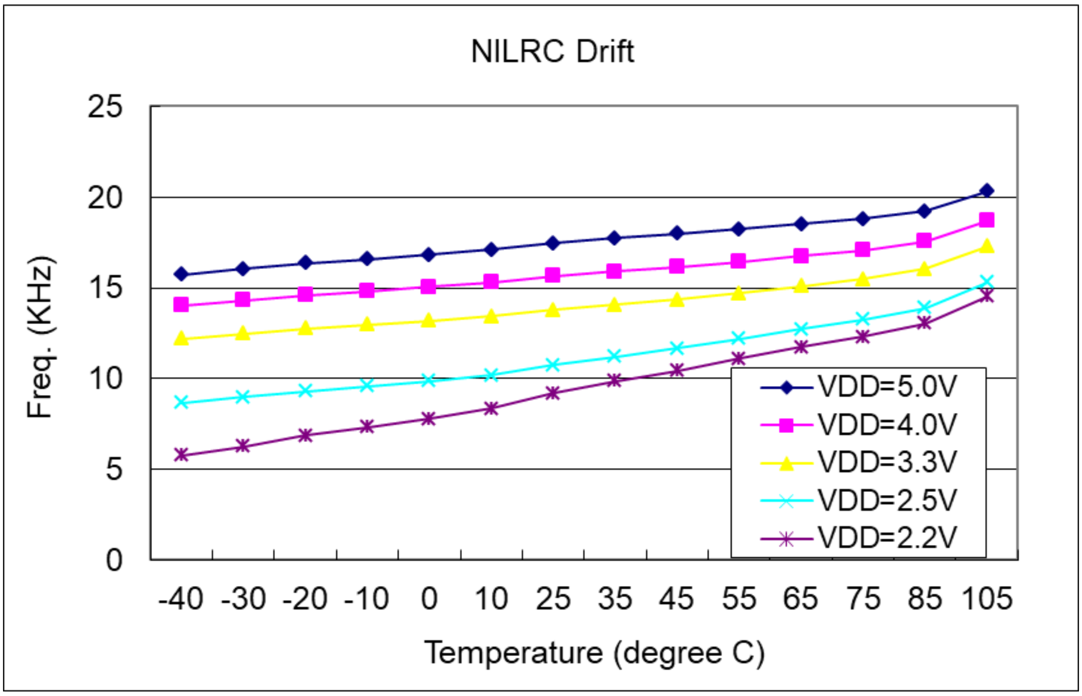
### 4.6. Typical IHRC Frequency vs. Temperature (calibrated to 16MHz)



## 4.7. Typical ILRC Frequency vs. Temperature



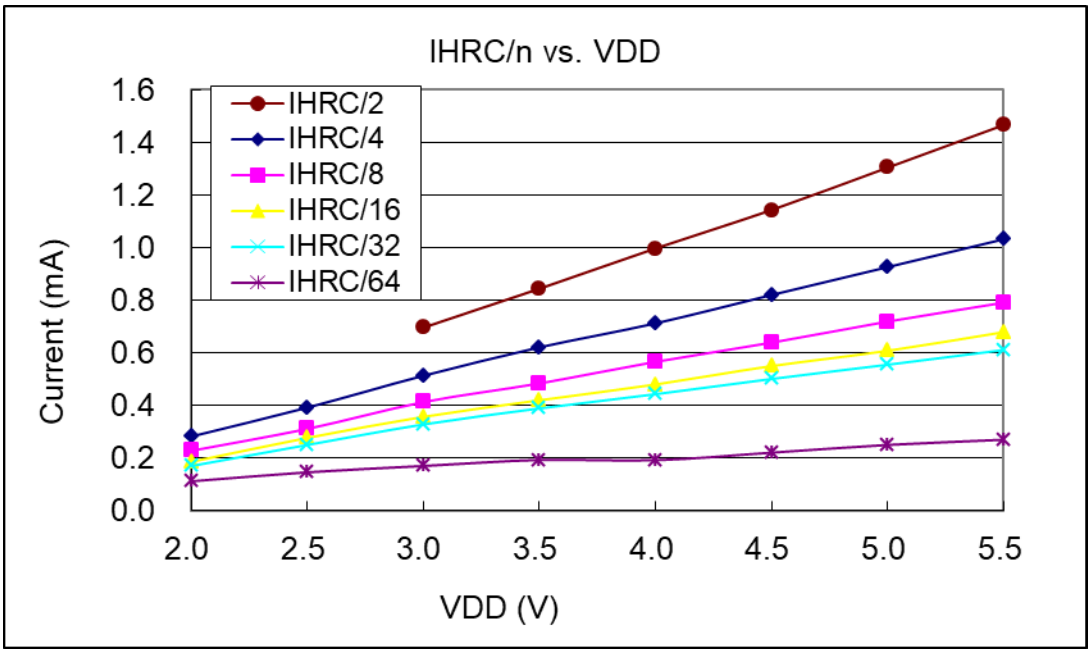
## 4.8. Typical NILRC Frequency vs. Temperature



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

- Conditions:

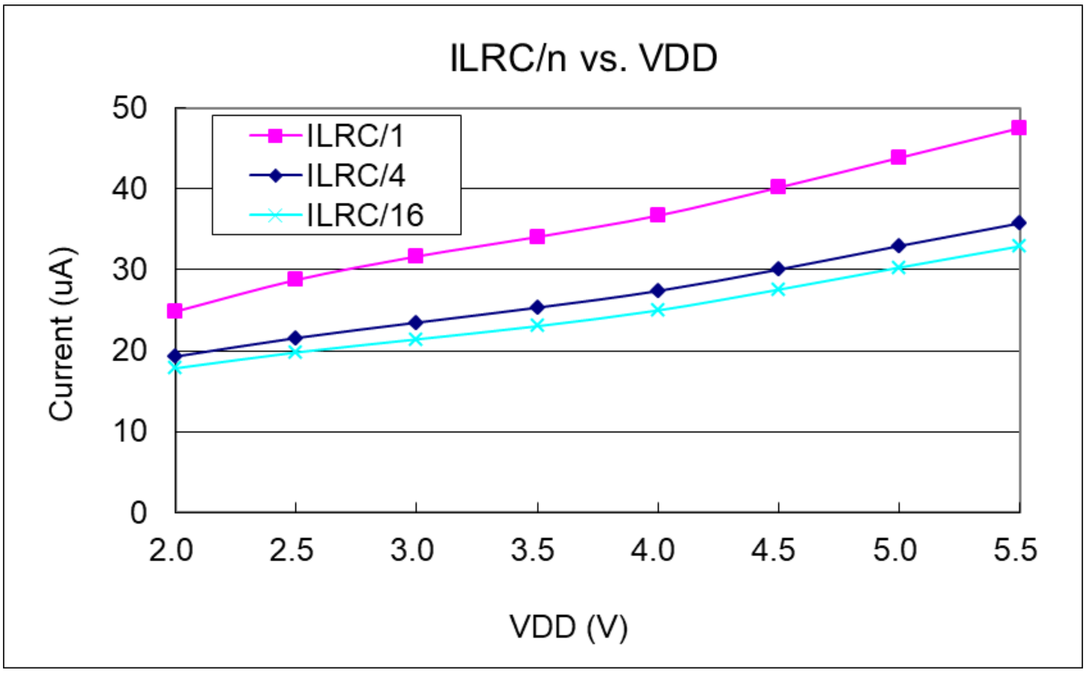
tog pa0(1s), **ON**: Bandgap, LVR, IHRC
no t16m, no interrupt, no floating IO pins, disable ILRC, **Touch**: Disable



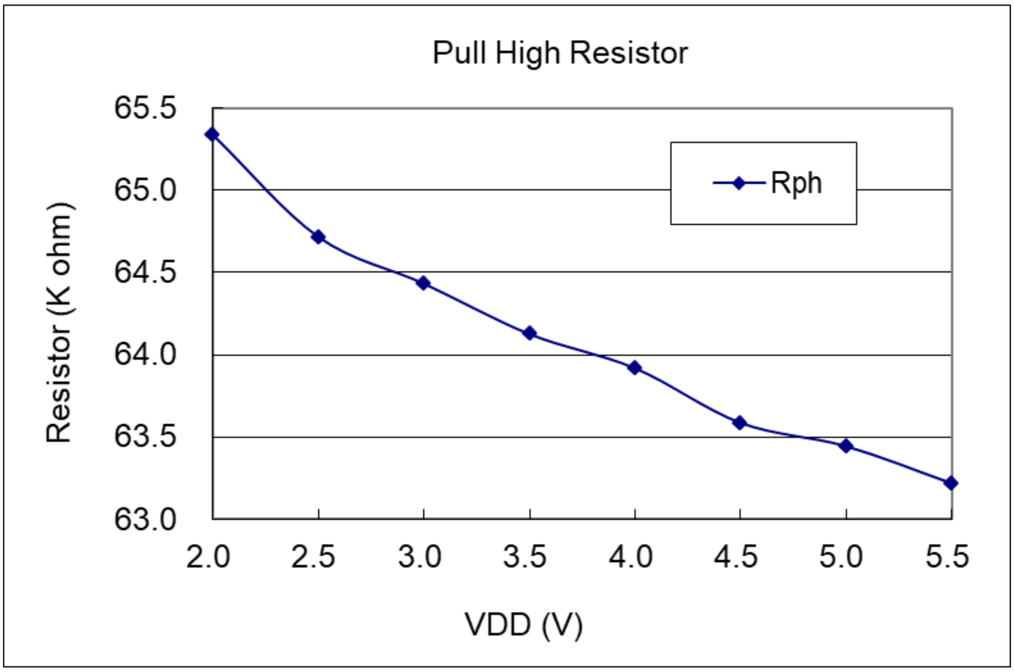
### 4.10. Typical Operating Current vs. VDD and CLK=ILRC/n

- Conditions:

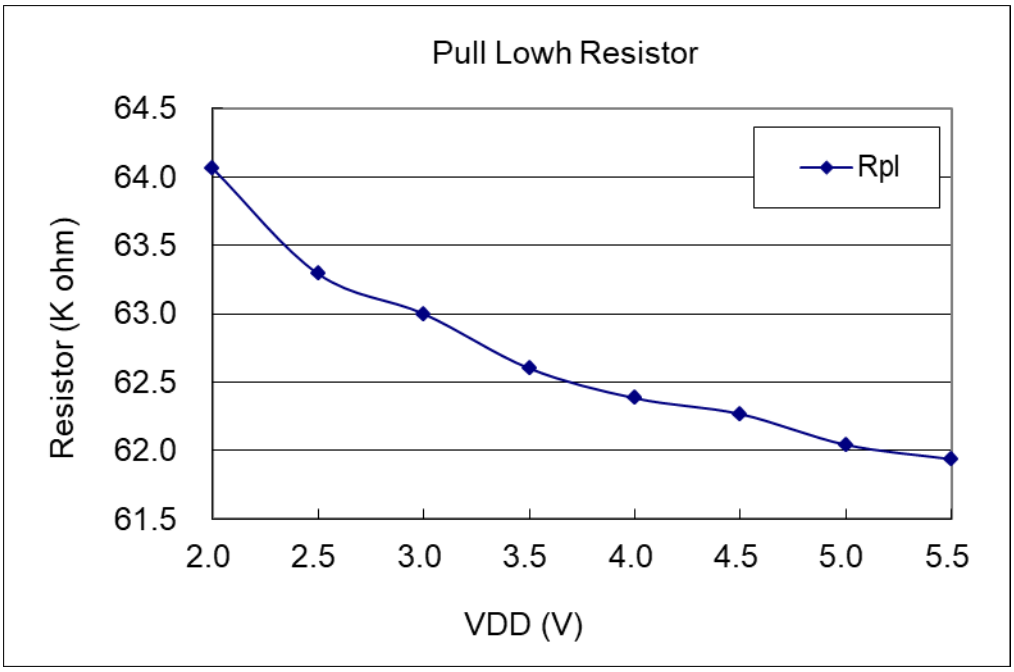
tog pa0(1s), **ON**: Bandgap, LVR, IHRC
no t16m, no interrupt, no floating IO pins, disable ILRC, **Touch**: Disable



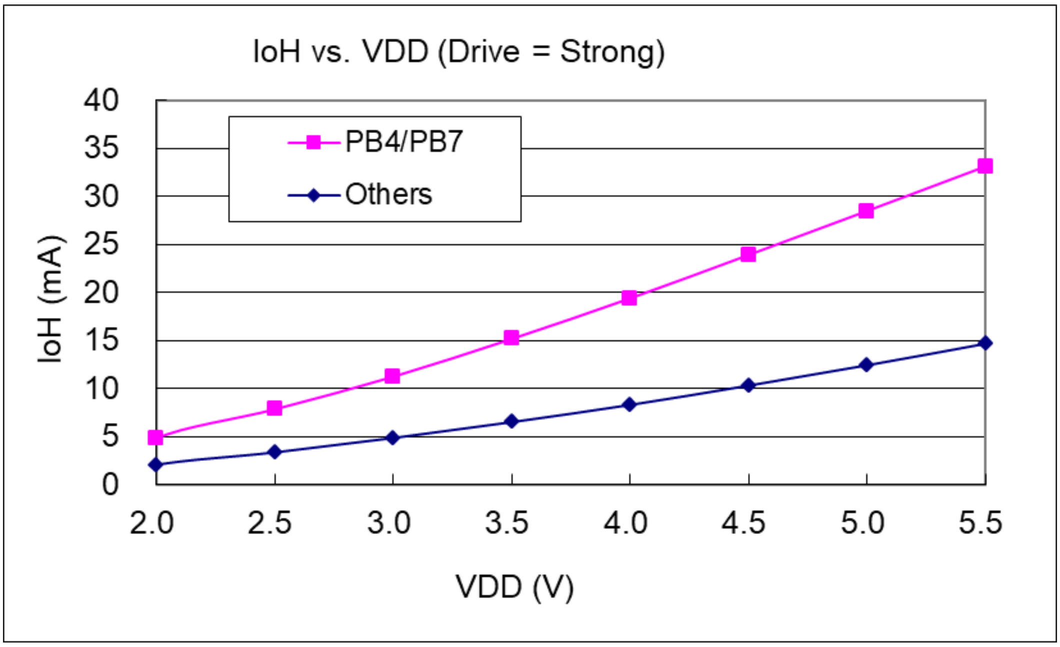
### 4.11. Typical IO pull high resistance

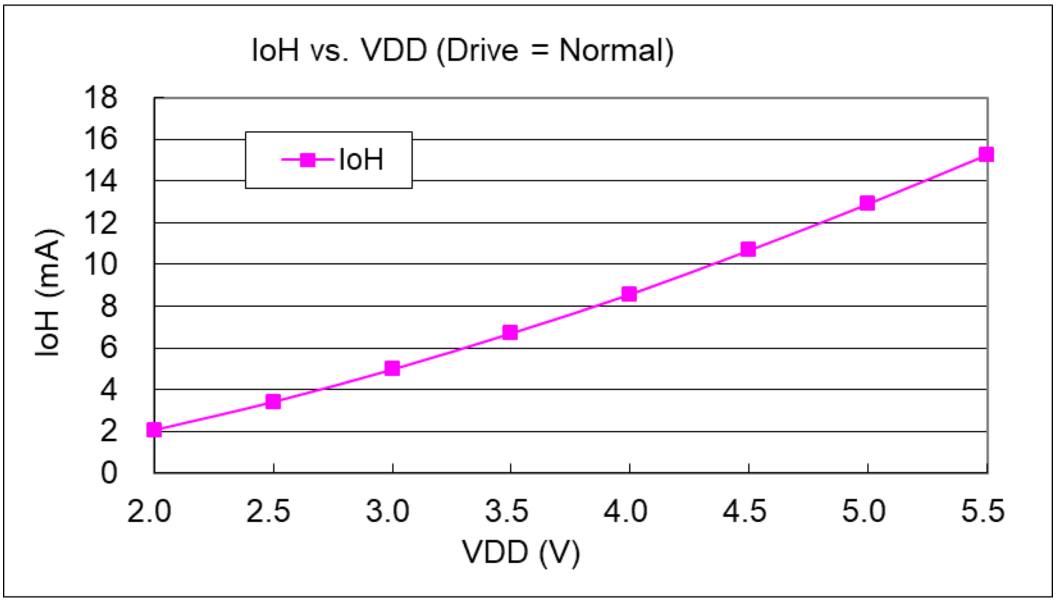


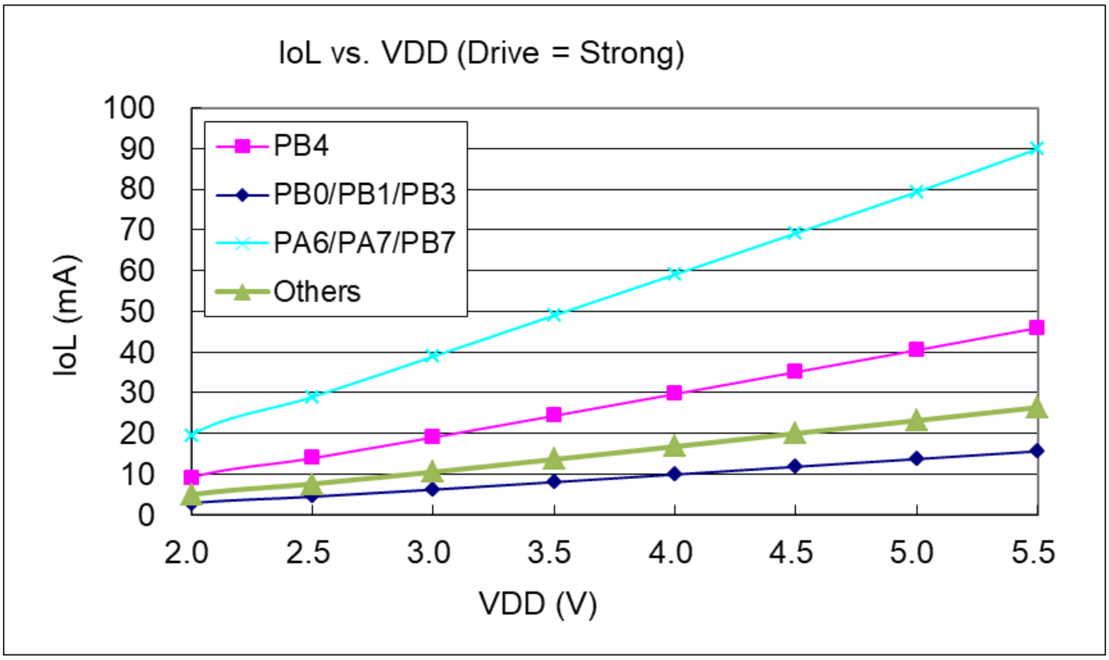
### 4.12. Typical IO pull low resistance

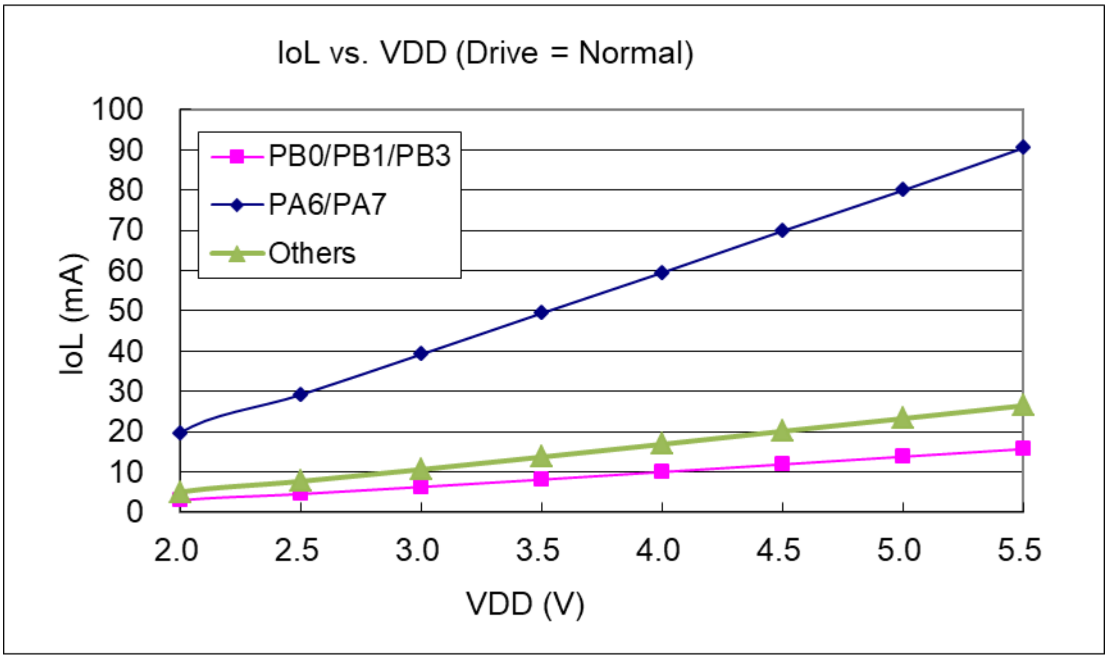


## 4.13. Typical IO driving current ($I_{OH}$) and sink current ($I_{OL}$) （VOH=0.9*VDD, VOL=0.1*VDD）

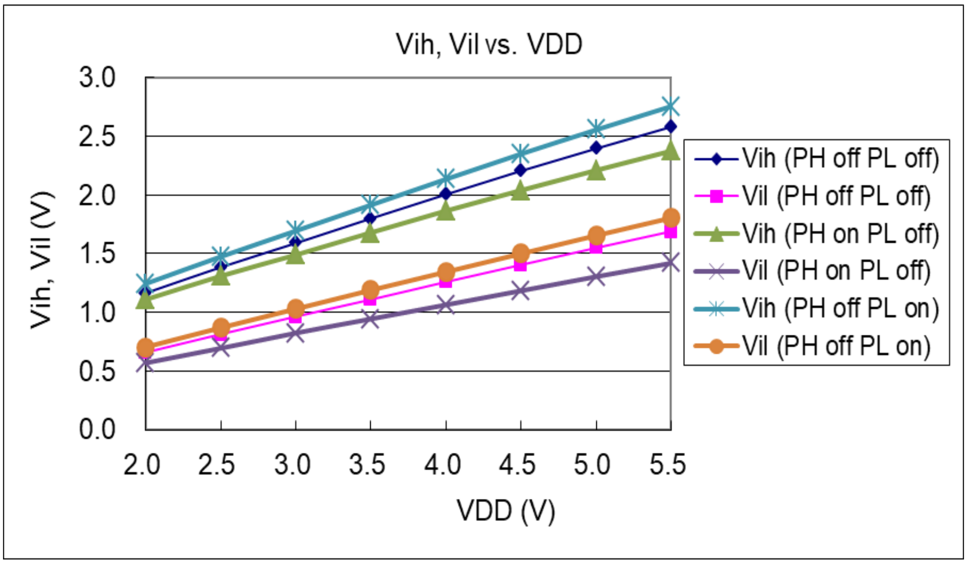




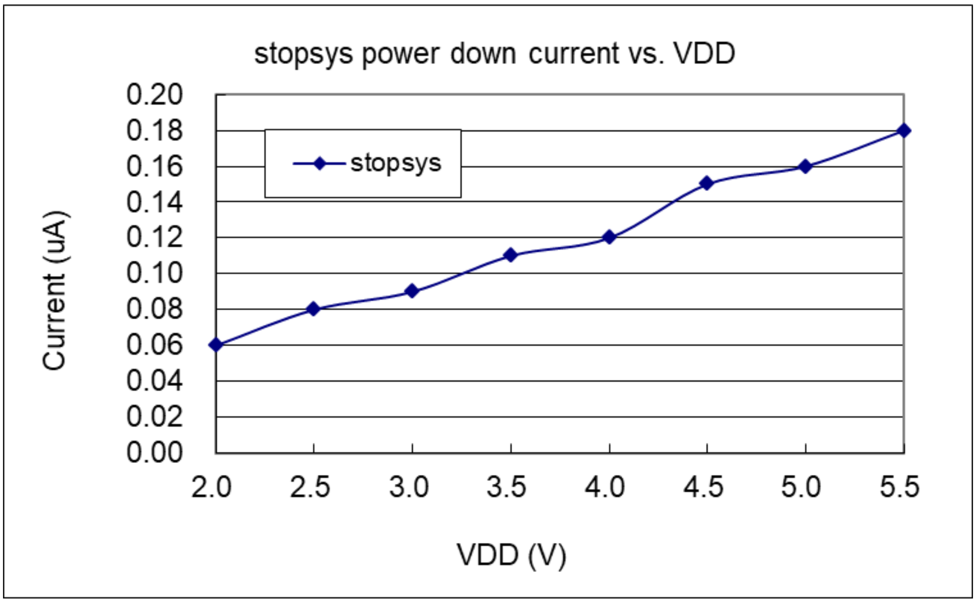

IoL vs. VDD (Drive = Strong)
IoL (mA)
100
90
80
70
60
50
40
30
20
10
0
PB4
PB0/PB1/PB3
PA6/PA7/PB7
Others
2.0
2.5
3.0
3.5
4.0
4.5
5.0
5.5
VDD (V)

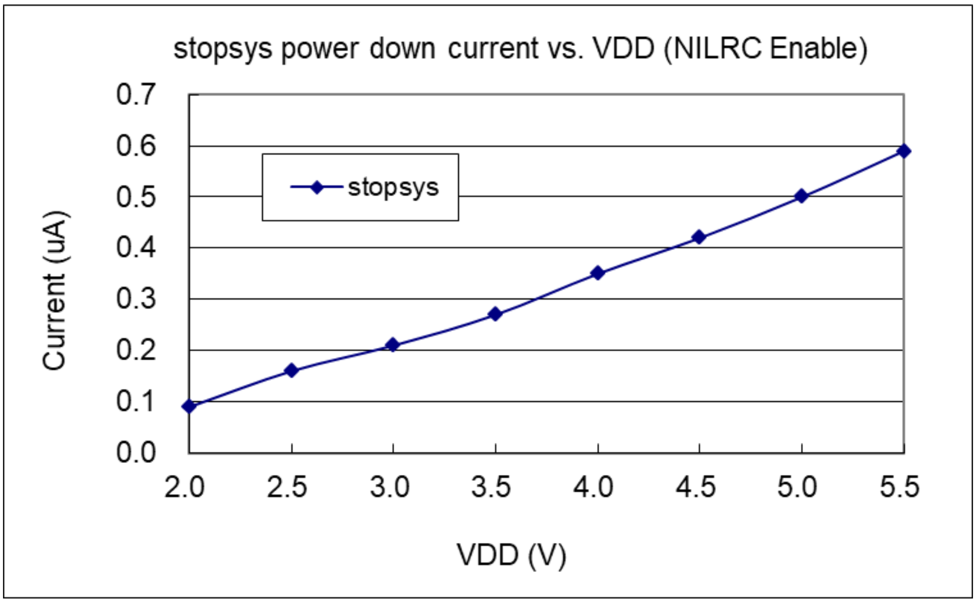

IoL vs. VDD (Drive = Normal)
IoL (mA)
100
90
80
70
60
50
40
30
20
10
0
PB0/PB1/PB3
PA6/PA7
Others
2.0
2.5
3.0
3.5
4.0
4.5
5.0
5.5
VDD (V)

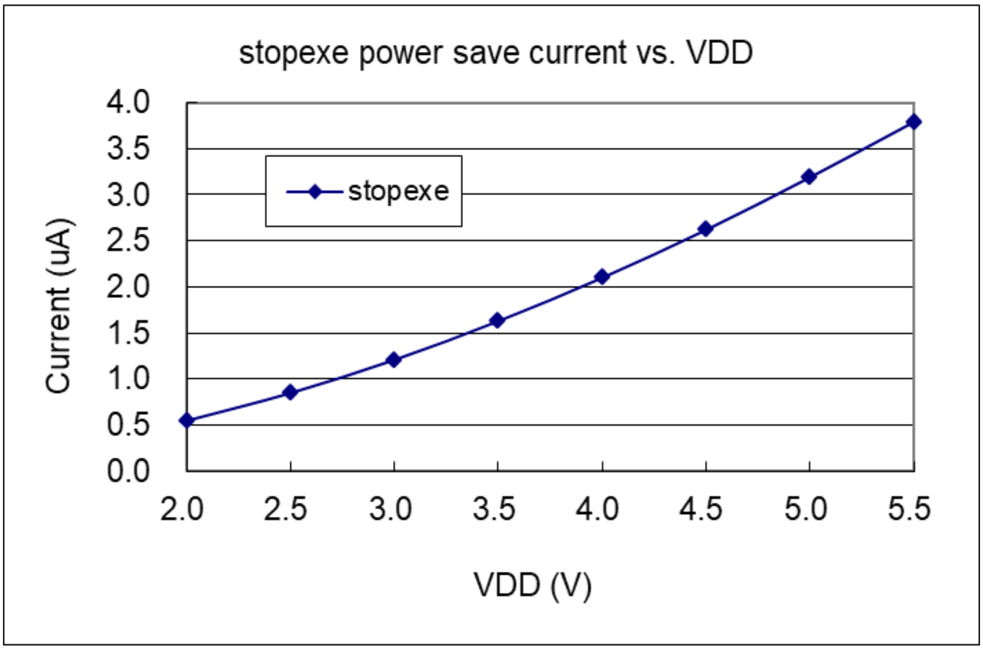
### 4.14. Typical IO input high/ low threshold voltage ($V_{IH}$/ $V_{IL}$)



### 4.15. Typical power down current ($I_{PD}$) and power save current ($I_{PS}$)




stopsys power down current vs. VDD (NILRC Enable)
Current (uA)
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
stopsys
2.0
2.5
3.0
3.5
4.0
4.5
5.0
5.5
VDD (V)


stopexe power save current vs. VDD
Current (uA)
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
stopexe
2.0
2.5
3.0
3.5
4.0
4.5
5.0
5.5
VDD (V)

# 5. Functional Description

## 5.1. Program Memory – OTP

The OTP (One Time Programmable) program memory is used to store the program instructions to be executed. The OTP program memory may contains the data, tables and interrupt entry. After reset, the initial address for FPP0 is 0x000. The interrupt entry is 0x010 if used. The OTP program memory for PMS163 is a 2.5KW that is partitioned as Table 1. The OTP memory from address 0x9F0 to 0x9FF is for system using, address space from 0x001 to 0x00F and from 0x011 to 0x9EF is user program space.

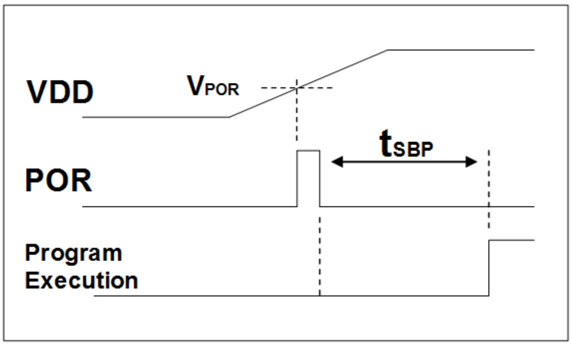
| Address | Function |
|---|---|
| 0x000 | FPP0 reset – goto instruction |
| 0x001 | User program |
| • | • |
| • | • |
| 0x00F | User program |
| 0x010 | Interrupt entry address |
| 0x011 | User program |
| • | • |
| 0x9EF | User program |
| 0x9F0 | System Using |
| • | • |
| 0x9FF | System Using |

Table 1: Program Memory Organization

## 5.2. Boot Up

POR (Power-On-Reset) is used to reset PMS163 when power up. The boot up time is 2900 ILRC clock cycles. 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. 2 and $t_{SBP}$ is the boot up time.

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



**Boot up from Power-On Reset**

Fig. 2: Power Up Sequence

## 5.3. Data Memory – SRAM

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

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

For indirect memory access mechanism, the data memory is used as the data pointer to address the data byte. All the data memory could be the data pointer; it's quite flexible and useful to do the indirect memory access. All the 160 bytes data memory of PMS163 can be accessed by indirect access mechanism.

## 5.4. Oscillator and clock

There are two oscillator circuits provided by PMS163: internal high RC oscillator (IHRC) and internal low RC oscillator (ILRC), and these two oscillators are enabled or disabled by registers clkmd.4 and clkmd.2 independently. User can choose one of these two oscillators as system clock source and use ***clkmd*** register to target the desired frequency as system clock to meet different application.

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

Table 2: Oscillator Module

### 5.4.1 Internal High RC oscillator and Internal Low RC oscillator

After boot-up, only the 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, PMS163 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**=14 ~ 18; In order to calibrate the chip to different frequency, 16MHz is the usually one.
**p3**=2.3 ~ 5.5; In order to calibrate the chip under different supply voltage.

### 5.4.3 IHRC Frequency Calibration and System Clock

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

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

Table 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 PMS163 for different option:

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

After boot up, CLKMD = 0x34:

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

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

After boot up, CLKMD = 0x14:

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

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

After boot up, CLKMD = 0x3C:

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

(4) .ADJUST_IC SYSCLK=IHRC/16, IHRC=16MHz, $V_{DD}$=2.3V

After boot up, CLKMD = 0x1C:

- ◆ IHRC frequency is calibrated to 16MHz@$V_{DD}$=2.3V and IHRC module is enabled
- ◆ System CLK = IHRC/16 = 1MHz
- ◆ Watchdog timer is disabled, ILRC is enabled, PA5 is in input mode

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

After boot up, CLKMD = 0x7C:

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

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

After boot up, CLKMD = 0XE4:

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

(7) .ADJUST_IC DISABLE

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

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

### 5.4.4 System Clock and LVR levels

The clock source of system clock comes from IHRC or ILRC, the hardware diagram of system clock in the PMS163 is shown as Fig. 3.

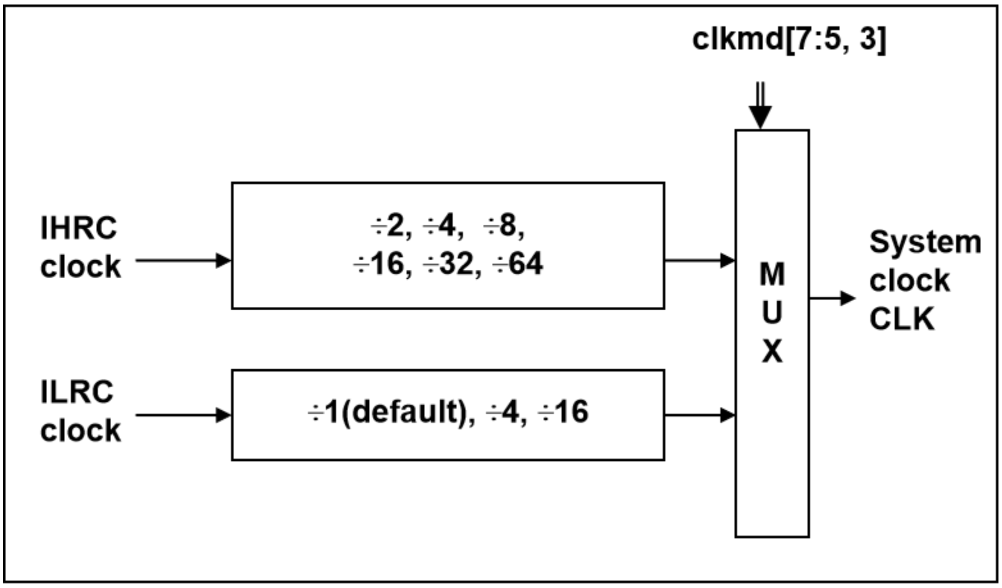


Fig. 3: Options of System Clock

User can choose different operating system clock depends on its requirement; the selected operating system clock should be combined with supply voltage and LVR level to make system stable. The LVR level will be selected during compilation, and the lowest LVR levels can be chosen for different operating frequencies. Please refer to Section 4.1.

### 5.4.5 System Clock Switching

After IHRC calibration, user may want to switch system clock to a new frequency or may switch system clock at any time to optimize the system performance and power consumption. Basically, the system clock of PMS163 can be switched among IHRC and ILRC by setting the ***clkmd*** register at any time; system clock will be the new one after writing to ***clkmd*** register immediately. Please notice that the original clock module can NOT be turned off at the same time as writing command to ***clkmd*** register. The examples are shown as below 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 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 3:** 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 4:** System may hang if it is to switch clock and turn off original oscillator at the same time

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

## 5.5. Comparator

One hardware comparator is built inside the PMS163; Fig.4 shows its hardware diagram. It can compare signals between two pins or with either internal reference voltage $V_{internal\ R}$ or internal bandgap reference voltage. The two signals to be compared, one is the plus input and the other one is the minus input. For the minus input of comparator can be PA3, PA4, Internal bandgap 1.20 volt, 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.

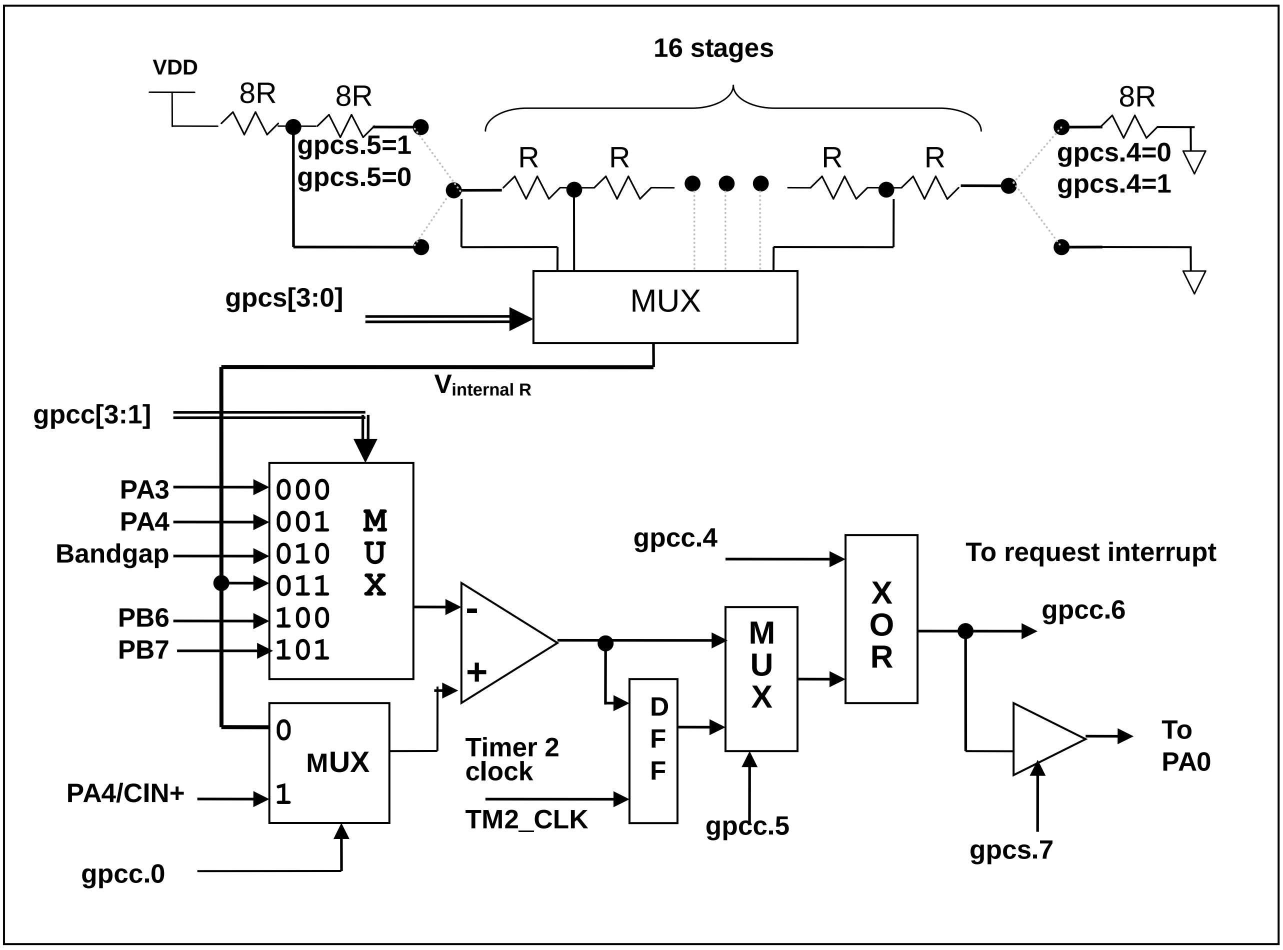
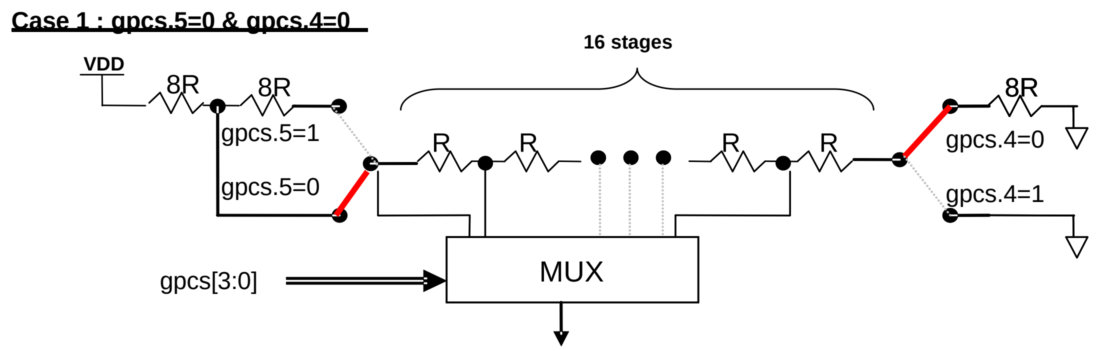


Fig.4: Hardware diagram of comparator

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

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

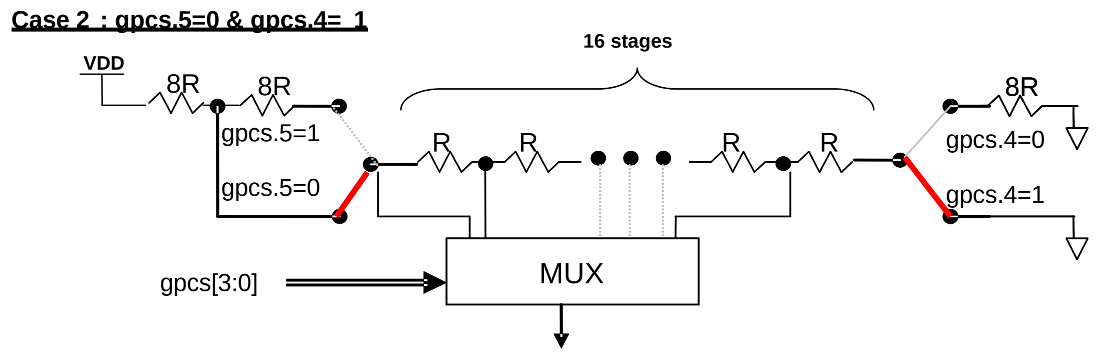


**$V_{\text{internal R}}$ = (3/4) VDD ~ (1/4) VDD + (1/32) VDD**

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

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

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

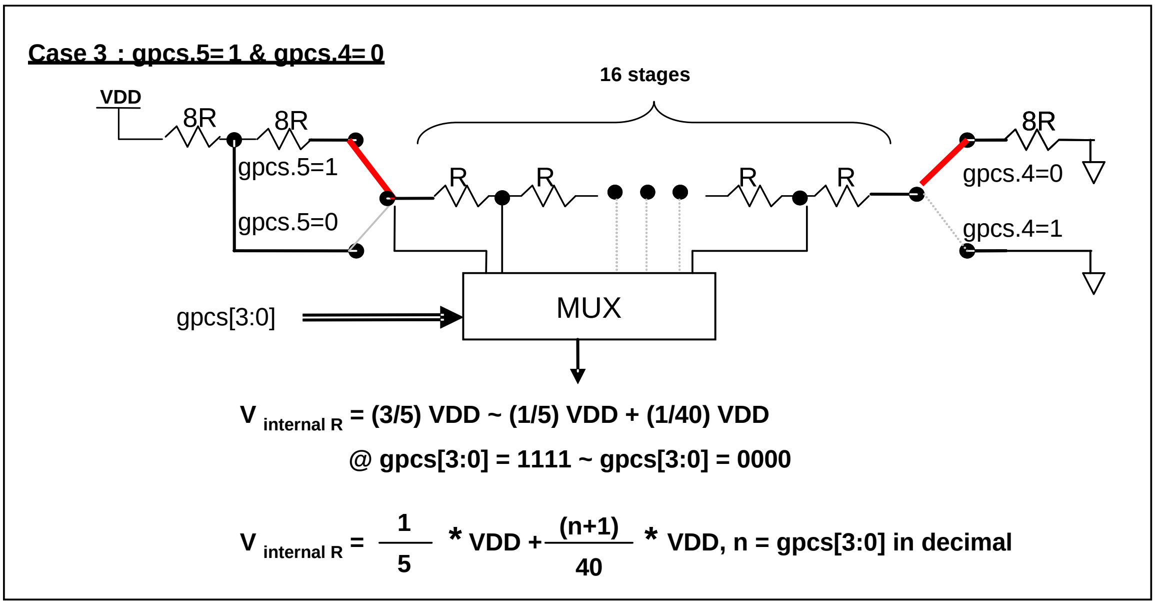


**$V_{\text{internal R}}$ = (2/3) VDD ~ (1/24) VDD**

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

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

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

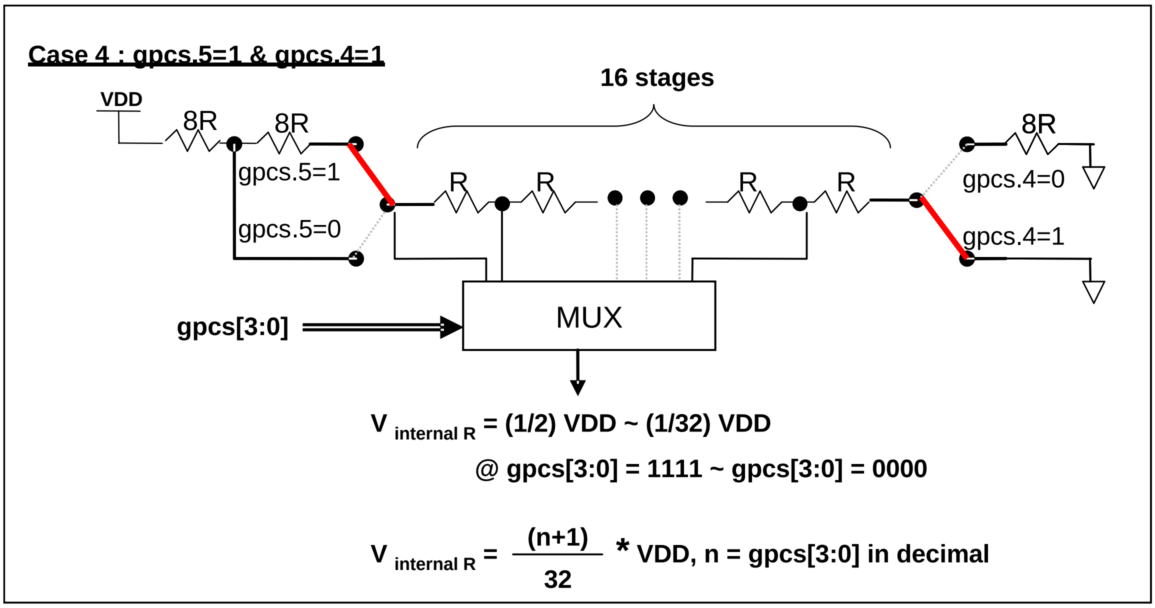


$$V_{\text{internal R}} = (3/5)\ \text{VDD} \sim (1/5)\ \text{VDD} + (1/40)\ \text{VDD}$$

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

$$V_{\text{internal R}} = \frac{1}{5} * \text{VDD} + \frac{(n+1)}{40} * \text{VDD},\ n = \text{gpcs[3:0] in decimal}$$

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



$$V_{\text{internal R}} = (1/2)\ \text{VDD} \sim (1/32)\ \text{VDD}$$

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

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

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

### 5.5.2 Using the comparator

Case1:

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

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

*or*

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

Case 2:

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

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

*or*

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

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.

### 5.5.3 Using the comparator and Bandgap 1.20V

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

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

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

*Case 1:*

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

## 5.6. 16-bit Timer (Timer16)

PMS163 provide a 16-bit hardware timer (Timer16) and its clock source may come from system clock (CLK), internal high RC oscillator (IHRC), internal low RC oscillator (ILRC), PA0 or PA4. Before sending clock to the 16-bit counter, a pre-scaling logic with divided-by-1, 4, 16 or 64 is selectable for wide range counting. The 16-bit counter performs up-counting operation only, the counter initial values can be stored from data memory by issuing the ***stt16*** instruction and the counting values can be loaded to data memory by issuing the ***ldt16*** instruction. The interrupt request from Timer16 will be triggered by the selected bit which comes from bit[15:8] of this 16-bit counter, rising edge or falling edge can be optional chosen by register ***integs.4***. The hardware diagram of Timer16 is shown as Fig. 9.

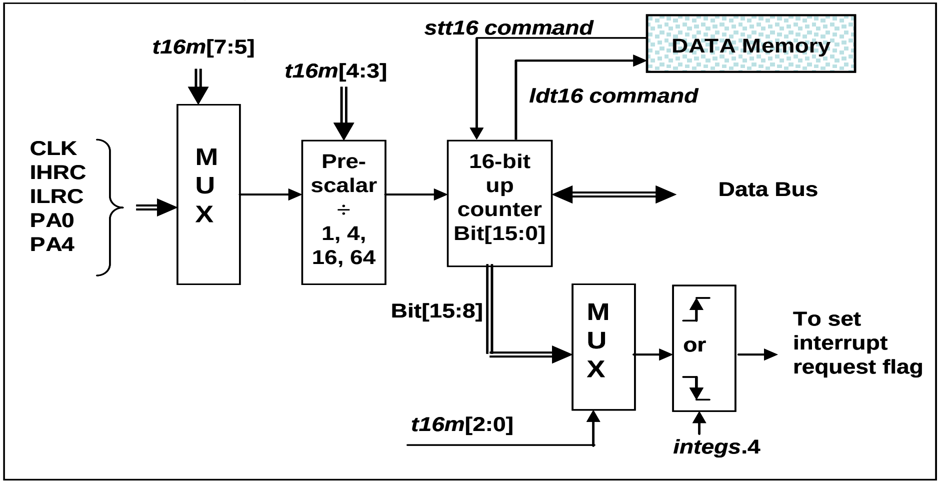


Fig. 9: Hardware diagram of Timer16

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

```
T16M    IO_RW    0x06
 $ 7~5:     STOP, SYSCLK, X, PA4_F, IHRC, X, ILRC, PA0_F          // 1st par.
 $ 4~3:     /1, /4, /16, /64                                      // 2nd par.
 $ 2~0:     BIT8, BIT9, BIT10, BIT11, BIT12, BIT13, BIT14, BIT15  // 3rd par.
```

User can choose the proper parameters of T16M to meet system requirement, examples as below:

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

$    T16M    PA0, /1, BIT8;
     // choose PA0 as clock source, every 2^9 to generate INTRQ.2=1
     // receiving every 512 times PA0 to generate INTRQ.2=1

$    T16M    STOP;
     // stop Timer16 counting
```

## 5.7. Watchdog Timer

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

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

The frequency of ILRC may drift a lot due to the variation of manufacture, supply voltage and temperature; user should reserve guard band for safe operation. Besides, the watchdog period will also be shorter than expected after Reset or Wakeup events. It is suggested to clear WDT by wdreset command after these events to ensure enough clock periods before WDT timeout.

When WDT is timeout, PMS163 will be reset to restart the program execution. The relative timing diagram of watchdog timer is shown as Fig.10.

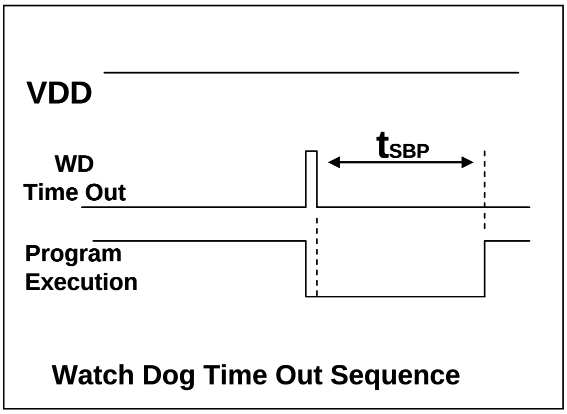


Fig. 10: Sequence of Watch Dog Time Out

## 5.8. Interrupt

There are ten interrupt lines for PMS163:

- External interrupt PA0 / PB5
- External interrupt PB0 / PA4
- ADC interrupt
- Timer16 interrupt
- Timer2 interrupt
- Timer3 / PWMG2 interrupt
- GPC / PWMG1 interrupt
- PWMG0 interrupt
- Two touch key interrupts (TK_OV and TK_END)

Every interrupt request line has its own corresponding interrupt control bit to enable or disable it; the hardware diagram of interrupt function is shown as Fig. 11. All the interrupt request flags are set by hardware and cleared by writing ***intrq*** register. When the request flags are set, it can be rising edge, falling edge or both, depending on the setting of register ***integs***. All the interrupt request lines are also controlled by ***engint*** instruction (enable global interrupt) to enable interrupt operation and ***disgint*** instruction (disable global interrupt) to disable it. The stack memory for interrupt is shared with data memory and its address is specified by stack register ***sp.*** Since the program counter is 16 bits width, the bit 0 of stack register ***sp*** should be kept 0. Moreover, user can use ***pushaf*** / ***popaf*** instructions to store or restore the values of ***ACC*** and ***flag*** register ***to*** / ***from*** stack memory.

Since the stack memory is shared with data memory, user should manipulate the memory using carefully. By adjusting the memory location of stack point, the depth of stack pointer could be fully specified by user to achieve maximum flexibility of system.

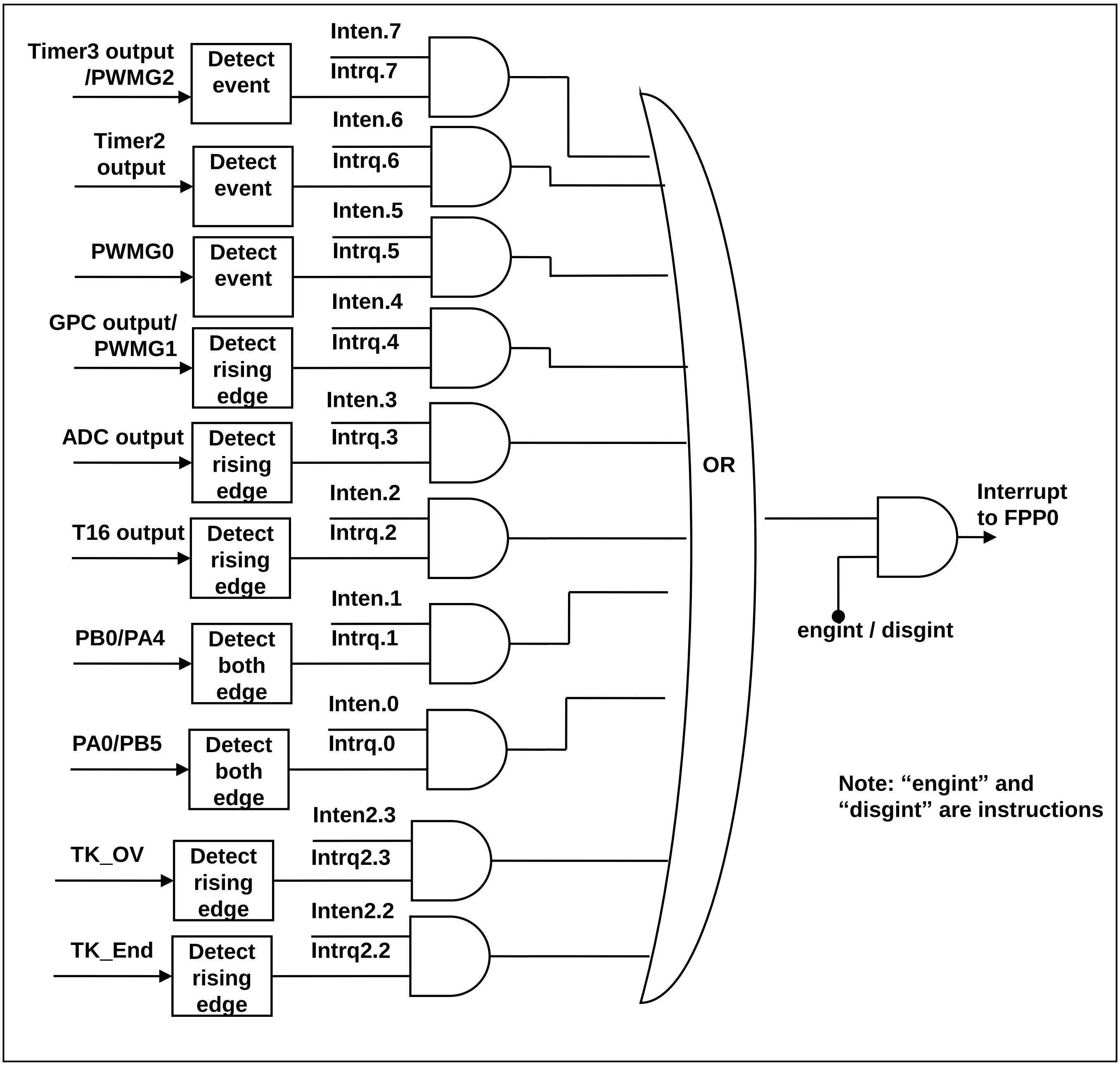


Fig. 11: Hardware diagram of Interrupt controller

Once the interrupt occurs, its operation will be:

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

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

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

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

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

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

```
void          FPPA0    (void)
{
    ...
    $   INTEN   PA0;           // INTEN =1; interrupt request when PA0 level changed
    INTRQ   =   0;             // clear INTRQ
    INTRQ2   =   0;            // clear INTRQ2
    ENGINT                     // global interrupt enable
    ...
    DISGINT                    // global interrupt disable
    ...
}
```

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

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

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

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

## 5.9. Power-Save and Power-Down

There are three operational modes defined by hardware: ON mode, Power-Save mode and Power-Down modes. ON mode is the state of normal operation with all functions ON, Power-Save mode (“***stopexe***”) is the state to reduce operating current and CPU keeps ready to continue, Power-Down mode (“***stopsys***”) is used to save power deeply. Therefore, Power-Save mode is used in the system which needs low operating power with wake-up occasionally and Power-Down mode is used in the system which needs power down deeply with seldom wake-up. Table 4 shows the differences in oscillator modules between Power-Save mode (“***stopexe***”) and Power-Down mode (“***stopsys***”).

| **Differences in oscillator modules between STOPSYS and STOPEXE** | | | |
|---|---|---|---|
| | IHRC | ILRC | NILRC |
| STOPSYS | Stop | Stop | No Change |
| STOPEXE | No Change | No Change | No Change |

Table 4: Differences in oscillator modules between STOPSYS and STOPEXE

### 5.9.1 Power-Save mode (“*stopexe*”)

Using “***stopexe***” instruction to enter the Power-Save mode, only system clock is disabled, remaining all the oscillator modules be active. For CPU, it stops executing; however, for Timer16, counter keep counting if its clock source is not the system clock. The wake-up sources for “***stopexe***” can be IO-toggle or Timer16 counts to set values when the clock source of Timer16 is IHRC or ILRC modules, TM2C/TM3C wake up with NILRC clock source which needs to set MISC2.0=1 to enable the NILRC or wake-up by comparator when setting GPCC.7=1 and GPCS.6=1 to enable the comparator wake-up function at the same time. Wake-up from input pins can be considered as a continuation of normal execution, the detail information for Power-Save mode shows below:

- IHRC oscillator modules: No change, keep active if it was enabled
- ILRC oscillator modules: must remain enabled, need to start with ILRC when be wakening up
- System clock: Disable, therefore, CPU stops execution
- OTP memory is turned off
- Timer counter: Stop counting if system clock is selected by clock source or the corresponding oscillator module is disabled; otherwise, it keeps counting. (The Timer contains TM16, TM2, TM3)
- Wake-up sources:
  a. IO toggle wake-up: IO toggling in digital input mode (*PxC* bit is 1 and *PxDIER* bit is 1).
  b. Timer wake-up: If the clock source of Timer is not the SYSCLK, the system will be awakened when the Timer counter reaches the set value.
  c. TM2C/TM3C wake up with NILRC clock source: it needs setting MISC2.0=1 to enable the NILRC, at the same time, the clock source of Timer2/Timer3 selects NILRC.
  d. Comparator wake-up: It needs setting *GPCC*.7=1 and *GPCS*.6=1 to enable the comparator wake-up function at the same time. Please note: the internal 1.20V bandgap reference voltage is not suitable for the comparator wake-up function.

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

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

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

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

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

### 5.9.2 Power-Down mode (“*stopsys*”)

Power-Down mode is the state of deeply power-saving with turning off all the oscillator modules. By using the “*stopsys*” instruction, this chip will be put on Power-Down mode directly. It is recommend to set GPCC.7=0 to disable the comparator before the command “*stopsys*”.The following shows the internal status of PMS163 in detail when “***stopsys***” command is issued:

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

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

```
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
```

### 5.9.3 Wake-up

After entering the Power-Down or Power-Save modes, the PMS163 can be resumed to normal operation by toggling IO pins or TM2C/TM3C wake up with NILRC clock source, Timer16 wake-up is available for Power-Save mode ONLY. Table 5 shows the differences in wake-up sources between STOPSYS and STOPEXE.

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

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

When using the IO pins to wake-up the PMS163, registers ***padier*** should be properly set to enable the wake-up function for every corresponding pin. The time for normal wake-up is about 3000 ILRC clocks counting from wake-up event; fast wake-up can be selected to reduce the wake-up time by *misc*.5 register, and the time for fast wake-up is 45 ILRC clocks from IO toggling.

| Suspend mode | Wake-up mode | Wake-up time ($t_{WUP}$) from IO toggle |
|---|---|---|
| STOPEXE suspend<br>or<br>STOPSYS suspend | fast wake-up | 45 * $T_{ILRC,}$<br>Where $T_{ILRC}$ is the time period of ILRC |
| STOPEXE suspend<br>or<br>STOPSYS suspend | normal wake-up | 3000 * $T_{ILRC}$,<br>Where $T_{ILRC}$ is the clock period of ILRC |

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

## 5.10. IO Pins

All the IO pins have the same structure. When PMS163 is put in power-down or power-save mode, every pin can be used to wake-up system by toggling its state. Therefore, those pins needed to wake-up system must be set to input mode and set the corresponding bits of registers ***padier*** to high**.** The same reason, ***padier***.0 should be set to high when PA0 is used as external interrupt pin.

All these pins have Schmitt-trigger input buffer and output driver with CMOS level. When it is set to output low, the pull-up resistor is turned off automatically. If user wants to read the pin state, please notice that it should be set to input mode before reading the data port; if user reads the data port when it is set to output mode, the reading data comes from data register, NOT from IO pad. As an example, Table 7 shows the configuration table of bit 0 of port A. The hardware diagram of IO buffer is also shown as Fig. 12.

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

Table 7: PA0 Configuration Table

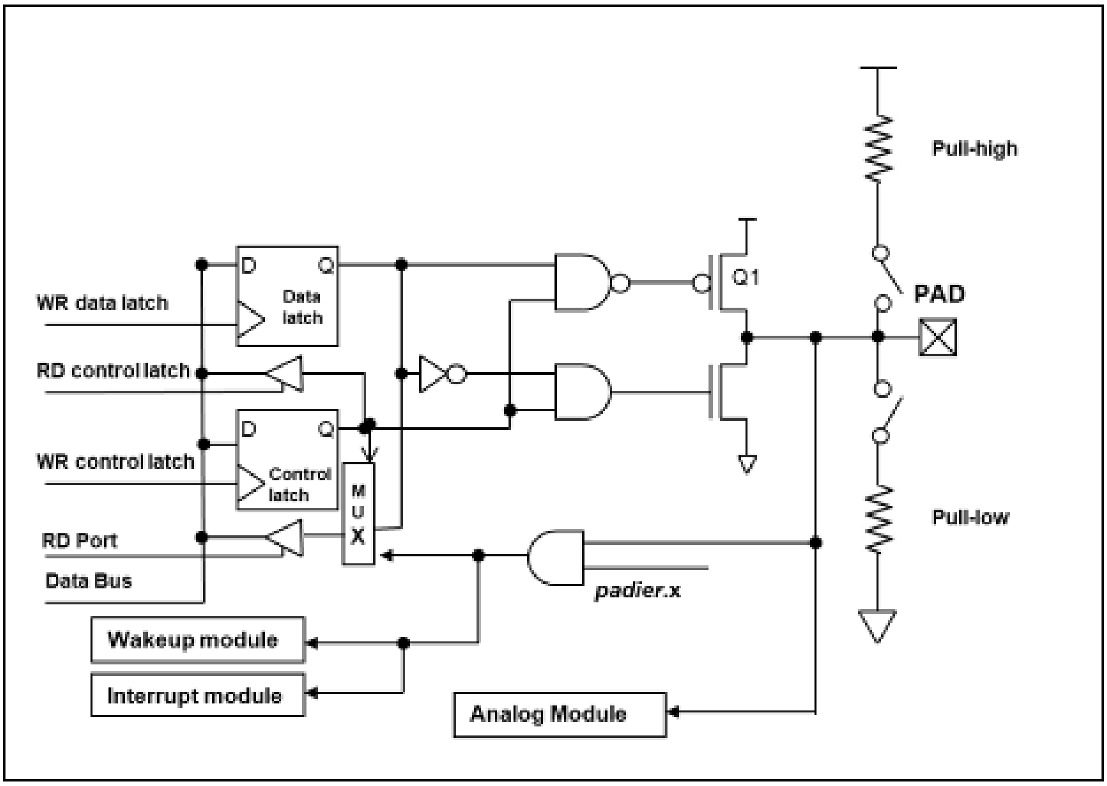


Fig. 12: Hardware diagram of IO buffer

All the IO pins have the same structure. The corresponding bits in registers ***padier*** should be set to low to prevent leakage current for those pins are selected to be analog function. When PMS163 is put in power-down or power-save mode, every pin can be used to wake-up system by toggling its state. Therefore, those pins needed to wake-up system must be set to input mode and set the corresponding bits of registers ***padier*** to high***.*** The same reason, ***padier***.0 should be set to high when PA0 is used as external interrupt pin.

## 5.11. Reset

There are many causes to reset the PMS163, once reset is asserted, all the registers in PMS163 will be set to default values, system should be restarted once abnormal cases happen, or by jumping program counter to address ’h0.

After power-up and LVR reset, the SRAM data will be kept when VDD>$V_{DR}$(SRAM data retention voltage). However, if SRAM is cleared after power-on again, the data cannot be kept. And, the data memory is in an uncertain state when VDD<$V_{DR}$.

The content will be kept when reset comes from PRSTB pin or WDT timeout.

## 5.12 8-bit Timer (Timer2/Timer3) with PWM generation

Two 8-bit hardware timers (Timer2 and Timer3) with PWM generation are implemented in the PMS163. The following descriptions thereinafter are for Timer2 only. It is because Timer3 have same structure with Timer2. Please refer to Fig.13 shown the hardware diagram of Timer2, the clock sources of Timer2 may come from system clock, internal high RC oscillator (IHRC), internal low RC oscillator (ILRC/NILRC),PA0, PB0, PA4 and comparator. Bit [7:4] of register tm2c are used to select the clock of Timer2. 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-1~31 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 to 8-bit PWM resolution, Fig.14 shows the timing diagram of Timer2 for both period mode and PWM mode.

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

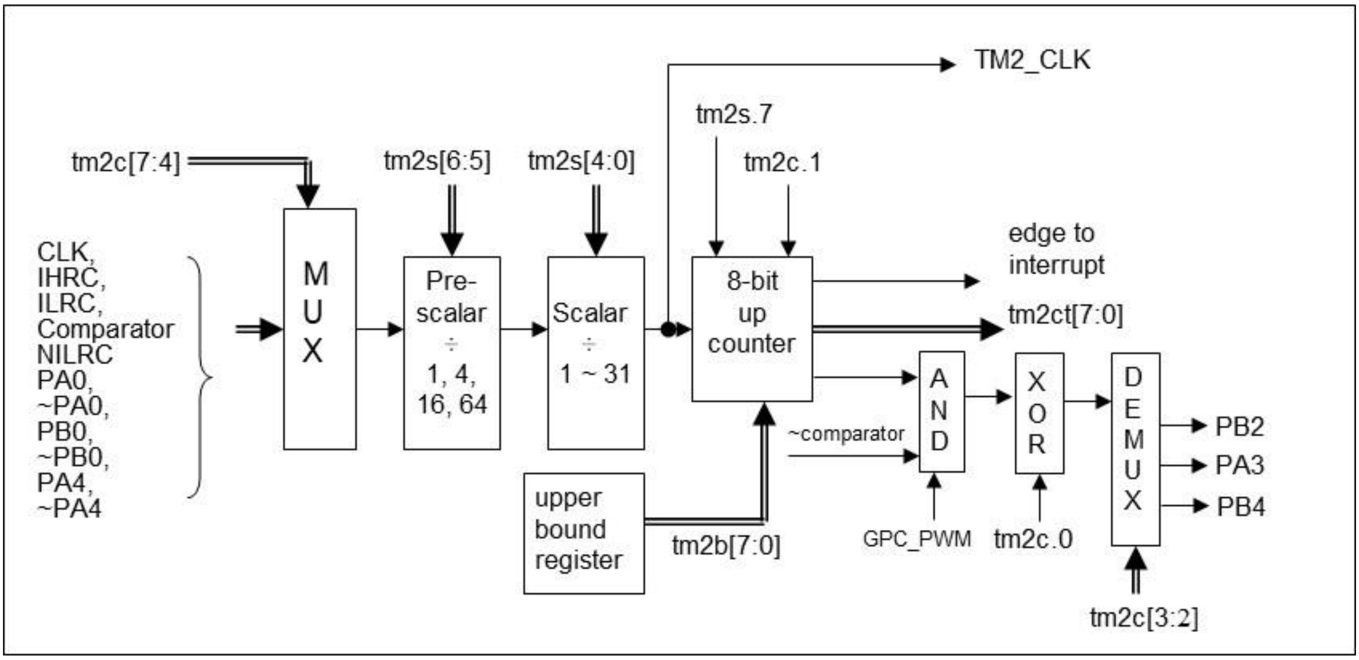


Fig.13: Timer2 hardware diagram

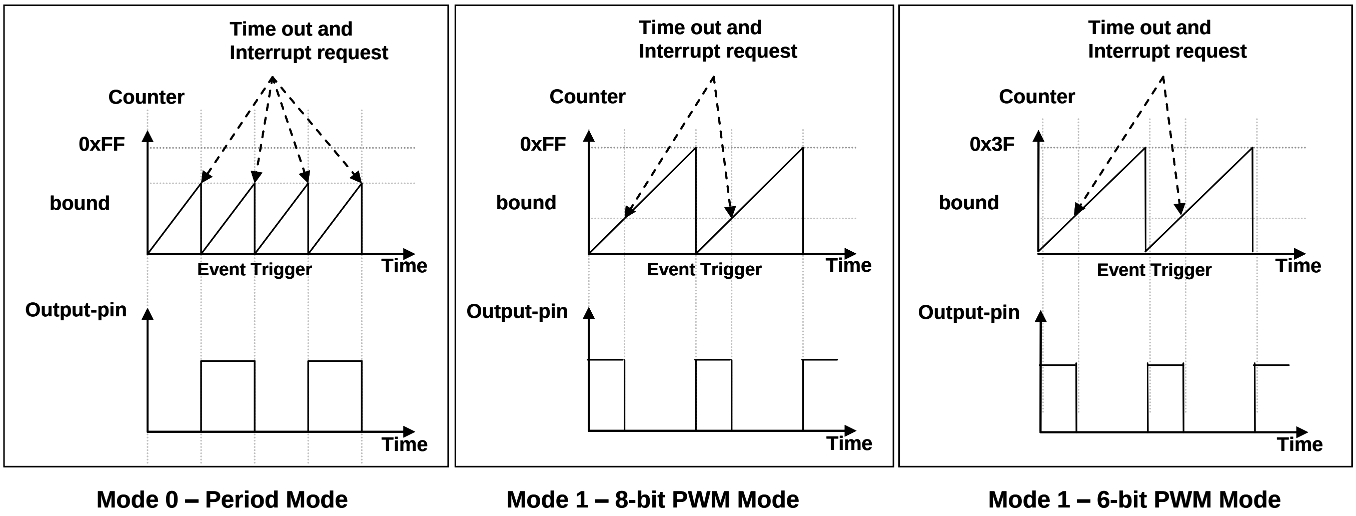


Fig.14: Timing diagram of Timer2 in period mode and PWM mode (tm2c.1=1)

A Code Option GPC_PWM is for the applications which need the generated PWM waveform to be controlled by the comparator result. If the Code Option GPC_PWM is selected, the PWM output stops while the comparator output is 1 and then the PWM output turns on while the comparator output goes back to 0, as shown in Fig. 15.

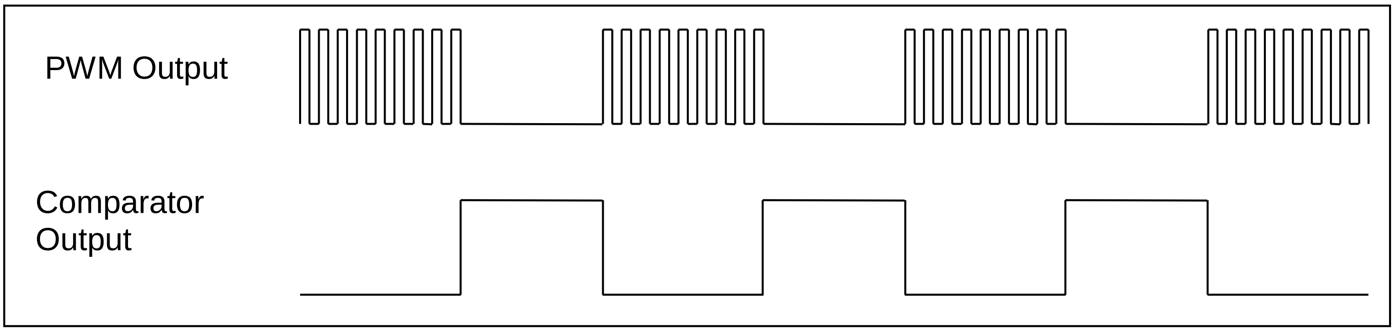


Fig.15：Comparator controls the output of PWM waveform

### 5.12.1. Using the Timer2 to generate periodical waveform

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

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

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

Example 1:

tm2c = 0b0001_1000, Y=8MHz
tm2b = 0b0111_1111, K=127
tm2s = 0b0000_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] = 0b0111_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 = 0b0000_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 = 0b0000_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 from 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;
                }
        }
```

### 5.12.2. 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 = 0b0000_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 = 0b0111_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 = 0b0000_00000, S1=1, S2=0
➔ PWM output keep high
➔ duty of output = [(255+1) ÷ 256] × 100% = 100%

Example 4:

tm2c = 0b0001_1010, Y=8MHz
tm2b = 0b0000_1001, K = 9
tm2s = 0b0000_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;
        }
    }
```

### 5.12.3. 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 = 0b1000_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 = 0b1111_11111, S1=64, S2=31
➔ frequency of output = 8MHz ÷ ( 64 × 64 × (31+1) ) = 61.03 Hz
➔ duty of output = [(31+1) ÷ 64] × 100% = 50%

Example 3:

tm2c = 0b0001_1010, Y=8MHz
tm2b = 0b0011_1111, K=63
tm2s = 0b1000_00000, S1=1, S2=0
➔ PWM output keep high
➔ duty of output = [(63+1) ÷ 64] × 100% = 100%

Example 4:

tm2c = 0b0001_1010, Y=8MHz
tm2b = 0b0000_0000, K=0
tm2s = 0b1000_00000, S1=1, S2=0
➔ frequency = 8MHz ÷ ( 64 × 1 × (0+1) ) = 125KHz
➔ duty = [(0+1) ÷ 64] × 100% =1.5%

## 5.13. 11-bit PWM Generator

Three 11-bit hardware PWM generators (PWMG0, PWMG1 & PWMG2) are implemented in the PMS163. The following descriptions thereinafter are for PWMG0 only. It is because PWMG1 & PWMG2 have the same structures and functions with PWMG0.

Their individual outputs are listed as below:

- PWMG0 – PA0, PA6, PB4, PB5
- PWMG1 – PA4, PB6, PB7
- PWMG2 – PA3, PA5, PA7, PB2, PB3

### 5.13.1. 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, $2^N \times T_{clock} = T_{Period}$).

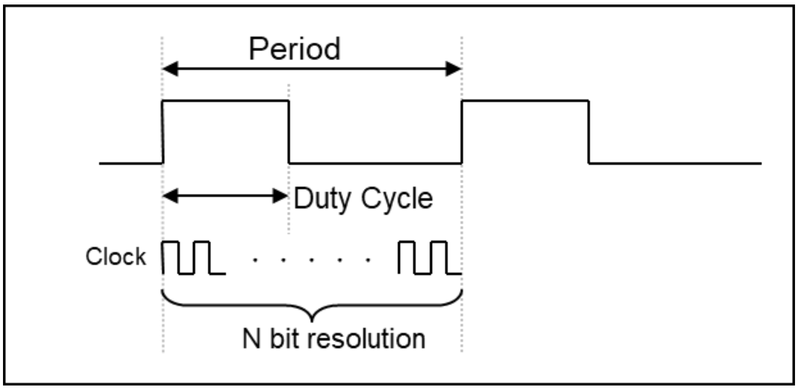


Fig.16: PWM Output Waveform

### 5.13.2. Hardware and Timing Diagram

Three 11-bit hardware PWM generators are built inside the PMS163; Fig.17 shows the hardware diagram PWMG0 as an example. 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. Users can also use the comparator result to control the output of the PWM waveform by using the GPC_PWM code option.

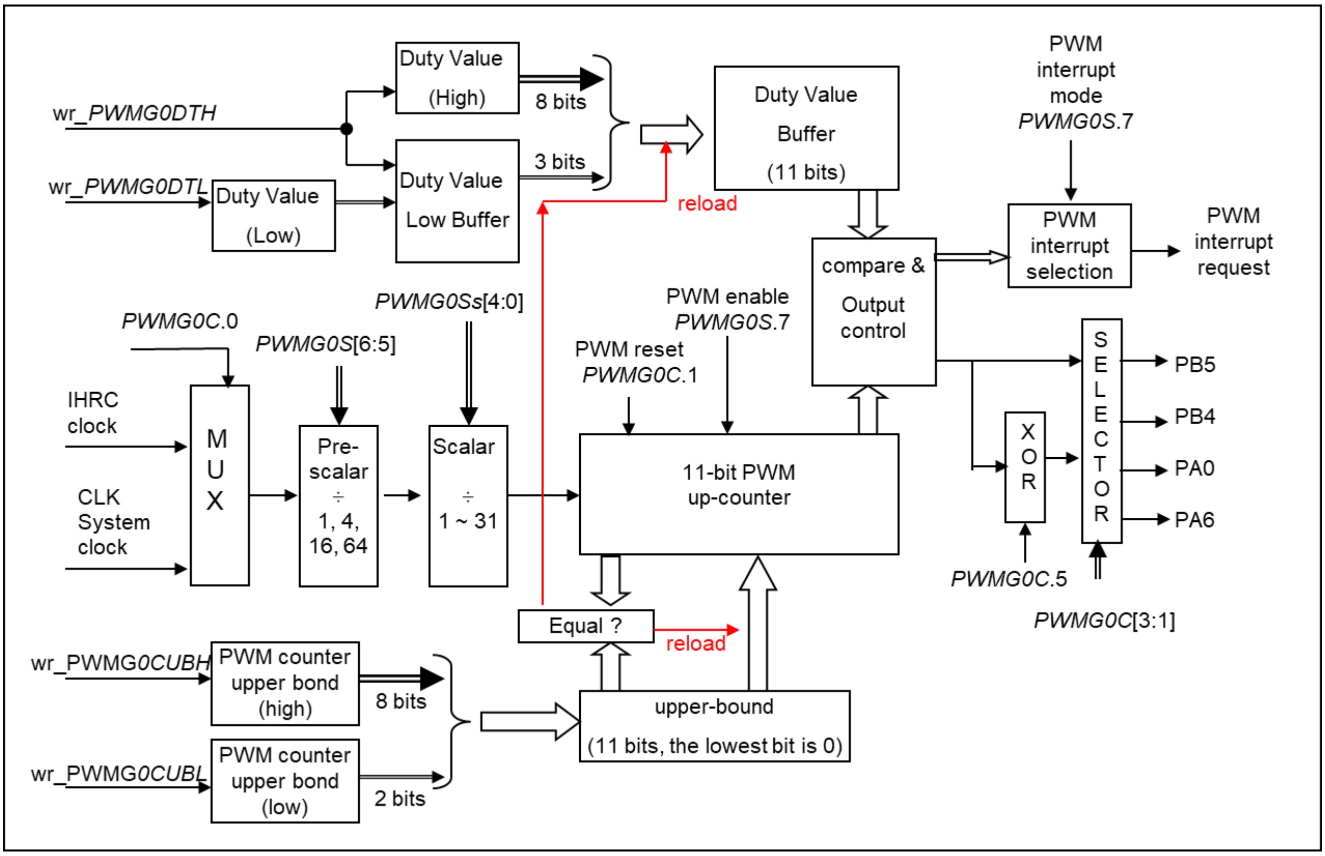


Fig.17: Hardware Diagram of 11-bit PWM Generator

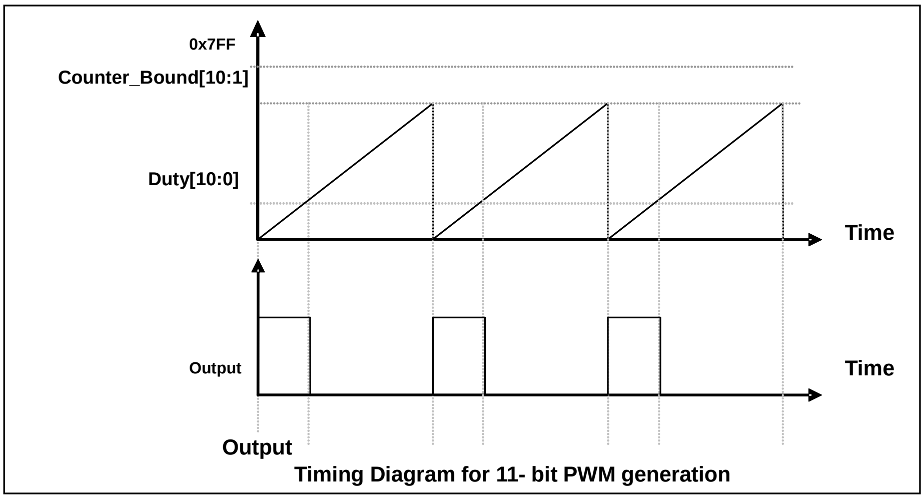


Fig.18: Output Timing Diagram of 11-bit PWM Generator

### 5.13.3. Equations for 11-bit PWM Generator

$$\text{PWM Frequency } F_{PWM} = F_{\text{clock source}} \div [\, P \times (K + 1) \times (CB10\_1 + 1)\, ]$$

$$\text{PWM Duty(in time)} = (1 / F_{PWM}) \times (\, DB10\_1 + DB0 \times 0.5 + 0.5) \div (CB10\_1 + 1)$$

$$\text{PWM Duty(in percentage)} = (\, DB10\_1 + DB0 \times 0.5 + 0.5) \div (CB10\_1 + 1) \times 100\%$$

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

### 5.13.4. 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.

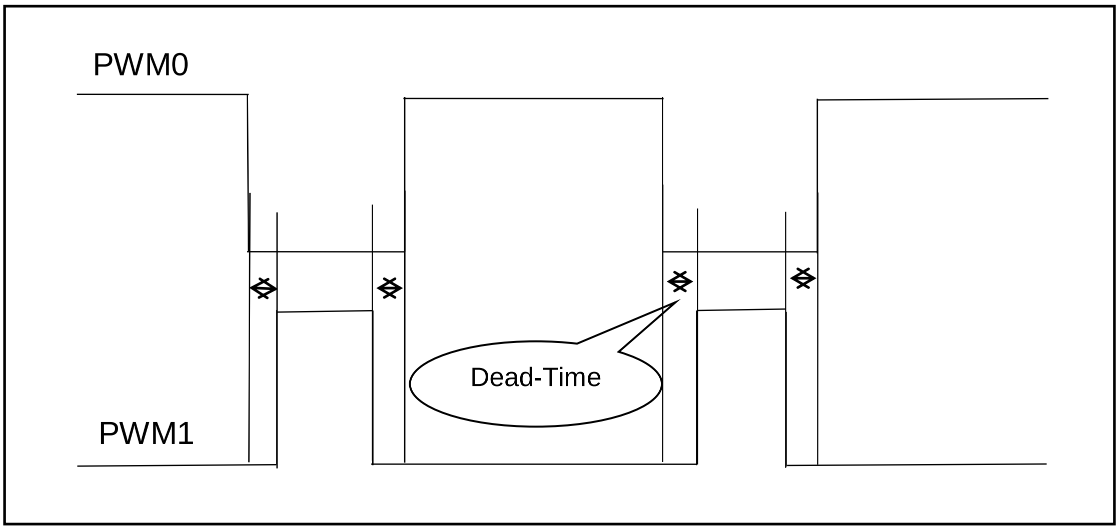


Fig. 19: Two complementary PWM waveforms with dead zones

Users can modify the **dead_zone_R** and **dead_zone_F** values in the program to adjust the dead-time. Table 8 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 8: 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 |
| ... | ... |

## 5.14. Touch Function

A touch detecting circuit is included in PMS163. Its functional block diagram is shown as Fig.20.

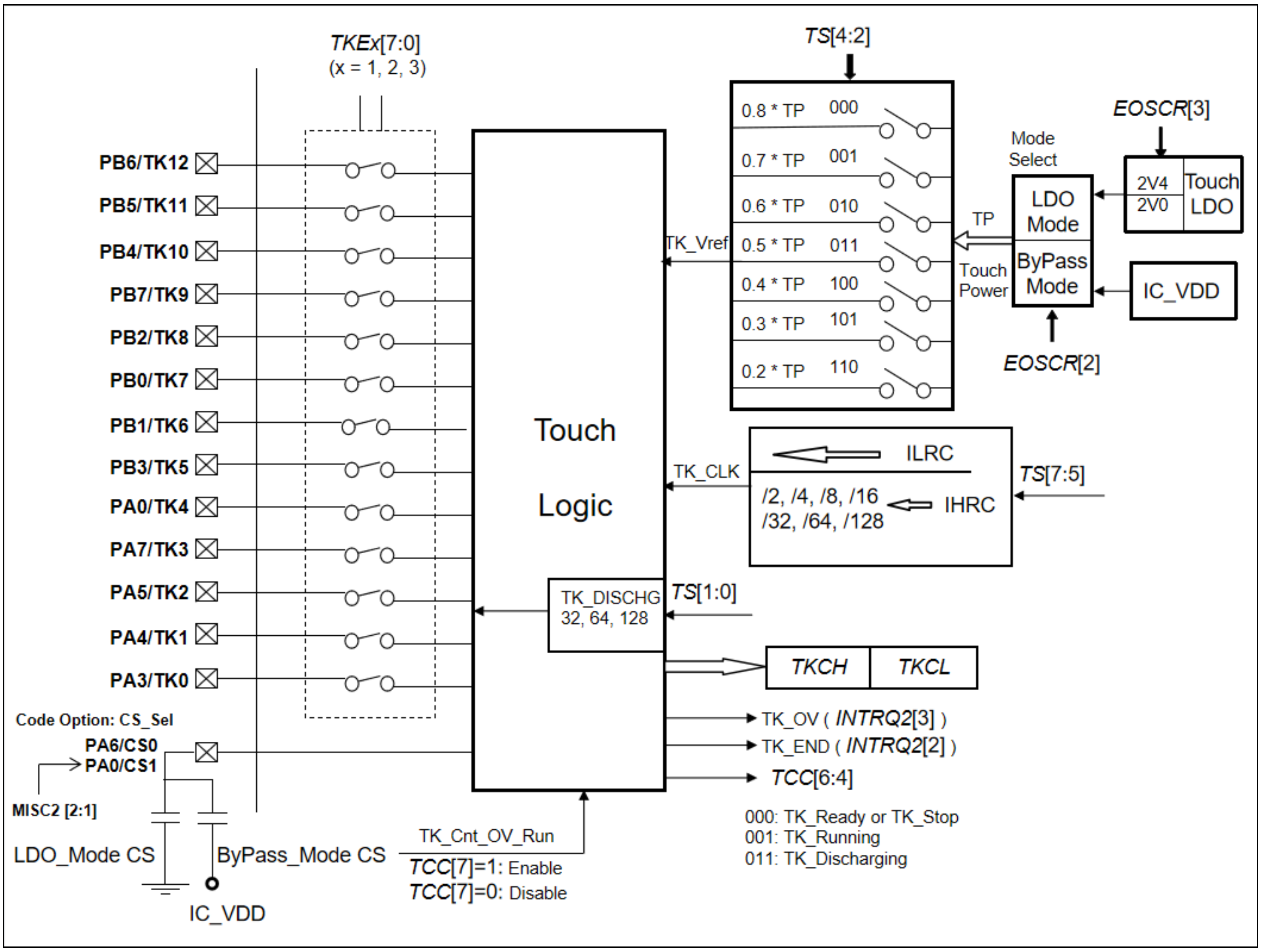


Fig. 20: Functional block diagram of the touch detecting circuit

The Touch detecting circuit in PMS163 applies the method of capacitive sensing, detecting the capacitive virtual ground effect of a finger, or the capacitance between sensors.

When using the touch function, the user can configure the touch module power through the register *ESOCR* [3:2].

1. Set *ESOCR*[2] to select ByPass/LDO mode.
2. when ByPass mode is selected, the touch module power supply is chip VDD, an accurate and low-leakage external capacitor CS is required to be connect between CS pin and VDD.
3. when the LDO mode is selected, the touch module power supply can be provided by 2.4V/2V LDO through the *ESCR*[3] selection, an accurate and low-leakage external capacitor CS is required to be connect between CS pin and GND.
4. In the mean time, user should set the misc2[2:1] or code option CS_Sel to configure it as CS pin, instead of PA6/PA0.

For starting touch detecting process, user should follow the procedures below:

1. Selecting the touch pad to be measure by setting TKE1 registers. Only one pad should be selected a time.
2. Issuing a Touch START command by writing “0x10” into TCC register. The capacitor CS will be automatically discharged to VSS firstly. The discharging time is selectable from 32, 64 and 128 Touch clocks by ***TS[1:0]***.
3. The larger the CS capacitance value, the longer the discharge time is needed to fully discharge the capacitor to VSS. However, in some cases, 128 Touch clocks may still be not long enough to fully discharge the CS capacitor. At this time, user should do it manually by writing “0x30” into TCC register instead of “0x10”. After a certain discharge time decided by the user, user can issue a Touch START (0x10) command to continue this touch conversion progress. Or user can also abort the conversion progress by writing “0x00” into TCC register.
4. After discharging, the CS will be charged toward VDD per Touch clock (TK_CLK). The charging speed is determined by the capacitance value of the selected Touch pad.
5. The charging progress will be stopped automatically when its voltage reaches the internal generated threshold voltage (VREF). The program determines whether the charging process is stopped by reading INTRQ[3].The VREF voltage is selectable from 0.8*TP, 0.7*TP, 0.6*TP, 0.5*TP, 0.4*TP, 0.3*TP and 0.2*TP by ***TS[4:2]***.
6. By reading the Touch Counter value from TKCH & TKCL registers, user can monitor the capacitance value change of the Touch pad. The value reads from Touch Counter is related to the ratio of CS and CP, while CP represents the total capacitance that is the combination of PCB, wire and touch pad whose capacitance can be varied by human finger’s touch. Once the CP value is altered, the periods required to charge the CS to VREF shorten. The user can judge whether the touch pad is pressed or released by reading the difference of the touch counter.
7. The user can change the sensitivity of the touch by adjusting the value of the CS capacitor. If a CS capacitor with an excessively large value is used, the touch counter value may overflow. At this time, the INTRQ.TK_OV flag will be automatically set by the hardware, and The touch count value will continue to count from 0 again.

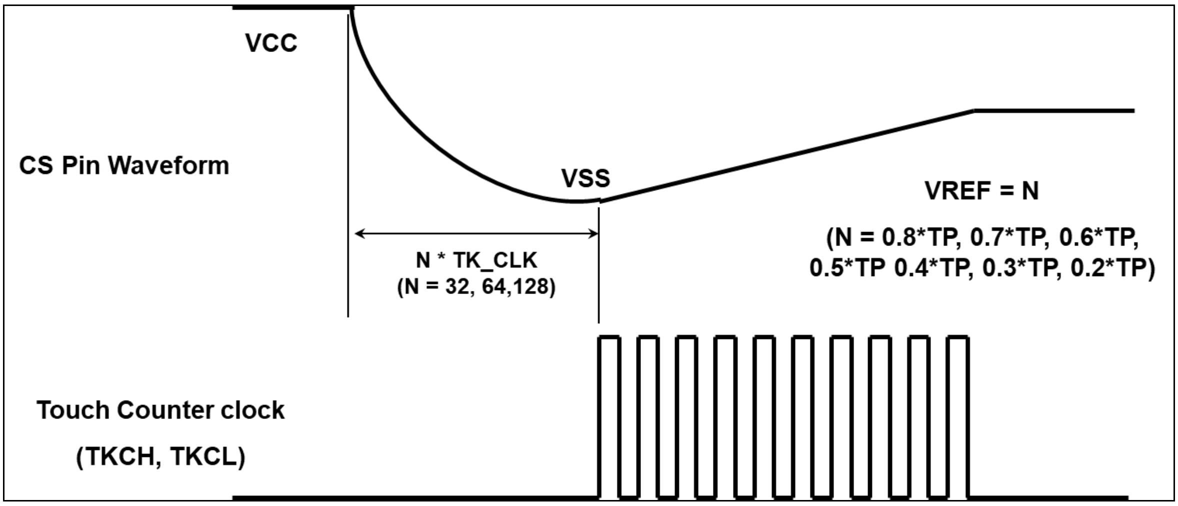


Fig. 21: Timing diagram of Touch converting progress

**Note:**

1. When the VREF voltage is first set or the reference voltage option is switched midway, please discard the first *TKCH* and *TKCL* data read after that.
2. The touch channel and ADC conversion channel should not be enabled on the same IO at the same time. If enabled at the same time, the touch key count will decrease. The default ADC conversion channel is PB0/TK7. When PB0/TK7 is used as the touch pin, the default ADC channel must be set to other pins.
3. Under the same conditions, the touch key count value of each IO pin may be different because of the capacitance effect of the IO pin (driving current, packaging, etc.). Touch key channels TK3/PA7 and TK4/PA0/CS1 have slightly smaller touch key counts than other pins.
4. In ByPass mode, the Touch START (write "0x10" into TCC register) command must be executed at a system frequency of 250KHz (IHRC/64). The LDO mode does not have this limitation.

## 5.15. Analog-to-Digital Conversion (ADC) module

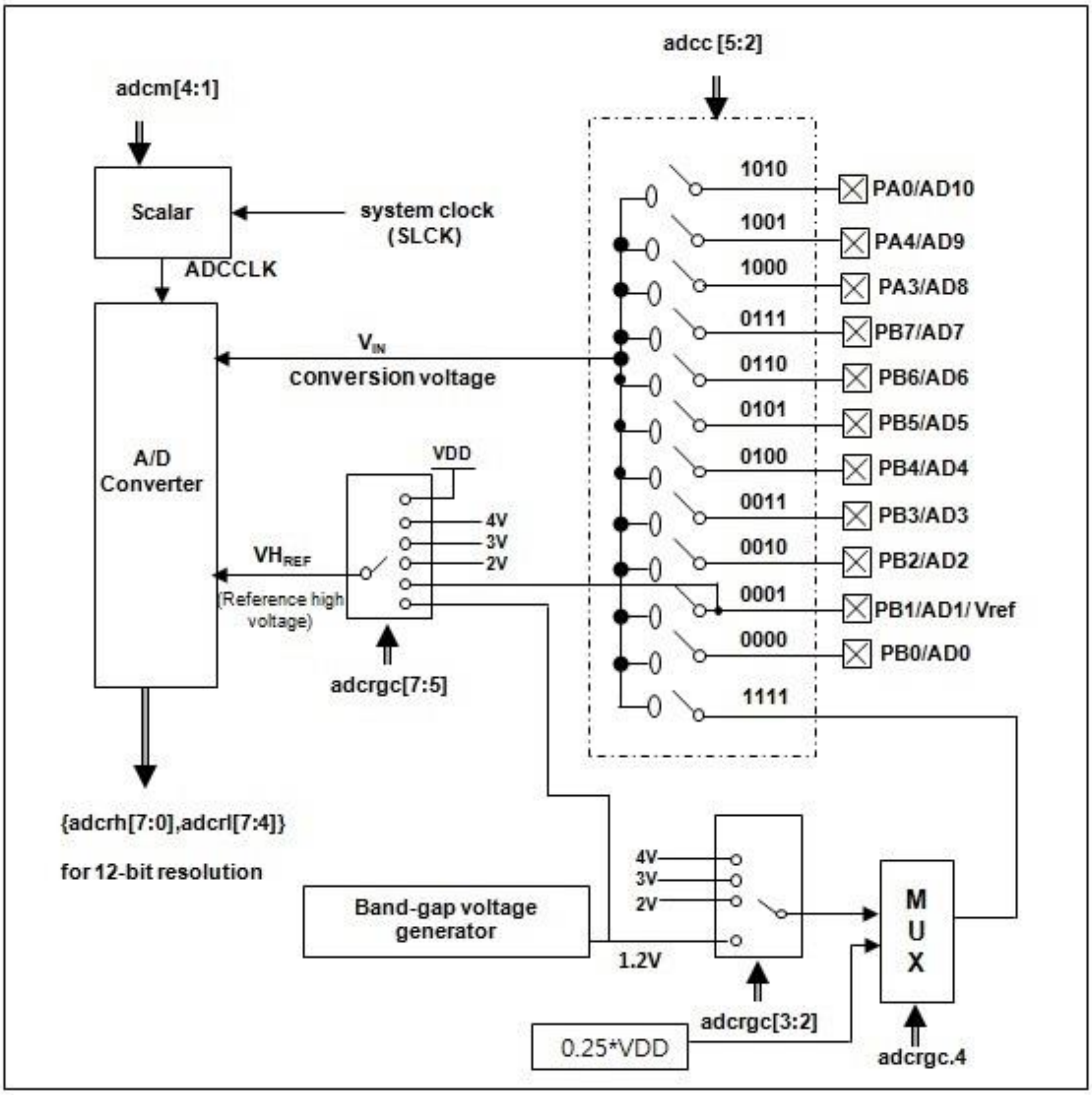


Fig.22: ADC Block Diagram

There are seven registers when using the ADC module, which are:

- ADC Control Register (***adcc***)
- ADC Regulator Control Register (***adcrgc***)
- ADC Mode Register (***adcm***)
- ADC Result High/Low Register (***adcrh***, ***adcrl***)
- Port A/B Digital Input Enable Register (***padier***, ***pbdier***)

The following steps are required to do the AD conversion procedure:

(1) Configure the voltage reference high by ***adcrgc*** register
(2) Configure the AD conversion clock by ***adcm*** register
(3) Configure the pin as analog input by ***padier***, ***pbdier*** register
(4) Select the ADC input channel by ***adcc*** register
(5) Enable the ADC module by ***adcc*** register
(6) Delay a certain amount of time after enabling the ADC module

Condition 1: Using bandgap 1.2V or 2V/3V/4V related circuit, either it is used as an internal reference high voltage or an AD Input channel, it must delay more than 1 ms when the time of 200 AD clocks is less than 1ms. Or it must delay 200 AD clocks when the time of 200 AD clocks is larger than 1ms. When internal BG/2V/3V/4V is enabled as reference high voltage, IHRC must be opened.

**Condition 2**: Without using any bandgap 1.2V or 2V/3V/4V related circuit, it needs to delay 200 AD clocks only.

**Note**: The 200 AD clocks in the above two conditions, which refer to the ADC conversion clock after configured by the ADCM register.

(7) Execute the AD conversion and check if ADC data is ready

set ‘1’ to ***addc***.6 to start the conversion and check whether ***addc***.6 is ‘1’

(8) Read the ADC result registers:

First read the ***adcrh*** register and then read the ***adcrl*** register.

If user power down the ADC and enable the ADC again, or switch ADC reference voltage and input channel, be sure to go to step 6 to confirm the ADC becomes ready before the conversion.

### 5.15.1. The input requirement for AD conversion

For the AD conversion to meet its specified accuracy, the charge holding capacitor ($C_{HOLD}$) must be allowed to fully charge to the voltage reference high level and discharge to the voltage reference low level. The analog input model is shown as Fig.23, the signal driving source impedance (Rs) and the internal sampling switch impedance (Rss) will affect the required time to charge the capacitor $C_{HOLD}$ directly. The internal sampling switch impedance may vary with ADC supply voltage; the signal driving source impedance will affect accuracy of analog input signal. User must ensure the measured signal is stable before sampling; therefore, the maximum signal driving source impedance is highly dependent on the frequency of signal to be measured. The recommended maximum impedance for analog driving source is about 10KΩ under 500KHz input frequency.

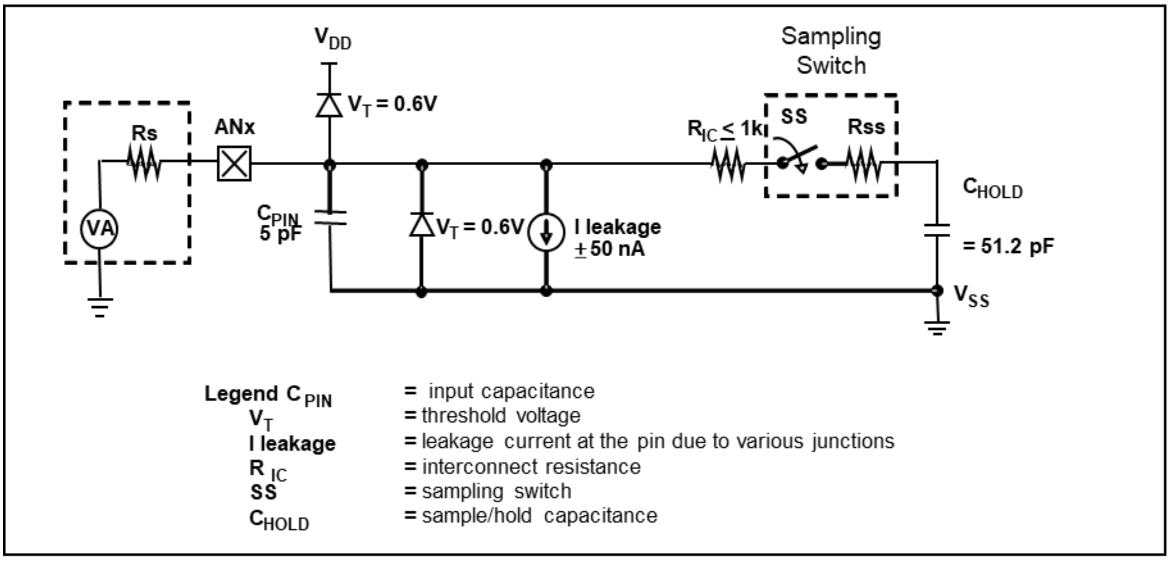

V_DD
V_T = 0.6V
Sampling Switch
Rs
ANx
R_IC ≤ 1k
SS
Rss
C_HOLD
VA
C_PIN 5 pF
V_T = 0.6V
I leakage ±50 nA
= 51.2 pF
V_SS
Legend C_PIN = input capacitance
V_T = threshold voltage
I leakage = leakage current at the pin due to various junctions
R_IC = interconnect resistance
SS = sampling switch
C_HOLD = sample/hold capacitance

Fig.23: Analog Input Model

Before starting the AD conversion, the minimum signal acquisition time should be met for the selected analog input signal, the selection of ADCLK must be met the minimum signal acquisition time.

### 5.15.2. Select the reference high voltage

The ADC reference high voltage can be selected via bit[7:5] of register ***adcrgc*** and its option can be $V_{DD}$, 4V, 3V, 2V, bandgap (1.20V) reference voltage or PB1 from external pin.

### 5.15.3. ADC clock selection

The clock of ADC module (ADCLK) can be selected by ***adcm*** register; there are 8 possible options for ADCLK from CLK÷1 to CLK÷128 (CLK is the system clock). Due to the signal acquisition time $T_{ACQ}$ is one clock period of ADCLK, the ADCLK must meet that requirement. The recommended ADC clock is to operate at 2us.

### 5.15.4. Configure the analog pins

There are 12 analog signals can be selected for AD conversion, 11 analog input signals come from external pins and one is from internal bandgap reference voltage or 0.25*$V_{DD}$. There are 4 voltage levels selectable for the internal Bandgap reference, they are 1.2V, 2V, 3V and 4V. For external pins, the analog signals are shared with Port A[0], Port A[3], Port A[4], and Port B[7:0]. To avoid leakage current at the digital circuit, those pins defined for analog input should disable the digital input function (set the corresponding bit of ***padier or pbdier*** register to be 0).

The measurement signals of ADC belong to small signal; it should avoid the measured signal to be interfered during the measurement period, the selected pin should (1) be set to input mode (2) turn off weak pull-high resistor (3) set the corresponding pin to analog input by port A/B digital input disable register (***padier / pbdier***).

### 5.15.5. Using the ADC

The following example shows how to use ADC with PB0~PB3.
First, defining the selected pins:

```
PBC      =   0B_XXXX_0000;        //    PB0 ~ PB3 as Input
PBPH     =   0B_XXXX_0000;        //    PB0 ~ PB3 without pull-high
PBDIER   =   0B_XXXX_0000;        //    PB0 ~ PB3 digital input is disabled
```

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

```
$   ADCC  Enable, PB3;          //    set PB3 as ADC input
$   ADCC  Enable, PB2;          //    set PB2 as ADC input
$   ADCC  Enable, PB0;          //    set PB0 as ADC input
// Note: Only one input channel can be seleceted for each AD conversion
```

Next, setting ***ADCM*** and ADCRGC register, example as below:

```
$   ADCM 12BIT, /16;                //    recommend /16 @System Clock=8MHz
                                    //    recommend ADCLK=500KHz
$   ADCM 12BIT, /8;                //    recommend /8 @System Clock=4MHz
                                    //    recommend ADCLK=500KHz
$   ADCRGC VDD;                     //    reference voltage is VDD
                                      //  the delay is 200 ADCLK
```

Next, delay 400us(ADCLK=500KHz, 200*ADCLK=400us), example as below:

```
.Delay 8*400;                 //  System Clock=8MHz
.Delay 4*400;                 //  System Clock=4MHz
```

Note: If using internal reference high voltage such as bandgap 1.2V or 2V/3V/4V, the delay time must be more than 1ms.

```
$  ASDCRGC   3V;              //    AD reference voltage is 3V
.Delay 4*1010;                //    if the system clock=4MHz
                                       //  the delay time must be more than 1ms
```

Please Note: If using bandgap 1.2V or 2V/3V/4V as ADC input channel, the delay time must be more than 1ms.

```
$  ADCC ADC
$  ADCRGC    VDD   ADC_BG   BG_2V    // reference voltage is VDD
                                     //      input channel is BG_2V
.Delay 4*1010;                //    if the system clock=4MHz
                                     //  the delay time must be more than 1ms
```

Then, start the ADC conversion:

```
AD_START      =     1;              //     start ADC conversion
while (! AD_DONE)   NULL;           //     wait ADC conversion result
```

Finally, it can read ADC result when AD_DONE is high:

```
WORD         Data;                  //     two bytes result: ADCRH and ADCRL
Data$1     =     ADCRH
Data$0    =    ADCRL;
Data       =     Data >> 4;
```

The ADC can be disabled by using the following method:

```
$  ADCC  Disable;
```

or

```
ADCC          =     0;
```

## 5.16. Multiplier

There is an 8x8 multiplier on-chip to enhance hardware capability in arithmetic function, its multiplication is an 8x8 unsigned operation and can be finished in one clock cycle. Before issuing the *mul* command, both multiplicand and multiplicator must be put on ACC and register *mulop* (0x08); After *mul* command, the high byte result will be put on register *mulrh* (0x09) and low byte result on ACC. The hardware diagram of this multiplier is shown as Fig.24.

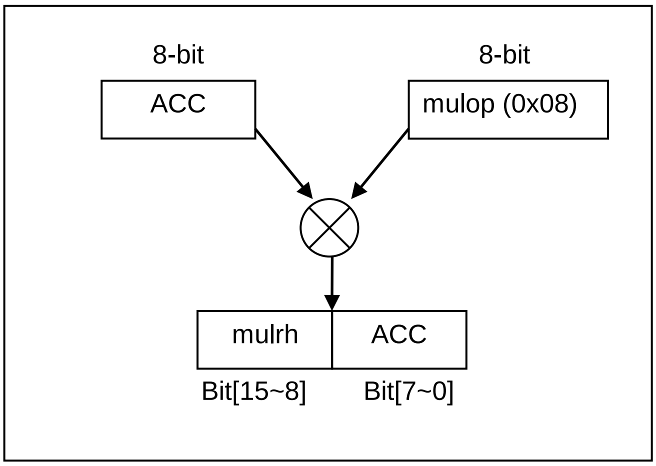


Fig.24: Block diagram of hardware multiplier

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

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

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

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

<table>
<tr><th>Bit</th><th>Reset</th><th>R/W</th><th colspan="2">Description</th></tr>
<tr><td rowspan="3">7 - 5</td><td rowspan="3">111</td><td rowspan="3">R/W</td><td colspan="2">System clock selection:</td></tr>
<tr><td>Type 0, clkmd[3]=0</td><td>Type 1, clkmd[3]=1</td></tr>
<tr><td>000: IHRC/4<br>001: IHRC/2<br>01x: reserved<br>100: reserved<br>101: reserved<br>110: ILRC/4<br>111: ILRC (default)</td><td>000: IHRC/16<br>001: IHRC/8<br>010: ILRC/16 (ICE doesn’t support)<br>011: IHRC/32<br>100: IHRC/64<br>110: reserved<br>1x1: reserved</td></tr>
<tr><td>4</td><td>0</td><td>R/W</td><td colspan="2">IHRC oscillator Enable. 0 / 1: disable / enable</td></tr>
<tr><td>3</td><td>0</td><td>R/W</td><td colspan="2">Clock Type Select. This bit is used to select the clock type in bit [7:5].<br>0 / 1: Type 0 / Type 1</td></tr>
<tr><td>2</td><td>1</td><td>R/W</td><td colspan="2">ILRC Enable. 0 / 1: disable / enable<br>If ILRC is disabled, watchdog timer is also disabled.</td></tr>
<tr><td>1</td><td>1</td><td>R/W</td><td colspan="2">Watch Dog Enable. 0 / 1: disable / enable</td></tr>
<tr><td>0</td><td>0</td><td>R/W</td><td colspan="2">Pin PA5/PRSTB function. 0 / 1: PA5 / PRSTB</td></tr>
</table>

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

| Bit | Reset | R/W | Description |
|---|---|---|---|
| 7 | 0 | R/W | Enable interrupt from Timer3 / PWMG2.<br>0 / 1: disable / enable |
| 6 | 0 | R/W | Enable interrupt from Timer2.<br>0 / 1: disable / enable |
| 5 | 0 | R/W | Enable interrupt from PWMG0.<br>0 / 1: disable / enable |
| 4 | 0 | R/W | Enable interrupt from comparator / PWMG1.<br>0 / 1: disable / enable |
| 3 | 0 | R/W | Enable interrupt from ADC.<br>0 / 1: disable / enable |
| 2 | 0 | R/W | Enable interrupt from Timer16 overflow.<br>0 / 1: disable / enable |
| 1 | 0 | R/W | Enable interrupt from PB0/PA4.<br>0 / 1: disable / enable |
| 0 | 0 | R/W | Enable interrupt from PA0/PB5.<br>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 / PWMG2, this bit is set by hardware and cleared by software.<br>0 / 1: No request / Request |
| 6 | - | R/W | Interrupt Request from Timer2, this bit is set by hardware and cleared by software.<br>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 / PWMG1, this bit is set by hardware and cleared by software.<br>0 / 1: No request / Request |
| 3 | - | R/W | Interrupt Request from ADC, this bit is set by hardware and cleared by software. 0 / 1: No request / Request |
| 2 | - | R/W | Interrupt Request from Timer16, this bit is set by hardware and cleared by software.<br>0 / 1: No request / Request |
| 1 | - | R/W | Interrupt Request from pin PB0/PA4, this bit is set by hardware and cleared by software.<br>0 / 1: No request / Request |
| 0 | - | R/W | Interrupt Request from pin PA0/PB5, this bit is set by hardware and cleared by software.<br>0 / 1: No request / Request |

## 6.6. Interrupt Enable 2 Register (*inten2*), IO address = 0x4a

| Bit | Reset | R/W | Description |
|---|---|---|---|
| 7 - 4 | 0 | R/W | Read as 0 |
| 3 | 0 | R/W | Enable interrupt from Touch Key TK_OV.<br>0 / 1: disable / enable |
| 2 | 0 | R/W | Enable interrupt from Touch Key TK_END.<br>0 / 1: disable / enable |
| 1 - 0 | 0 | R/W | Reserved. |

## 6.7. Interrupt Request 2 Register (*intrq2*), IO address = 0x4b

| Bit | Reset | R/W | Description |
|---|---|---|---|
| 7 - 4 | - | - | Reserved. |
| 3 | - | R/W | Interrupt Request from Touch Key TK_OV, this bit is set by hardware and cleared by software.<br>0 / 1: No request / Request |
| 2 | - | R/W | Interrupt Request from Touch Key TK_END, this bit is set by hardware and cleared by software.<br>0 / 1: No request / Request |
| 1 - 0 | - | - | Reserved. |

## 6.8. Multiplier Operand Register (*mulop*), IO address = 0x08

| Bit | Reset | R/W | Description |
|---|---|---|---|
| 7 - 0 | - | R/W | Operand for hardware multiplication operation. |

## 6.9. Multiplier Result High Byte Register (*mulrh*), IO address = 0x09

| Bit | Reset | R/W | Description |
|---|---|---|---|
| 7 - 0 | - | RO | High byte result of multiplication operation (read only). |

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

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

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

| Bit | Reset | R/W | Description |
|---|---|---|---|
| 7 - 5 | - | - | Reserved. Please keep 0. |
| 4 | 0 | WO | Timer16 edge selection.<br>0 : rising edge to trigger interrupt<br>1 : falling edge to trigger interrupt |
| 3 - 2 | 00 | WO | PB0 / PA4 edge selection.<br>00 : both rising edge and falling edge to trigger interrupt<br>01 : rising edge to trigger interrupt<br>10 : falling edge to trigger interrupt<br>11 : reserved. |
| 1 - 0 | 00 | WO | PA0 / PB5 edge selection.<br>00 : both rising edge and falling edge to trigger interrupt<br>01 : rising edge to trigger interrupt<br>10 : falling edge to trigger interrupt<br>11 : reserved. |

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

| Bit | Reset | R/W | Description |
|---|---|---|---|
| 7 - 6 | 11 | WO | Enable PA7~PA6 digital input and wake up event. 1 / 0 : enable / disable<br>These bits can be set to low to disable wake up from PA7~PA6 toggling. |
| 5 | 1 | WO | Enable PA5 digital input and wake up event. 1 / 0 : enable / disable<br>These bits can be set to low to disable wake up from PA5 toggling. |
| 4 | 1 | WO | Enable PA4 digital input, wake-up event and interrupt request. 1 / 0 : enable / disable.<br>This bit can be set to low to prevent leakage current when PA4 is assigned as AD input, and to disable wake-up from PA4 toggling and interrupt request from this pin. |
| 3 | 1 | WO | Enable PA3 digital input and wake-up event. 1 / 0 : enable / disable.<br>This bit should be set to low when PA3 is assigned as AD input to prevent leakage current. If this bit is set to low, PA3 can NOT be used to wake-up the system. |
| 2 - 1 | - | - | Reserved. (Please keep 00 for future compatibility) |
| 0 | 1 | WO | Enable PA0 digital input, wake up event and interrupt request. 1 / 0 : enable / disable<br>This bit can be set to low to disable wake up from PA0 toggling and interrupt request from this pin. |

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

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

## 6.14. 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. |

## 6.15. 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 |

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

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

## 6.17. Port B Data Register (*pb*), IO address = 0x14

| Bit | Reset | R/W | Description |
|---|---|---|---|
| 7 - 0 | 0x00 | R/W | Data register for Port B. |

## 6.18. Port B Control Register (*pbc*), IO address = 0x15

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

### 6.19. Port B Pull-High Register (*pbph*), IO address = 0x16

| Bit | Reset | R/W | Description |
|---|---|---|---|
| 7 - 0 | 0x00 | R/W | Port B pull-high register. This register is used to enable the internal pull-high device on each corresponding pin of port B and this pull high function is active only for input mode.<br>0 / 1 : disable / enable |

### 6.20. Port B Pull-Low Registers (*pbpl*), IO address = 0x1f

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

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

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

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

| Bit | Reset | R/W | Description |
|---|---|---|---|
| 7 | 0 | WO | Comparator output enable (to PA0).<br>0 / 1 : disable / enable |
| 6 | 0 | WO | Wakeup by comparator enable. (The comparator wakeup effectively when gpcc.6 electrical level changed)<br>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.<br>0000 (lowest) ~ 1111 (highest) |

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

| Bit | Reset | R/W | Description |
|---|---|---|---|
| 7 - 4 | 0000 | R/W | Timer2 clock selection.<br>0000 : disable<br>0001 : system clock<br>0010 : IHRC<br>0011 : reserved<br>0100 : ILRC<br>0101 : comparator output<br>0110 : reserved<br>0111 : NILRC<br>1000 : PA0 (rising edge)<br>1001 : ~PA0 (falling edge)<br>1010 : PB0 (rising edge)<br>1011 : ~PB0 (falling edge)<br>1100 : PA4 (rising edge)<br>1101 : ~PA4 (falling edge) |
| 3 - 2 | 00 | R/W | Timer2 output selection.<br>00 : disable<br>01 : PB2<br>10 : PA3<br>11 : PB4 |
| 1 | 0 | R/W | Timer2 mode selection.<br>0 / 1 : period mode / PWM mode |
| 0 | 0 | R/W | Enable to inverse the polarity of Timer2 output.<br>0 / 1: disable / enable |

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

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

### 6.25. Timer2 Scalar Register (*tm2s*), IO address = 0x1e

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

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

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

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

| Bit | Reset | R/W | Description |
|---|---|---|---|
| 7 - 4 | 0000 | R/W | Timer3 clock selection.<br>0000 : disable<br>0001 : system clock<br>0010 : IHRC<br>0011 : reserved<br>0100 : ILRC<br>0101 : comparator output<br>0110 : reserved<br>0111 : NILRC<br>1000 : PA0 (rising edge)<br>1001 : ~PA0 (falling edge)<br>1010 : PB0 (rising edge)<br>1011 : ~PB0 (falling edge)<br>1100 : PA4 (rising edge)<br>1101 : ~PA4 (falling edge) |
| 3 - 2 | 00 | R/W | Timer3 output selection.<br>00 : disable<br>01 : PB5<br>10 : PB6<br>11 : PB7 |
| 1 | 0 | R/W | Timer3 mode selection.<br>0 / 1 : period mode / PWM mode |
| 0 | 0 | R/W | Enable to inverse the polarity of Timer3 output.<br>0 / 1: disable / enable |

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

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

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

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

### 6.30. Timer3 Bound Register (*tm3b*), IO address = 0x3f

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

### 6.31. Touch Selection Register (*ts*), IO address = 0x41

| Bit | Reset | R/W | Description |
|---|---|---|---|
| 7 – 5 | 000 | R/W | Touch clock selection (TK_CLK)<br>000: ILRC<br>001: IHRC/2<br>010: IHRC/4<br>011: IHRC/8<br>100: IHRC/16<br>101: IHRC/32<br>110: IHRC/64<br>111: IHRC/128 |
| 4 – 2 | 011 | R/W | Touch VREF selection (TP: Touch Power, the default is LDO 2V)<br>000: 0.8 * TP<br>001: 0.7 * TP<br>010: 0.6 * TP<br>011: 0.5 * TP<br>100: 0.4 * TP<br>101: 0.3 * TP<br>110: 0.2 * TP<br>111: reserved |
| 1 - 0 | 00 | R/W | Select the discharge time before starting the touch function (TK_DISCHG)<br>0x: reserved<br>10: 64 * CLK<br>11: 128 * CLK |

Note: LDO mode TP defaults to LDO 2V, ByPass mode TP is IC_VDD.

### 6.32. Touch Charge Control Register (*tcc*), IO address = 0x42

<table>
<tr><th>Bit</th><th>Reset</th><th>R/W</th><th colspan="3">Description</th></tr>
<tr><td>7</td><td>0</td><td>-</td><td colspan="3">OV Enable 0/1: disable/enable<br>After OV is enabled, the counter will start counting from zero automatically when it overflows.</td></tr>
<tr><td rowspan="6">6 - 4</td><td rowspan="6">-</td><td rowspan="6">WO</td><td colspan="3">Touch control and status</td></tr>
<tr><td><b>Data</b></td><td><b>Command (W)</b></td><td><b>Status (R)</b></td></tr>
<tr><td>000</td><td>TK_STOP<br>(Touch module power down)</td><td>Ready / End</td></tr>
<tr><td>001</td><td>TK_RUN<br>(Touch START)</td><td>Running</td></tr>
<tr><td>011</td><td>Discharge<br>(Discharge CS capacitor)</td><td>Discharging</td></tr>
<tr><td>Others</td><td>Reserved</td><td>Reserved</td></tr>
<tr><td>3 - 0</td><td>-</td><td>-</td><td colspan="3">reserved</td></tr>
</table>

Note: In ByPass mode, the Touch START (write "0x10" into TCC register) command must be executed at a system frequency of 250KHz (IHRC/64). The LDO mode does not have this limitation.

### 6.33. Touch Key Enable 2 Register (*tke2*), IO address = 0x43

| Bit | Reset | R/W | Description |
|---|---|---|---|
| 7 - 5 | - | - | reserved |
| 4 | 0 | R/W | Enable PB6/TK12. 0/1: disable/enable |
| 3 | 0 | R/W | Enable PB5/TK11. 0/1: disable/enable |
| 2 | 0 | R/W | Enable PB4/TK10. 0/1: disable/enable |
| 1 | 0 | R/W | Enable PB7/TK9. 0/1: disable/enable |
| 0 | 0 | R/W | Enable PB2/TK8. 0/1: disable/enable |

Note: The touch channel and ADC conversion channel should not be enabled on the same IO at the same time.

### 6.34. Touch Key Enable 1 Register (*tke1*), IO address = 0x45

| Bit | Reset | R/W | Description |
|---|---|---|---|
| 7 | 0 | R/W | Enable PB0/TK7. 0/1: disable/enable |
| 6 | 0 | R/W | Enable PB1/TK6. 0/1: disable/enable |
| 5 | 0 | R/W | Enable PB3/TK5. 0/1: disable/enable |
| 4 | 0 | R/W | Enable PA0/TK4. 0/1: disable/enable |
| 3 | 0 | R/W | Enable PA7/TK3. 0/1: disable/enable |
| 2 | 0 | R/W | Enable PA5/TK2. 0/1: disable/enable |
| 1 | 0 | R/W | Enable PA4/TK1. 0/1: disable/enable |
| 0 | 0 | R/W | Enable PA3/TK0. 0/1: disable/enable |

Note:

1. When PB0/TK7 is used as the touch key channel, it is necessary to change the default ADC channel to another channel.
2. The touch channel and ADC conversion channel should not be enabled on the same IO at the same time.

### 6.35. Touch Key Charge Counter High Register (*tkch*), IO address = 0x48

| Bit | Reset | R/W | Description |
|---|---|---|---|
| 7 - 4 | - | - | Reserved |
| 3 - 0 | - | RO | tkc [11:8] of touch key charge counter. |

### 6.36. Touch Key Charge Counter Low Register (*tkcl*), IO address = 0x49

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

### 6.37. Touch parameter setting Register (*tps*), IO address = 0x46

| Bit | Reset | R/W | Description |
|---|---|---|---|
| 7 - 0 | 0x00 | R/W | Reserved, keep Default value or 0x00 |

Note: TPS = 0x00;

### 6.38. Touch parameter setting Register 2 (*tps2*), IO address = 0x47

| Bit | Reset | R/W | Description |
|---|---|---|---|
| 7 - 6 | - | R/W | Touch module type：(For specific settings of different power supplies, please refer to Section 11.6)<br>00：Type A (ByPass mode，CS capacitor connect to VDD)<br>01：Type B (LDO mode，CS capacitor connect to GND) |
| 5 - 2 | 0000 | R/W | Reserved, keep 0 |
| 1 - 0 | 00 | R/W | 01：VREF always on throught entire process. Suggestion:keep 0x01 |

```
Case:
// Bypass mode:
$ EOSCR TK_VDD;
$ TPS2 Type A, Always On            //ByPass, Type A, CS capacitor connect to VDD

// LDO mode
$ EOSCR TK 2V,TK_LDO
$ TPS2 Type B, Always_On       //LDO, Type B, CS capacitor connect to GND
```

### 6.39. External Power Operate Register (eoscr), IO address = 0x0a

| Bit | Reset | R/W | Description |
|---|---|---|---|
| 7-4 | - | - | Reserved |
| 3 | 0 | WO | LDO output voltage option。0/1：2.4V/2V |
| 2 | 0 | WO | Touch module power option。0/1：VDD/LDO |
| 1 | - | - | Reserved |
| 0 | 0 | WO | BG LVD power。0/1：On/Off |

Note: Set EOSCR[3:2] to select touch module power (TP).

### 6.40. ADC Control Register (*adcc*), IO address = 0x3b

| Bit | Reset | R/W | Description |
|---|---|---|---|
| 7 | 0 | R/W | Enable ADC function. 0/1: Disable/Enable. |
| 6 | 0 | R/W | ADC process control bit.<br>Read "1" to indicate the ADC is ready. |
| 5 - 2 | 0000 | R/W | Channel selector. These four bits are used to select input signal for AD conversion.<br>0000: PB0/AD0,<br>0001: PB1/AD1,<br>0010: PB2/AD2,<br>0011: PB3/AD3,<br>0100: PB4/AD4,<br>0101: PB5/AD5,<br>0110: PB6/AD6,<br>0111: PB7/AD7,<br>1000: PA3/AD8,<br>1001: PA4/AD9,<br>1010: PA0/AD10,<br>1111: (Channel F) Bandgap reference voltage or 0.25*$V_{DD}$<br>Others: reserved |
| 0 - 1 | - | - | Reserved. (keep 0 for future compatibility) |

### 6.41. ADC Mode Register (*adcm*), IO address = 0x3c

| Bit | Reset | R/W | Description |
|---|---|---|---|
| 7 - 4 | - | - | Reserved (keep 0 for future compatibility) |
| 3 - 1 | 000 | WO | ADC clock source selection.<br>000: CLK (system clock) ÷ 1,<br>001: CLK (system clock) ÷ 2,<br>010: CLK (system clock) ÷ 4,<br>011: CLK (system clock) ÷ 8,<br>100: CLK (system clock) ÷ 16,<br>101: CLK (system clock) ÷ 32,<br>110: CLK (system clock) ÷ 64,<br>111: CLK (system clock) ÷ 128, |
| 0 | - | - | Reserved |

### 6.42. ADC Regulator Control Register (adcrgc), IO address = 0x3d

| Bit | Reset | R/W | Description |
|---|---|---|---|
| 7 - 5 | 000 | WO | These three bits are used to select input signal for ADC reference high voltage.<br>000: $V_{DD}$,<br>001: 2V,<br>010: 3V,<br>011: 4V,<br>100: PB1,<br>101: Bandgap 1.20 volt reference voltage<br>Others: reserved |
| 4 | 0 | WO | ADC channel F selector:<br>0: Bandgap reference voltage<br>1: 0.25*$V_{DD}$. The deviation is within ±0.01*$V_{DD}$ mostly. |
| 3 - 2 | 00 | WO | Bandgap reference voltage selector for ADC channel F:<br>00: 1.2V<br>01: 2V<br>10: 3V<br>11: 4V |
| 1 - 0 | - | - | Reserved. Please keep 0. |

### 6.43. ADC Result High Register (*adcrh*), IO address = 0x3e

| Bit | Reset | R/W | Description |
|---|---|---|---|
| 7 - 0 | - | RO | These eight read-only bits will be the bit [11:4] of AD conversion result. The bit 7 of this register is the MSB of ADC result for any resolution. |

### 6.44. ADC Result Low Register (*adcrl*), IO address = 0x3f

| Bit | Reset | R/W | Description |
|---|---|---|---|
| 7 - 4 | - | RO | These four bits will be the bit [3:0] of AD conversion result. |
| 3 - 0 | - | - | Reserved |

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

| Bit | Reset | R/W | Description |
|---|---|---|---|
| 7 | 0 | WO | Enable PWMG0 generator. 0 / 1 : disable / enable. |
| 6 | - | RO | Output status of PWMG0 generator. |
| 5 | 0 | WO | Enable to inverse the polarity of PWMG0 generator output. 0 / 1 : disable / enable. |
| 4 | 0 | WO | PWMG0 counter reset.<br>Writing “1” to clear PWMG0 counter and this bit will be self clear to 0 after counter reset. |
| 3 - 1 | 0 | WO | Select PWM output pin for PWMG0.<br>000: none<br>001: PB5<br>010:PA6<br>011: PA0<br>100: PB4<br>Others: reserved |
| 0 | 0 | WO | Clock source of PWMG0 generator.<br>0 : SYSCLK<br>1 : IHRC or IHRC * 2 (by Code Option: PWM_Source) |

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

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

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

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

### 6.48. 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.

### 6.49. 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. |

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

| Bit | Reset | R/W | Description |
|---|---|---|---|
| 7 - 6 | 00 | WO | Bit[2:1] of PWMG0 counter upper bound. |
| 5 - 0 | - | - | Reserved |

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

| Bit | Reset | R/W | Description |
|---|---|---|---|
| 7 | 0 | WO | Enable PWMG1. 0 / 1 : disable / enable. |
| 6 | - | RO | Output of PWMG1. |
| 5 | 0 | WO | Enable to inverse the polarity of PWMG1 output. 0 / 1 : disable / enable. |
| 4 | 0 | WO | PWMG1 counter reset.<br>Writing “1” to clear PWMG1 counter. |
| 3 - 1 | 0 | WO | Select PWMG1 output pin.<br>000: none<br>001: PB6<br>011: PA4<br>100: PB7<br>Others: reserved |
| 0 | 0 | WO | Clock source of PWMG1.<br>0 : SYSCLK<br>1 : IHRC or IHRC * 2 (by Code Option : PWM_Source) |

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

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

### 6.53. 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. |

### 6.54. PWMG1 Counter Upper Bound Low Register (*pwmg1cubl*), IO address = 0x2B

| Bit | Reset | R/W | Description |
|---|---|---|---|
| 7 - 6 | 00 | WO | Bit[2:1] of PWMG1 counter upper bound. |
| 5 - 0 | - | - | Reserved |

### 6.55. 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. |

### 6.56. 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.

### 6.57. PWMG2 control Register (*pwmg2c*), IO address = 0x2C

| Bit | Reset | R/W | Description |
|---|---|---|---|
| 7 | 0 | WO | Enable PWMG2. 0 / 1 : disable / enable. |
| 6 | - | RO | Output of PWMG2. |
| 5 | 0 | WO | Enable to inverse the polarity of PWMG2 output. 0 / 1 : disable / enable. |
| 4 | 0 | WO | PWMG2 counter reset.<br>Writing “1” to clear PWMG2 counter. |
| 3 - 1 | 0 | WO | Select PWMG2 output pin.<br>000: disable<br>001: PB3<br>010: PA7<br>011: PA3<br>100: PB2<br>101: PA5 ( ICE does NOT support )<br>Others: reserved |
| 0 | 0 | WO | Clock source of PWMG2.<br>0 : SYSCLK<br>1 : IHRC or IHRC * 2 (by Code Option : PWM_Source) |

### 6.58. PWMG2 Scalar Register (*pwmg2s*), IO address = 0x2D

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

### 6.59. 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. |

### 6.60. 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.

### 6.61. 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. |

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

| Bit | Reset | R/W | Description |
|---|---|---|---|
| 7 - 6 | 00 | WO | Bit[2:1] of PWMG2 counter upper bound. |
| 5 - 0 | - | - | Reserved |

### 6.63. Miscellaneous Register (*misc*), IO address = 0x17

| 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.<br>0: Normal wake-up.<br>The wake-up time is 3000 ILRC clocks (Not for fast boot-up)<br>1: Fast wake-up.<br>The wake-up time is 45 ILRC clocks |
| 4 - 3 | - | - | Reserved. (keep 0 for future compatibility) |
| 2 | 0 | WO | Disable LVR function.<br>0 / 1 : Enable / Disable |
| 1 - 0 | 00 | WO | Watch dog time out period.<br>00: 8k ILRC clock period<br>01: 16k ILRC clock period<br>10: 64k ILRC clock period<br>11: 256k ILRC clock period |

### 6.64. Miscellaneous Register2 (*misc2*), IO address = 0x0f

| Bit | Reset | R/W | Description |
|---|---|---|---|
| 7 | - | - | Reserved |
| 6 - 5 | - | - | GPC trigger interrupt edge<br>00: Bidirectional edge<br>01: Rising edge<br>10: Falling edge<br>11: N.A. |
| 4 | 0 | WO | Interrupt 7 source from<br>0: TM3.<br>1: PWMG2. |
| 3 | 0 | WO | Interrupt 4 source from<br>0: GPC0.<br>1: PWMG1. |
| 2 | 1 | WO | CS0/PA6 select<br>0 / 1 : CS0 / PA6 |
| 1 | 1 | WO | CS1/PA0 select<br>0 / 1 : CS1 / PA0 |
| 0 | 0 | WO | NILRC enable<br>0: disable<br>1: enable |

## 7. Instructions

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

## 7.1. Data Transfer Instructions

| | |
|---|---|
| *mov* a, I | Move immediate data into ACC.<br>Example: *mov* a, 0x0f;<br>Result: a ← 0fh;<br>Affected flags: 『N』Z 『N』C 『N』AC 『N』OV |
| *mov* M, a | Move data from ACC into memory<br>Example: *mov* MEM, a;<br>Result: MEM ← a<br>Affected flags: 『N』Z 『N』C 『N』AC 『N』OV |
| *mov* a, M | Move data from memory into ACC<br>Example: *mov* a, MEM ;<br>Result: a ← MEM; Flag Z is set when MEM is zero.<br>Affected flags: 『Y』Z 『N』C 『N』AC 『N』OV |
| *mov* a, IO | Move data from IO into ACC<br>Example: *mov* a, pa ;<br>Result: a ← pa; Flag Z is set when pa is zero.<br>Affected flags: 『Y』Z 『N』C 『N』AC 『N』OV |
| *mov* IO, a | Move data from ACC into IO<br>Example: *mov* pa, a;<br>Result: pa ← a<br>Affected flags: 『N』Z 『N』C 『N』AC 『N』OV |
| *ldt16 word* | Move 16-bit counting values in Timer16 to memory in word.<br>Example: ldt16 word;<br>Result: word ← 16-bit timer<br>Affected flags: 『N』Z 『N』C 『N』AC 『N』OV<br><br>Application Example:<br>------------------------------------------------------------------------------------------------------------------------<br>word T16val ; // declare a RAM word<br>…<br>clear lb@ T16val ; // clear T16val (LSB)<br>clear hb@ T16val ; // clear T16val (MSB)<br>stt16 T16val ; // initial T16 with 0<br>…<br>set1 t16m.5 ; // enable Timer16<br>…<br>set0 t16m.5 ; // disable Timer 16<br>ldt16 T16val ; // save the T16 counting value to T16val<br>….<br>------------------------------------------------------------------------------------------------------------------------ |

*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<br>Example: xch MEM ;<br>Result: MEM ← a , a ← MEM<br>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.<br>Example: pushaf;<br>Result: [sp] ← {flag, ACC};<br>sp ← sp + 2 ;<br>Affected flags: 『N』Z 『N』C 『N』AC 『N』OV<br><br>Application Example:<br>------------------------------------------------------------------------------------------------------------------<br>.romadr 0x10 ; // ISR entry address<br>pushaf ; // put ACC and flag into stack memory<br>… // ISR program<br>… // ISR program<br>popaf ; // restore ACC and flag from stack memory<br>reti ;<br>------------------------------------------------------------------------------------------------------------------ |
| *popaf* | Restore ACC and flag from the memory which address is specified in the stack pointer.<br>Example: popaf;<br>Result: sp ← sp - 2 ;<br>{Flag, ACC} ← [sp] ;<br>Affected flags: 『Y』Z 『Y』C 『Y』AC 『Y』OV |
| *ldtabh* index | Load high byte data in OTP program memory to ACC by using index as OTP address. It needs 2T to execute this instruction.<br>Example: *ldtabh index;*<br>Result: a ← {bit 15~8 of OTP [index]};<br>Affected flags: 『N』Z 『N』C 『N』AC 『N』OV<br>Application Example:<br>------------------------------------------------------------------------------------------------------------------<br>*word ROMptr ; // declare a pointer of ROM in RAM*<br>*...*<br>*mov a, la@TableA ; // assign pointer to ROM TableA (LSB)*<br>*mov lb@ROMptr, a ; // save pointer to RAM (LSB)*<br>*mov a, ha@TableA ; // assign pointer to ROM TableA (MSB)*<br>*mov hb@ROMptr, a ; // save pointer to RAM (MSB)*<br>*...*<br>*ldtabh ROMptr ; // load TableA MSB to ACC (ACC=0X02)*<br>*...*<br>*TableA : dc 0x0234, 0x0042, 0x0024, 0x0018 ;*<br>------------------------------------------------------------------------------------------------------------------ |

| | |
|---|---|
| *ldtabl* index | Load low byte data in OTP to ACC by using index as OTP address. It needs 2T to execute this instruction.<br>Example: *ldtabl index;*<br>Result: a ← {bit7~0 of OTP [index]};<br>Affected flags: 『N』Z 『N』C 『N』AC 『N』OV<br>Application Example:<br>------------------------------------------------------------------------------------------------------------------------<br>*word ROMptr ; // declare a pointer of ROM in RAM*<br>*...*<br>*mov a, la@TableA ; // assign pointer to ROM TableA (LSB)*<br>*mov lb@ROMptr, a ; // save pointer to RAM (LSB)*<br>*mov a, ha@TableA ; // assign pointer to ROM TableA (MSB)*<br>*mov hb@ROMptr, a ; // save pointer to RAM (MSB)*<br>*...*<br>*ldtabl ROMptr ; // load TableA LSB to ACC (ACC=0x34)*<br>*...*<br>*TableA : dc 0x0234, 0x0042, 0x0024, 0x0018 ;*<br>------------------------------------------------------------------------------------------------------------------------ |

## 7.2. Arithmetic Operation Instructions

| | |
|---|---|
| *add* a, I | Add immediate data with ACC, then put result into ACC<br>Example: *add* a, 0x0f ;<br>Result: a ← a + 0fh<br>Affected flags: 『Y』Z 『Y』C 『Y』AC 『Y』OV |
| *add* a, M | Add data in memory with ACC, then put result into ACC<br>Example: *add* a, MEM ;<br>Result: a ← a + MEM<br>Affected flags: 『Y』Z 『Y』C 『Y』AC 『Y』OV |
| *add* M, a | Add data in memory with ACC, then put result into memory<br>Example: *add* MEM, a;<br>Result: MEM ← a + MEM<br>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<br>Example: *addc* a, MEM ;<br>Result: a ← a + MEM + C<br>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<br>Example: *addc* MEM, a ;<br>Result: MEM ← a + MEM + C<br>Affected flags: 『Y』Z 『Y』C 『Y』AC 『Y』OV |
| *addc* a | Add carry with ACC, then put result into ACC<br>Example: *addc* a ;<br>Result: a ← a + C<br>Affected flags: 『Y』Z 『Y』C 『Y』AC 『Y』OV |

| | |
|---|---|
| *addc* M | Add carry with memory, then put result into memory<br>Example: *addc* MEM ;<br>Result: MEM ← MEM + C<br>Affected flags: 『Y』Z 『Y』C 『Y』AC 『Y』OV |
| *nadd* a, M | Add negative logic (2's complement) of ACC with memory<br>Example: *nadd* *a, MEM ;*<br>Result: a ← 〒a + MEM<br>Affected flags: 『Y』Z 『Y』C 『Y』AC 『Y』OV |
| *nadd* M, a | Add negative logic (2's complement) of memory with ACC<br>Example: *nadd* *MEM, a ;*<br>Result: MEM ← 〒MEM + a<br>Affected flags: 『Y』Z 『Y』C 『Y』AC 『Y』OV |
| *sub* a, I | Subtraction immediate data from ACC, then put result into ACC.<br>Example: *sub* a, 0x0f;<br>Result: a ← a - 0fh ( a + [2's complement of 0fh] )<br>Affected flags: 『Y』Z 『Y』C 『Y』AC 『Y』OV |
| *sub* a, M | Subtraction data in memory from ACC, then put result into ACC<br>Example: *sub* a, MEM ;<br>Result: a ← a - MEM ( a + [2's complement of M] )<br>Affected flags: 『Y』Z 『Y』C 『Y』AC 『Y』OV |
| *sub* M, a | Subtraction data in ACC from memory, then put result into memory<br>Example: *sub* MEM, a;<br>Result: MEM ← MEM - a ( MEM + [2's complement of a] )<br>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<br>Example: *subc* a, MEM;<br>Result: a ← a – MEM - C<br>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<br>Example: *subc* MEM, a ;<br>Result: MEM ← MEM – a - C<br>Affected flags: 『Y』Z 『Y』C 『Y』AC 『Y』OV |
| *subc* a | Subtraction carry from ACC, then put result into ACC<br>Example: *subc* a;<br>Result: a ← a - C<br>Affected flags: 『Y』Z 『Y』C 『Y』AC 『Y』OV |
| *subc* M | Subtraction carry from the content of memory, then put result into memory<br>Example: *subc* MEM;<br>Result: MEM ← MEM - C<br>Affected flags: 『Y』Z 『Y』C 『Y』AC 『Y』OV |
| *inc* M | Increment the content of memory<br>Example: *inc* MEM ;<br>Result: MEM ← MEM + 1<br>Affected flags: 『Y』Z 『Y』C 『Y』AC 『Y』OV |

| | |
|---|---|
| *dec* M | Decrement the content of memory<br>Example: *dec* MEM;<br>Result: MEM ← MEM - 1<br>Affected flags: 『Y』 Z 『Y』 C 『Y』 AC 『Y』 OV |
| *clear* M | Clear the content of memory<br>Example: *clear* MEM ;<br>Result: MEM ← 0<br>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<br>Example: *sr* a ;<br>Result: a (0,b7,b6,b5,b4,b3,b2,b1) ← a (b7,b6,b5,b4,b3,b2,b1,b0), C ← a(b0)<br>Affected flags: 『N』 Z 『Y』 C 『N』 AC 『N』 OV |
| *src* a | Shift right of ACC with carry bit 7 to flag<br>Example: *src* a ;<br>Result: a (c,b7,b6,b5,b4,b3,b2,b1) ← a (b7,b6,b5,b4,b3,b2,b1,b0), C ← a(b0)<br>Affected flags: 『N』 Z 『Y』 C 『N』 AC 『N』 OV |
| *sr* M | Shift right the content of memory, shift 0 to bit 7<br>Example: *sr* MEM ;<br>Result: MEM(0,b7,b6,b5,b4,b3,b2,b1) ← MEM(b7,b6,b5,b4,b3,b2,b1,b0), C ← MEM(b0)<br>Affected flags: 『N』 Z 『Y』 C 『N』 AC 『N』 OV |
| *src* M | Shift right of memory with carry bit 7 to flag<br>Example: *src* MEM ;<br>Result: MEM(c,b7,b6,b5,b4,b3,b2,b1) ← MEM (b7,b6,b5,b4,b3,b2,b1,b0), C ← MEM(b0)<br>Affected flags: 『N』 Z 『Y』 C 『N』 AC 『N』 OV |
| *sl* a | Shift left of ACC shift 0 to bit 0<br>Example: *sl* a ;<br>Result: a (b6,b5,b4,b3,b2,b1,b0,0) ← a (b7,b6,b5,b4,b3,b2,b1,b0), C ← a (b7)<br>Affected flags: 『N』 Z 『Y』 C 『N』 AC 『N』 OV |
| *slc* a | Shift left of ACC with carry bit 0 to flag<br>Example: *slc* a ;<br>Result: a (b6,b5,b4,b3,b2,b1,b0,c) ← a (b7,b6,b5,b4,b3,b2,b1,b0), C ← a(b7)<br>Affected flags: 『N』 Z 『Y』 C 『N』 AC 『N』 OV |
| *sl* M | Shift left of memory, shift 0 to bit 0<br>Example: *sl* MEM ;<br>Result: MEM (b6,b5,b4,b3,b2,b1,b0,0) ← MEM (b7,b6,b5,b4,b3,b2,b1,b0), C ← MEM(b7)<br>Affected flags: 『N』 Z 『Y』 C 『N』 AC 『N』 OV |
| *slc* M | Shift left of memory with carry bit 0 to flag<br>Example: *slc* MEM ;<br>Result: MEM (b6,b5,b4,b3,b2,b1,b0,C) ← MEM (b7,b6,b5,b4,b3,b2,b1,b0), C ← MEM (b7)<br>Affected flags: 『N』 Z 『Y』 C 『N』 AC 『N』 OV |
| *swap* a | Swap the high nibble and low nibble of ACC<br>Example: *swap* a ;<br>Result: a (b3,b2,b1,b0,b7,b6,b5,b4) ← a (b7,b6,b5,b4,b3,b2,b1,b0)<br>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<br>Example: *and* a, 0x0f ;<br>Result: a ← a & 0fh<br>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<br>Example: *and* a, RAM10 ;<br>Result: a ← a & RAM10<br>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<br>Example: *and* MEM, a ;<br>Result: MEM ← a & MEM<br>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<br>Example: *or* a, 0x0f ;<br>Result: a ← a \| 0fh<br>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<br>Example: *or* a, MEM ;<br>Result: a ← a \| MEM<br>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<br>Example: *or* MEM, a ;<br>Result: MEM ← a \| MEM<br>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<br>Example: *xor* a, 0x0f ;<br>Result: a ← a ^ 0fh<br>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<br>Example: *xor* *pa, a* ***;***<br>Result: pa ← a ^ pa ; // pa is the data register of port A<br>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<br>Example: *xor* a, MEM ;<br>Result: a ← a ^ RAM10<br>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<br>Example: *xor* MEM, a ;<br>Result: MEM ← a ^ MEM<br>Affected flags: 『Y』 Z 『N』 C 『N』 AC 『N』 OV |

| | |
|---|---|
| *not* a | Perform 1's complement (logical complement) of ACC<br>Example: *not* a ;<br>Result: a ← ～a<br>Affected flags: 『Y』Z 『N』C 『N』AC 『N』OV<br><br>Application Example:<br>-------------------------------------------------------------------------------------------------------------------<br>*mov* a, 0x38 ; // ACC=0X38<br>*not* a ; // ACC=0XC7<br>------------------------------------------------------------------------------------------------------------------- |
| *not* M | Perform 1's complement (logical complement) of memory<br>Example: *not* MEM ;<br>Result: MEM ← ～MEM<br>Affected flags: 『Y』Z 『N』C 『N』AC 『N』OV<br><br>Application Example:<br>-------------------------------------------------------------------------------------------------------------------<br>*mov* a, 0x38 ;<br>*mov* mem, a ; // mem = 0x38<br>*not* mem ; // mem = 0xC7<br>------------------------------------------------------------------------------------------------------------------- |
| *neg* a | Perform 2's complement of ACC<br>Example: *neg* a;<br>Result: a ← ∓a<br>Affected flags: 『Y』Z 『N』C 『N』AC 『N』OV<br><br>Application Example:<br>-------------------------------------------------------------------------------------------------------------------<br>*mov* a, 0x38 ; // ACC=0X38<br>*neg* a ; // ACC=0XC8<br>------------------------------------------------------------------------------------------------------------------- |
| *neg* M | Perform 2's complement of memory<br>Example: *neg* MEM;<br>Result: MEM ← ∓MEM<br>Affected flags: 『Y』Z 『N』C 『N』AC 『N』OV<br><br>Application Example:<br>-------------------------------------------------------------------------------------------------------------------<br>*mov* a, 0x38 ;<br>*mov* mem, a ; // mem = 0x38<br>*not* mem ; // mem = 0xC8<br>------------------------------------------------------------------------------------------------------------------- |

| | |
|---|---|
| *comp* a, M | Compare ACC with the content of memory<br>Example: *comp a, MEM;*<br>Result: Flag will be changed by regarding as ( a - MEM )<br>Affected flags: 『Y』Z 『Y』C 『Y』AC 『Y』OV<br>Application Example:<br>---------------------------------------------------------------------------------------------------------------<br>*mov a, 0x38 ;*<br>*mov mem, a ;*<br>*comp a, mem ; // Z flag is set as 1*<br>*mov a, 0x42 ;*<br>*mov mem, a ;*<br>*mov a, 0x38 ;*<br>*comp a, mem ; // C flag is set as 1*<br>--------------------------------------------------------------------------------------------------------------- |
| *comp* M, a | Compare ACC with the content of memory<br>Example: *comp MEM, a;*<br>Result: Flag will be changed by regarding as ( MEM - a )<br>Affected flags: 『Y』Z 『Y』C 『Y』AC 『Y』OV |

## 7.5. Bit Operation Instructions

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

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

## 7.6. Conditional Operation Instructions

| | |
|---|---|
| *ceqsn* a, I | Compare ACC with immediate data and skip next instruction if both are equal.<br>Flag will be changed like as (a ← a - I)<br>Example: *ceqsn* a, 0x55 ;<br>*inc* MEM ;<br>*goto* error ;<br>Result: If a=0x55, then “goto error”; otherwise, “inc MEM”.<br>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.<br>Flag will be changed like as (a ← a - M)<br>Example: *ceqsn* a, MEM;<br>Result: If a=MEM, skip next instruction<br>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.<br>Flag will be changed like as (a ← a - M)<br>Example: *cneqsn a, MEM;*<br>Result: If a≠MEM, skip next instruction<br>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.<br>Flag will be changed like as (a ← a - I)<br>Example: *cneqsn a,0x55 ;*<br>*inc MEM ;*<br>*goto error ;*<br>Result: If a≠0x55, then “goto error”; Otherwise, “inc MEM”.<br>Affected flags: 『Y』Z 『Y』C 『Y』AC 『Y』OV |
| *t0sn* IO.n | Check IO bit and skip next instruction if it’s low<br>Example: *t0sn* pa.5;<br>Result: If bit 5 of port A is low, skip next instruction<br>Affected flags: 『N』Z 『N』C 『N』AC 『N』OV |
| *t1sn* IO.n | Check IO bit and skip next instruction if it’s high<br>Example: *t1sn* pa.5 ;<br>Result: If bit 5 of port A is high, skip next instruction<br>Affected flags: 『N』Z 『N』C 『N』AC 『N』OV |
| *t0sn* M.n | Check memory bit and skip next instruction if it’s low<br>Example: *t0sn* MEM.5 ;<br>Result: If bit 5 of MEM is low, then skip next instruction<br>Affected flags: 『N』Z 『N』C 『N』AC 『N』OV |
| *t1sn* M.n | Check memory bit and skip next instruction if it’s high<br>Example: *t1sn* MEM.5 ;<br>Result: If bit 5 of MEM is high, then skip next instruction<br>Affected flags: 『N』Z 『N』C 『N』AC 『N』OV |
| *izsn* a | Increment ACC and skip next instruction if ACC is zero<br>Example: *izsn* a;<br>Result: a ← a + 1,skip next instruction if a = 0<br>Affected flags: 『Y』Z 『Y』C 『Y』AC 『Y』OV |
| *dzsn* a | Decrement ACC and skip next instruction if ACC is zero<br>Example: *dzsn* a;<br>Result: A ← A - 1,skip next instruction if a = 0<br>Affected flags: 『Y』Z 『Y』C 『Y』AC 『Y』OV |
| *izsn* M | Increment memory and skip next instruction if memory is zero<br>Example: *izsn* MEM;<br>Result: MEM ← MEM + 1, skip next instruction if MEM= 0<br>Affected flags: 『Y』Z 『Y』C 『Y』AC 『Y』OV |
| *dzsn* M | Decrement memory and skip next instruction if memory is zero<br>Example: *dzsn* MEM;<br>Result: MEM ← MEM - 1, skip next instruction if MEM = 0<br>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<br>Example: *call* function1;<br>Result: [sp] ← pc + 1<br>pc ← function1<br>sp ← sp + 2<br>Affected flags: 『N』Z 『N』C 『N』AC 『N』OV |
| *goto* label | Go to specific address which can be full range address space<br>Example: *goto* error;<br>Result: Go to error and execute program.<br>Affected flags: 『N』Z 『N』C 『N』AC 『N』OV |
| *ret* I | Place immediate data to ACC, then return<br>Example: *ret* 0x55;<br>Result: A ← 55h<br>ret ;<br>Affected flags: 『N』Z 『N』C 『N』AC 『N』OV |
| *ret* | Return to program which had function call<br>Example: ret;<br>Result: sp ← sp - 2<br>pc ← [sp]<br>Affected flags: 『N』Z 『N』C 『N』AC 『N』OV |
| *reti* | Return to program from interrupt service routine. After this command is executed, global interrupt is enabled automatically.<br>Example: reti;<br>Affected flags: 『N』Z 『N』C 『N』AC 『N』OV |
| *nop* | No operation<br>Example: nop;<br>Result: nothing changed<br>Affected flags: 『N』Z 『N』C 『N』AC 『N』OV |
| *pcadd a* | Next program counter is current program counter plus ACC.<br>Example: pcadd a;<br>Result: pc ← pc + a<br>Affected flags: 『N』Z 『N』C 『N』AC 『N』OV<br><br>Application Example:<br>------------------------------------------------------------------------------------------------------------------------<br>…<br>mov a, 0x02 ;<br>pcadd a ; // PC <- PC+2<br>goto err1 ;<br>goto correct ; // jump here<br>goto err2 ;<br>goto err3 ;<br>…<br>correct: // jump here<br>…<br>------------------------------------------------------------------------------------------------------------------------ |

| | |
|---|---|
| *engint* | Enable global interrupt enable<br>Example:   *engint*;<br>Result: Interrupt request can be sent to FPP0<br>Affected flags: 『N』Z 『N』C 『N』AC 『N』OV |
| *disgint* | Disable global interrupt enable<br>Example:   *disgint* ;<br>Result: Interrupt request is blocked from FPP0<br>Affected flags: 『N』Z 『N』C 『N』AC 『N』OV |
| *stopsys* | System halt.<br>Example:   stopsys;<br>Result: Stop the system clocks and halt the system<br>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.<br>Example:   stopexe;<br>Result: Stop the system clocks and keep oscillator modules active.<br>Affected flags: 『N』Z 『N』C 『N』AC 『N』OV |
| *reset* | Reset the whole chip, its operation will be same as hardware reset.<br>Example:   reset;<br>Result: Reset the whole chip.<br>Affected flags: 『N』Z 『N』C 『N』AC 『N』OV |
| *wdreset* | Reset Watchdog timer.<br>Example:   wdreset ;<br>Result: Reset Watchdog timer.<br>Affected flags: 『N』Z 『N』C 『N』AC 『N』OV |

## 7.8. Summary of Instructions Execution Cycle

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

## 7.9. Summary of affected flags by Instructions

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

## 7.10. BIT definition

Bit defined: Only addressed at 0x00 ~ 0x7F.

## 8. Code Option Table

| Option | Selection | Description |
|---|---|---|
| Security | Enable | OTP content is protected and program cannot be read back |
| | Disable | OTP content is not protected so program can be read back |
| EMI | Disable | Disable EMI optimize option |
| | Enable | The system clock will be slightly vibrated for better EMI performance |
| LVR | 16 levels | 4.5V, 4.0V, 3.75V, 3.5V, 3.3V, 3.15V, 3.0V, 2.7V, 2.5V, 2.4V, 2.3V, 2.2V, 2.1V, 2.0V, 1.9V, 1.8V |
| ICE Set | PA7_as_CS | At ICE, use PA7 as CS pin |
| | PA5_as_CS | At ICE, use PA5 as CS pin |
| | Disable | Not use ICE |
| PB4_Drive | Normal | PB4 Drive Current is Normal |
| | **Strong (default)** | PB4 Drive Current is Strong |
| PB7_Drive | Normal | PB7 Drive Current is Normal |
| | **Strong (default)** | PB7 Drive Current is Strong |
| PWM_Source | **16MHZ (default)** | When *PWMG0C*.0= 1, PWMG0 clock source = IHRC = 16MHZ<br>When *PWMG1C*.0= 1, PWMG1 clock source = IHRC = 16MHZ<br>When *PWMG2C*.0= 1, PWMG2 clock source = IHRC = 16MHZ |
| | 32MHZ | When *PWMG0C*.0= 1, PWMG0 clock source = IHRC*2 = 32MHZ<br>When *PWMG1C*.0= 1, PWMG1 clock source = IHRC*2 = 32MHZ<br>When *PWMG2C*.0= 1, PWMG2 clock source = IHRC*2 = 32MHZ |
| GPC_PWM | **Disable (default)** | GPC / PWM are independent |
| | Enable | GPC output control PWM output |
| Comparator_Edge | **All_Edge (default)** | 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 |
| 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 |
| Interrupt Src0 | **PA.0 (default)** | *INTEN/ INTRQ*.Bit0 is from PA.0 |
| | PB.5 | *INTEN/ INTRQ*.Bit0 is from PB.5 |

| Option | Selection | Description |
|---|---|---|
| Interrupt Src1 | **PB.0 (default)** | *INTEN/ INTRQ*.Bit1 is from PB.0 |
| | PA.4 | *INTEN/ INTRQ*.Bit1 is from PA.4 |

# 9. Special Notes

This chapter is to remind user who use PMS163 IC in order to avoid frequent errors upon operation.

## 9.1. Warning

User must read all application notes of the IC by detail before using it. Please download the related application notes from the company website.

## 9.2. Using IC

### 9.2.1. IO pin usage and setting

(1) IO pin is set to be digital input

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

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

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

(3) PA5 is set to be PRSTB input pin

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

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

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

(5) In order to provide the IC with better electrical characteristics, please add a 0.1uF capacitor as close as possible to the VDD/GND of IC. And it is recommended to connect an electrolytic capacitor at least 10uF in parallel.

### 9.2.2. Interrupt

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

Step1: Set INTEN/INTEN2 register, enable the interrupt control bit.
Step2: Clear INTRQ/INTRQ2 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/INTEN2 and INTRQ/INTRQ2 have no initial values. Please set required value before enabling interrupt function.

### 9.2.3. System clock switching

System clock can be switched by CLKMD register. Please notice that, NEVER switch the system clock and turn off the original clock source at the same time. For example: When switching from clock A to clock B, please switch to clock B first; and after that turn off the clock A oscillator through CLKMD.

- Switch system clock from ILRC to IHRC/2

  ```
  CLKMD   =   0x36;          // switch to IHRC, ILRC can not be disabled here
  CLKMD.2 =    0;            // ILRC can be disabled at this time
  ```

- **ERROR.** Switch ILRC to IHRC and turn off ILRC simultaneously

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

### 9.2.4. Power down mode, wakeup and watchdog

Watchdog will be inactive once ILRC is disabled.

### 9.2.5. TIMER time out

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

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

### 9.2.6. IHRC

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

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

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

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

### 9.2.7. LVR

LVR level selection is done at compile time. User must select LVR based on the system working frequency and power supply voltage to make the MCU work stably.
The following are Suggestions for setting operating frequency, power supply voltage and LVR level:

| SYSCLK | VDD | LVR |
|---|---|---|
| 2MHz | ≧ 2.2V | ≧ 2.2V |
| 4MHz | ≧ 2.5V | ≧ 2.5V |
| 8MHz | ≧ 3.5V | ≧ 3.5V |

Table 9: LVR setting for reference

(1) The setting of LVR (1.8V ~ 4.5V) will be valid just after successful power-on process.

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

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

### 9.2.8. Programming Writing

There are 6 signals for programming PMS163: PA3, PA4, PA5, PA6, $V_{DD}$, and GND.

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

- Special notes about voltage and current while Multi-Chip-Package(MCP) or On-Board Programming

  (1) PA5 ($V_{PP}$) may be higher than 6.5V.

  (2) $V_{DD}$ may be higher than 10.0V, and its maximum current may reach about 20mA.

  (3) All other signal pins level (except GND) are the same as $V_{DD}$.

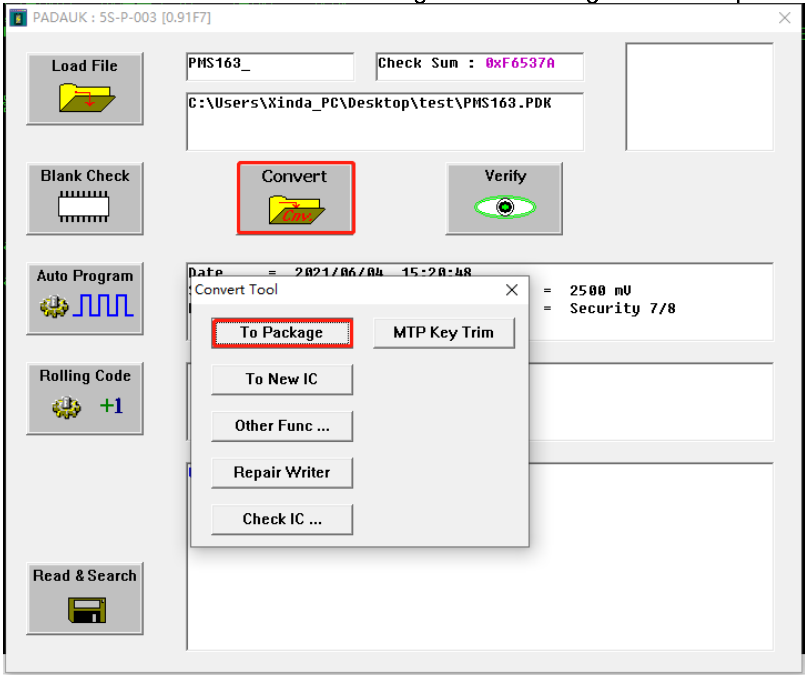
User should confirm when using this product in MCP or On-Board Programming, the peripheral circuit or components will not be destroyed or limit the above voltages.

### 9.2.8.1. Using 5S-P-003 to write PMS163

5S-P-003 writing all packages of PMS163 need special convert. Taking the writing of PMS163-S08 as an example, other packages only need to change the chip package in the “package setting” interface and jumper7 connection.

1. Convert the PDK file

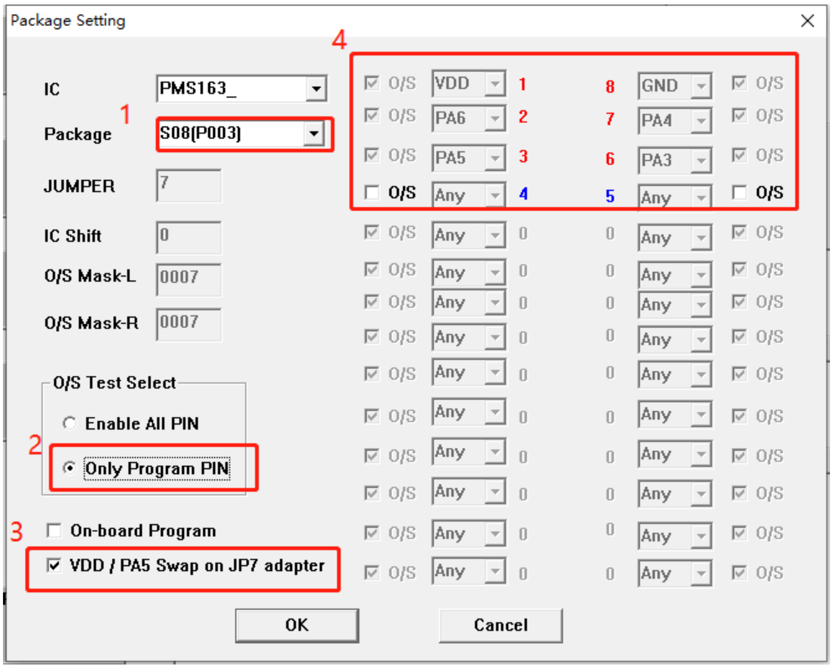
Enter the writing interface from the IDE, then click “Convert” -> “To Package”. In the “Package Setting” interface, select the package with the suffix [P003] (as shown in Figure.25), then click “Only Program PIN” and “VDD/PA5 Swap on JP7 adapter”. After confirming information about the IC pin, save and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



PMS163_
Check Sum : 0xF6537A
C:\Users\Xinda_PC\Desktop\test\PMS163.PDK

```
Date       =  2021/06/04  15:20:48
           =  2500 mV
           =  Security 7/8
```

Fig.25: convert the PDK file



IC: PMS163_
Package: S08(P003)
JUMPER: 7
IC Shift: 0
O/S Mask-L: 0007
O/S Mask-R: 0007

O/S Test Select
○ Enable All PIN
● Only Program PIN

□ On-board Program
☑ VDD / PA5 Swap on JP7 adapter

| | | | | | |
|---|---|---|---|---|---|
| ☑ O/S | VDD | 1 | 8 | GND | ☑ O/S |
| ☑ O/S | PA6 | 2 | 7 | PA4 | ☑ O/S |
| ☑ O/S | PA5 | 3 | 6 | PA3 | ☑ O/S |
| □ O/S | Any | 4 | 5 | Any | □ O/S |
| ☑ O/S | Any | 0 | 0 | Any | ☑ O/S |
| ☑ O/S | Any | 0 | 0 | Any | ☑ O/S |
| ☑ O/S | Any | 0 | 0 | Any | ☑ O/S |
| ☑ O/S | Any | 0 | 0 | Any | ☑ O/S |
| ☑ O/S | Any | 0 | 0 | Any | ☑ O/S |
| ☑ O/S | Any | 0 | 0 | Any | ☑ O/S |
| ☑ O/S | Any | 0 | 0 | Any | ☑ O/S |
| ☑ O/S | Any | 0 | 0 | Any | ☑ O/S |
| ☑ O/S | Any | 0 | 0 | Any | ☑ O/S |
| ☑ O/S | Any | 0 | 0 | Any | ☑ O/S |

Fig.26: PMS163-S08 package setting when using P003

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

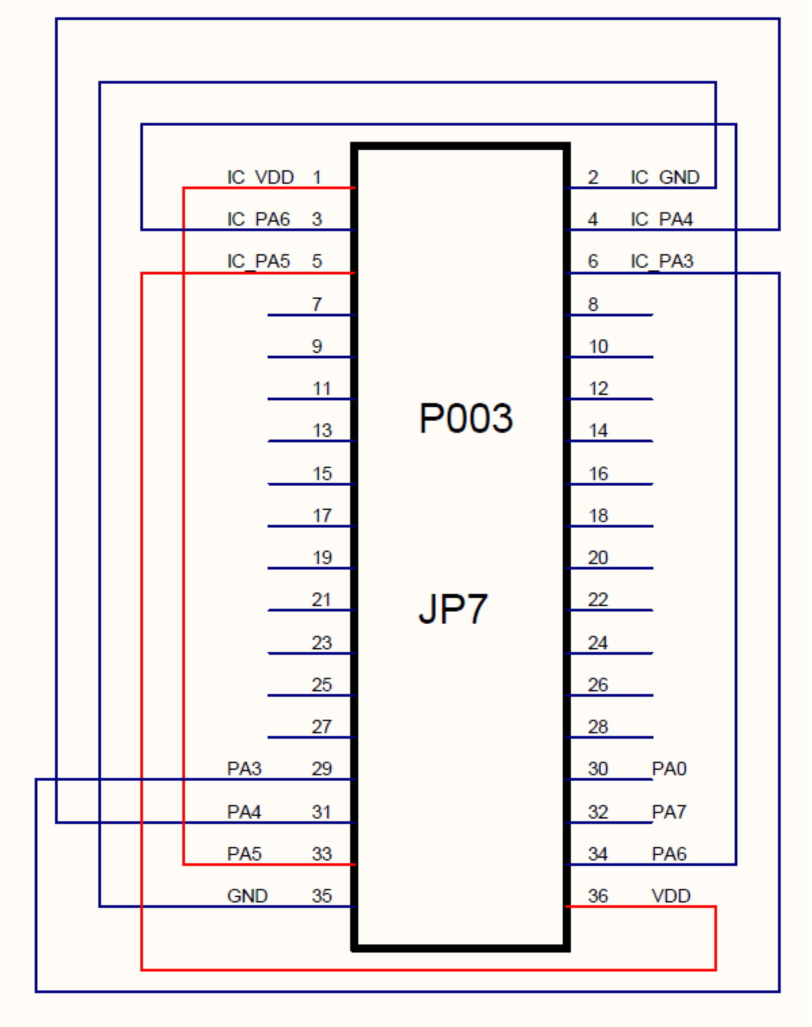


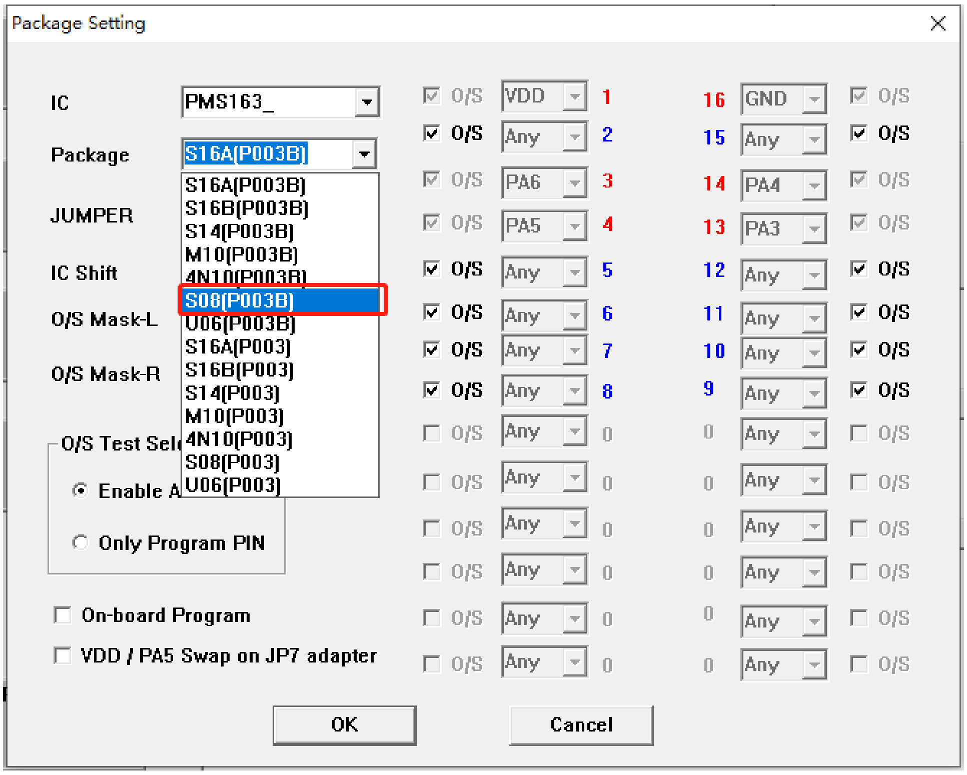
Fig.27: schematic diagram of PMS163-S08 Jumper7 when using p003

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

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

#### 9.2.8.2. Using 5S-P-003B to write PMS163

1. For 5S-P-003B to write PMS163-S16A or PMS163-S14, just use jumper2 and it needs downward four spaces on the Textool. Other packages need to convert the file and use jumper7. Taking the writing of PMS163-S08 as an example, other packages only need to change the chip package and jumper7 connection in the “package setting” interface. The package settings are shown in Figure 28:



Package Setting ×

IC: PMS163_

Package: S16A(P003B)

- S16A(P003B)
- S16B(P003B)
- S14(P003B)
- M10(P003B)
- 4N10(P003B)
- S08(P003B)
- U06(P003B)
- S16A(P003)
- S16B(P003)
- S14(P003)
- M10(P003)
- 4N10(P003)
- S08(P003)
- U06(P003)

JUMPER

IC Shift

O/S Mask-L

O/S Mask-R

O/S Test Sel

● Enable A

○ Only Program PIN

□ On-board Program

□ VDD / PA5 Swap on JP7 adapter

| | | | | | | |
|---|---|---|---|---|---|---|
| ☑ O/S | VDD | 1 | 16 | GND | ☑ O/S |
| ☑ O/S | Any | 2 | 15 | Any | ☑ O/S |
| ☑ O/S | PA6 | 3 | 14 | PA4 | ☑ O/S |
| ☑ O/S | PA5 | 4 | 13 | PA3 | ☑ O/S |
| ☑ O/S | Any | 5 | 12 | Any | ☑ O/S |
| ☑ O/S | Any | 6 | 11 | Any | ☑ O/S |
| ☑ O/S | Any | 7 | 10 | Any | ☑ O/S |
| ☑ O/S | Any | 8 | 9 | Any | ☑ O/S |
| □ O/S | Any | 0 | 0 | Any | □ O/S |
| □ O/S | Any | 0 | 0 | Any | □ O/S |
| □ O/S | Any | 0 | 0 | Any | □ O/S |
| □ O/S | Any | 0 | 0 | Any | □ O/S |
| □ O/S | Any | 0 | 0 | Any | □ O/S |
| □ O/S | Any | 0 | 0 | Any | □ O/S |

OK Cancel

Fig.28: PMS163-S08 package setting when using P003B

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

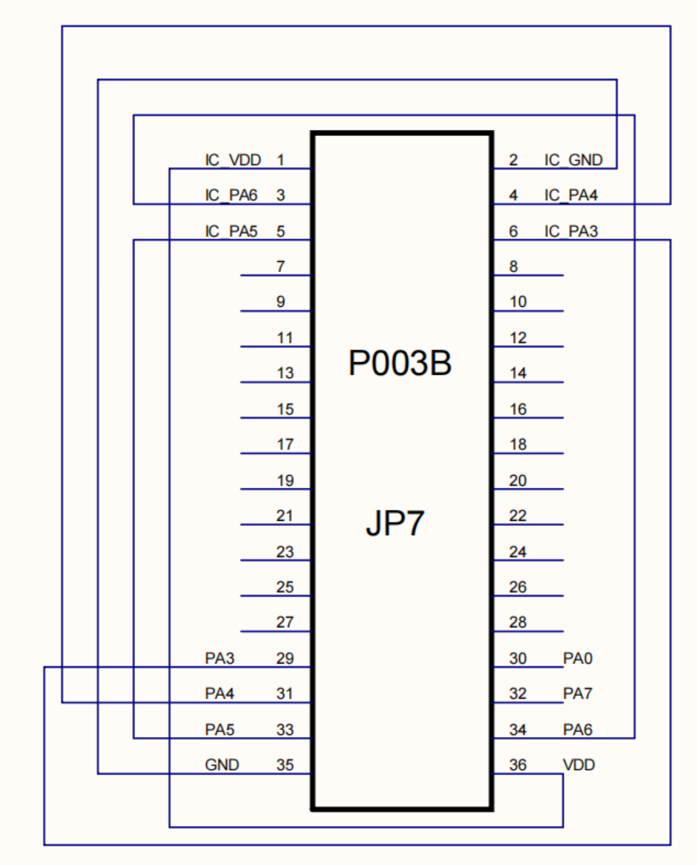


Fig.29: schematic diagram of PMS163-S08 Jumper7 when using P003B

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

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

### 9.3. Using ICE

The following items should be noted when using 5S-I-S01/2(B) or 6S-M-001 to emulate PMS163:

(1) PWMG0C output is PA6 on chip; it is PC2 in ICE.

(2) PWMG2C output is PA7on chip; it is PC0 in ICE.

(3) 5S-I-S01/2(B) or 6S-M-001 doesn’t support PWMG2C PA5 output.

(4) ICE can only select PA7 or PA5 as the CS pin through Code Option, but after selection, the pin is occupied by the simulated touch function and cannot be used. The actual IC has no effect.

(5) When simulating with the 6S-M-001 emulator, PA7 should be selected as the pin for the actual IC TK3 touch channel, and PC7 should be selected as the pin on the 6S-M-001 emulator.

(6) The 5S-I-S01/2(B) does not support the emulation pin pull-low function, but the 6S-M-001 can emulate.

(7) 5S-I-S01/2(B) doesn’t support Timer2/Timer3 function with comparator as clock source, but 6S-M-001 can emulate.

(8) 5S-I-S01/2(B) doesn’t support Timer2/Timer3 function with NILRC as clock source.

6S-M-001 ONLY supports Timer3 function with NILRC as clock source, while Timer2 cannot emulate this function.

(9) 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.

(10)Fast Wakeup time is different from 5S-I-S01/2(B): 128 SysClk, PMS163: 45 ILRC.

(11)Watch dog time out period is different from 5S-I-S01/2(B).

| WDT period | 5S-I-S01/2(B) | PMS163 |
|---|---|---|
| misc[1:0]=00 | 2048 * $T_{ILRC}$ | 8192 * $T_{ILRC}$ |
| misc[1:0]=01 | 4096 * $T_{ILRC}$ | 16384 * $T_{ILRC}$ |
| misc[1:0]=10 | 16384 * $T_{ILRC}$ | 65536 * $T_{ILRC}$ |
| misc[1:0]=11 | 256 * $T_{ILRC}$ | 262144 * $T_{ILRC}$ |