PADAUK 應廣科技

RapiDragon（迅龙）

# PFS122

## 8bit MTP Type MCU with 12-bit R-Type ADC

## Data Sheet

*Version 0.00*

*May 28, 2020*

Copyright © 2020 by PADAUK Technology Co., Ltd., all rights reserved.

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

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

# IMPORTANT NOTICE

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

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

**PADAUK Technology assumes no responsibility for any issue caused by a customer's product design. Customers should design and verify their products within the ranges guaranteed by PADAUK Technology. In order to minimize the risks in customers' products, customers should design a product with adequate operating safeguards.**

©Copyright 2020, PADAUK Technology Co. Ltd

# Table of Contents

**1. Features ..... 8**

1.1. Special Features ..... 8
1.2. System Features ..... 8
1.3. CPU Features ..... 8
1.4. Package Information ..... 8

**2. General Description and Block Diagram ..... 9**

**3. Pin Assignment and Description ..... 10**

**4. Device Characteristics ..... 16**

4.1. AC/DC Device Characteristics ..... 16
4.2. Absolute Maximum Ratings ..... 18
4.3. Typical ILRC frequency vs. VDD ..... 18
4.4. Typical IHRC frequency deviation vs. VDD (calibrated to 16MHz) ..... 18
4.5. Typical ILRC Frequency vs. Temperature ..... 19
4.6. Typical IHRC Frequency vs. Temperature (calibrated to 16MHz) ..... 19
4.7. Typical operating current vs. VDD @ system clock = ILRC/n ..... 20
4.8. Typical operating current vs. VDD @ system clock = IHRC/n ..... 20
4.9. Typical operating current vs. VDD @ system clock = 4MHz EOSC / n ..... 21
4.10. Typical operating current vs. VDD @ system clock = 32KHz EOSC / n ..... 21
4.11. Typical operating current vs. VDD @ system clock = 1MHz EOSC / n ..... 22
4.12. Typical IO driving current ($I_{OH}$) and sink current ($I_{OL}$) ..... 22
4.13. Typical IO input high/low threshold voltage ($V_{IH}$/$V_{IL}$) ..... 24
4.14. Typical resistance of IO pull high/low device ..... 25
4.15. Typical power down current ($I_{PD}$) and power save current ($I_{PS}$) ..... 26

**5. Functional Description ..... 27**

5.1. Program Memory - MTP ..... 27
5.2. Boot Procedure ..... 27
5.2.1. Timing charts for reset conditions ..... 28
5.3. Data Memory - SRAM ..... 29
5.4. Oscillator and clock ..... 29
5.4.1. Internal High RC oscillator and Internal Low RC oscillator ..... 29
5.4.2. Chip calibration ..... 29
5.4.3. IHRC Frequency Calibration and System Clock ..... 30
5.4.4. External Crystal Oscillator ..... 31

©Copyright 2020, PADAUK Technology Co. Ltd

5.4.5. System Clock and LVR level....33
5.4.6. System Clock Switching....34
5.5. Comparator....35
5.5.1 Internal reference voltage ($V_{internal\ R}$)....36
5.5.2 Using the comparator....38
5.5.3 Using the comparator and bandgap 1.20V....39
5.6 16-bit Timer (Timer16)....40
5.7 8-bit Timer (Timer2/Timer3) with PWM generation....42
5.7.1 Using the Timer2 to generate periodical waveform....43
5.7.2 Using the Timer2 to generate 8-bit PWM waveform....45
5.7.3 Using the Timer2 to generate 6-bit PWM waveform....46
5.7.4 Complementary PWM with Dead Zones....47
5.8 WatchDog Timer....50
5.9 Interrupt....50
5.10 Power-Save and Power-Down....53
5.10.1 Power-Save mode ("*stopexe*")....53
5.10.2 Power-Down mode ("*stopsys*")....54
5.10.3 Wake-up....55
5.11 IO Pins....56
5.12 Reset and LVR....57
5.12.1 Reset....57
5.12.2 LVR reset....57
5.13 Analog-to-Digital Conversion (ADC) module....57
5.13.1 The input requirement for AD conversion....58
5.13.2 ADC clock selection....59
5.13.3 Configure the analog pins....59
5.13.4 Using the ADC....59
5.13.5 How to calculate ADC input voltage $V_{IN}$....60

**6. IO Registers....61**

6.1. ACC Status Flag Register (*flag*), IO address = 0x00....61
6.2. Stack Pointer Register (*sp*), IO address = 0x02....61
6.3. Clock Mode Register (*clkmd*), IO address = 0x03....61
6.4. Interrupt Enable Register (*inten*), IO address = 0x04....62
6.5. Interrupt Request Register (*intrq*), IO address = 0x05....62
6.6. Timer16 mode Register (*t16m*), IO address = 0x06....63
6.7. External Oscillator setting Register (*eoscr*), IO address = 0x0a....63
6.8. Interrupt Edge Select Register (*integs*), IO address = 0x0c....64

©Copyright 2020, PADAUK Technology Co. Ltd

6.9. Port A Digital Input Enable Register (*padier*), IO address = 0x0d ..... 64
6.10. Port B Digital Input Enable Register (*pbdier*), IO address = 0x0e ..... 65
6.11. Port A Data Register (*pa*), IO address = 0x10 ..... 65
6.12. Port A Control Register (*pac*), IO address = 0x11 ..... 65
6.13. Port A Pull-High Register (*paph*), IO address = 0x12 ..... 65
6.14. Port A Pull-Low Register (*papl*), IO address = 0x13 ..... 65
6.15. Port B Data Register (*pb*), IO address = 0x15 ..... 65
6.16. Port B Control Register (*pbc*), IO address = 0x16 ..... 66
6.17. Port B Pull-High Register (*pbph*), IO address = 0x17 ..... 66
6.18. Port B Pull-Low Register (*pbpl*), IO address = 0x18 ..... 66
6.19. ADC Control Register (*adcc*), IO address = 0x20 ..... 66
6.20. ADC Mode Register (*adcm*), IO address = 0x21 ..... 67
6.21. ADC Result High Register (*adcrh*), IO address = 0x22 ..... 67
6.22. ADC Result Low Register (*adcrl*), IO address = 0x23 ..... 67
6.23. MISC Register (*misc*), IO address = 0x26 ..... 67
6.24. Comparator Control Register (*gpcc*), IO address = 0x2b ..... 68
6.25. Comparator Selection Register (*gpcs*), IO address = 0x2c ..... 68
6.26. Timer2 Control Register (*tm2c*), IO address = 0x30 ..... 69
6.27. Timer2 Counter Register (*tm2ct*), IO address = 0x31 ..... 69
6.28. Timer2 Scalar Register (*tm2s*), IO address = 0x32 ..... 69
6.29. Timer2 Bound Register (*tm2b*), IO address = 0x33 ..... 70
6.30. Timer3 Control Register (*tm3c*), IO address = 0x34 ..... 70
6.31. Timer3 Counter Register (*tm3ct*), IO address = 0x35 ..... 70
6.32. Timer3 Scalar Register (*tm3s*), IO address = 0x36 ..... 71
6.33. Timer3 Bound Register (*tm3b*), IO address = 0x37 ..... 71

**7. Instructions ..... 72**

7.1. Data Transfer Instructions ..... 73
7.2. Arithmetic Operation Instructions ..... 76
7.3. Shift Operation Instructions ..... 78
7.4. Logic Operation Instructions ..... 79
7.5. Bit Operation Instructions ..... 82
7.6. Conditional Operation Instructions ..... 83
7.7. System control Instructions ..... 84
7.8. Summary of Instructions Execution Cycle ..... 85
7.9. Summary of affected flags by Instructions ..... 86
7.10. BIT definition ..... 86

**8. Code Options ..... 87**

©Copyright 2020, PADAUK Technology Co. Ltd

**9. Special Notes ..... 88**

9.1. Warning ..... 88
9.2. Using IC ..... 88
9.2.1. IO pin usage and setting ..... 88
9.2.2. Interrupt ..... 89
9.2.3. System clock switching ..... 89
9.2.4. Watchdog ..... 89
9.2.5. TIMER time out ..... 90
9.2.6. IHRC ..... 90
9.2.7. LVR ..... 90
9.2.8. Programming Writing ..... 91
9.3. Using ICE ..... 93

©Copyright 2020, PADAUK Technology Co. Ltd

**Revision History:**

| Revision | Date | Description |
|---|---|---|
| 0.00 | 2020/05/28 | Preliminary version |

# 1. Features

## 1.1. Special Features

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

## 1.2. System Features

- 2KW MTP program memory (programming cycle at least 1,000 times)
- 128 Bytes data SRAM
- One hardware 16-bit timer
- Two hardware 8-bit timers with PWM generation
- One hardware comparator
- Bandgap circuit to provide 1.20V reference voltage
- Up to 12-channel 12-bit resolution R-type* ADC with one channel comes from internal bandgap voltage
  *Note: R-Type ADC means Resistive type ADC. Its linearity (INL&DNL), heat stability and power noise rejection would not be as good as the C-Type ADC which is built in the PMS13x series.
- Max. 14 IO pins with optional pull-high / pull-low resistor
- Every IO pin can be configured to enable wake-up function
- Clock sources: IHRC, ILRC and EOSC (XTAL)
- For every wake-up enabled IO, two optional wake-up speed are supported: normal and fast
- 8 selectable levels of LVR reset from 1.8V to 4.5V
- Two selectable external interrupt pins by code option

## 1.3. CPU Features

- 8-bit high performance RISC CPU
- 86 powerful instructions
- Most instructions are 1T execution cycle
- Programmable stack pointer to provide adjustable stack level
- Direct and indirect addressing modes for data access. Data memories are available for use as an index pointer of Indirect addressing mode
- IO register space and memory space are independent

## 1.4. Package Information

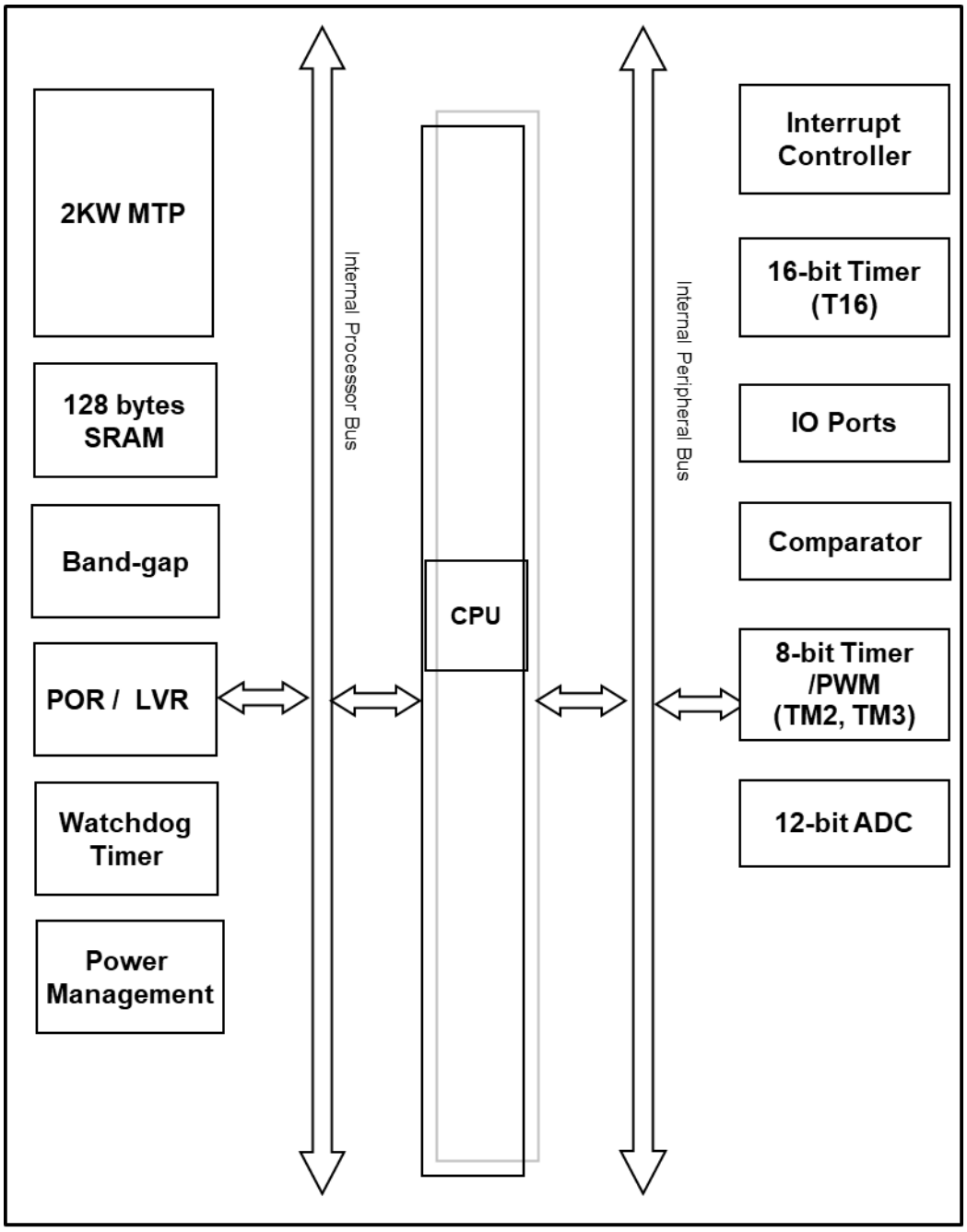
- PFS122-U06: SOT23-6 (60mil)
- PFS122-S08: SOP8 (150mil)
- PFS122-M10: MSOP10 (118mil)
- PFS122-EY10: ESSOP10 (150mil)
- PFS122-4N10: DFN3*3-10pin (0.5pitch)
- PFS122-S14: SOP14 (150mil)
- PFS122-S16A: SOP16 (150mil)
- PFS122-1J16A: QFN3*3-16pin (0.5pitch)
- PFS122-2J16A: QFN4*4-16pin (0.65pitch)

©Copyright 2020, PADAUK Technology Co. Ltd

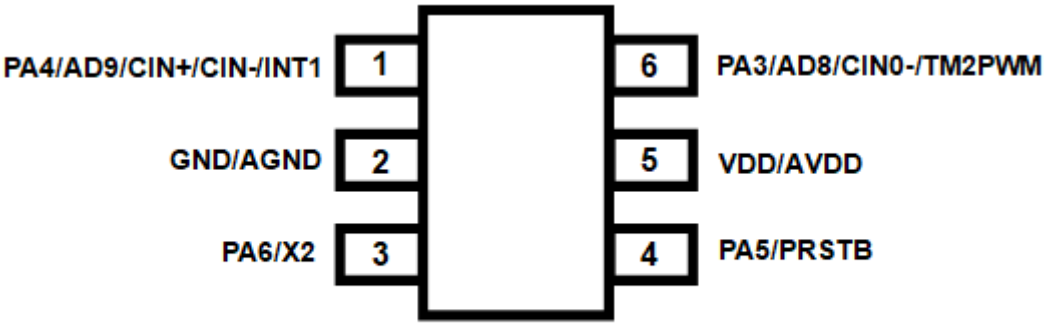
## 2. General Description and Block Diagram

The PFS122 is an MTP-based 8-bit microcontroller with 12-bit R-Type ADC. It employs RISC architecture and all the instructions are executed in one cycle except that some instructions are two cycles that handle indirect memory access.

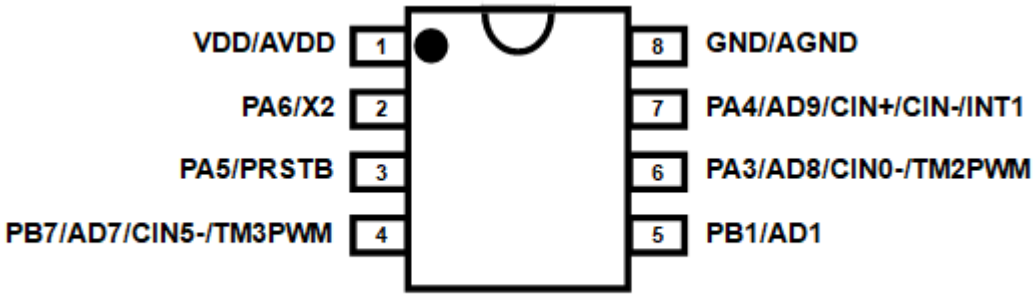
Up to 2KW MTP program memory and 128 bytes data SRAM are inside. One up to 12 channels 12-bit R-Type ADC is built inside the chip. PFS122 also provides three hardware timers: one is 16-bit timer and two are 8-bit timers which PWM generation are included. PFS122 also supports one hardware comparator.



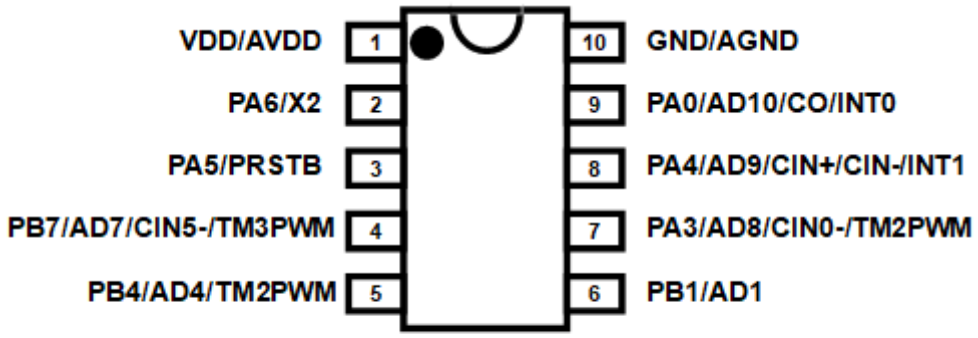
## 3. Pin Assignment and Description




PFS122-U06 (SOT23-6 60mil)

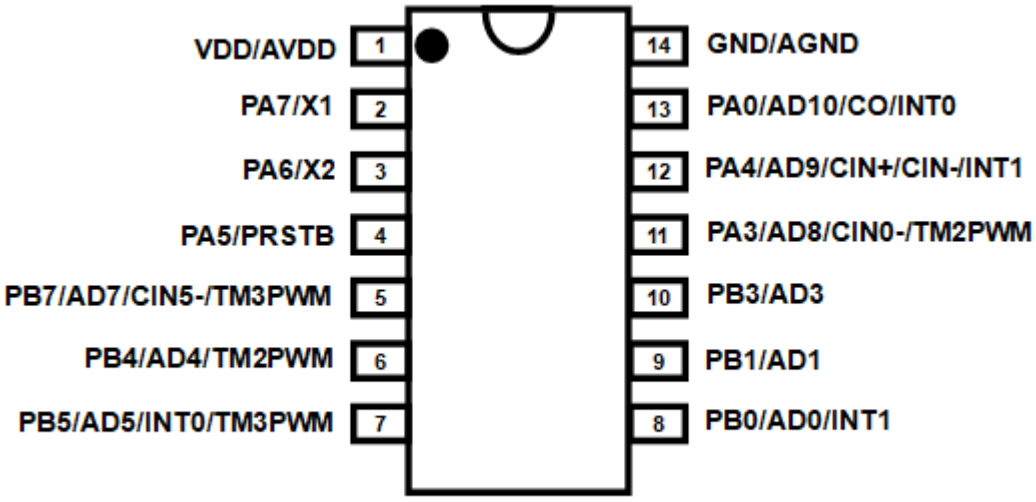



PFS122-S08 (SOP8-150mil)

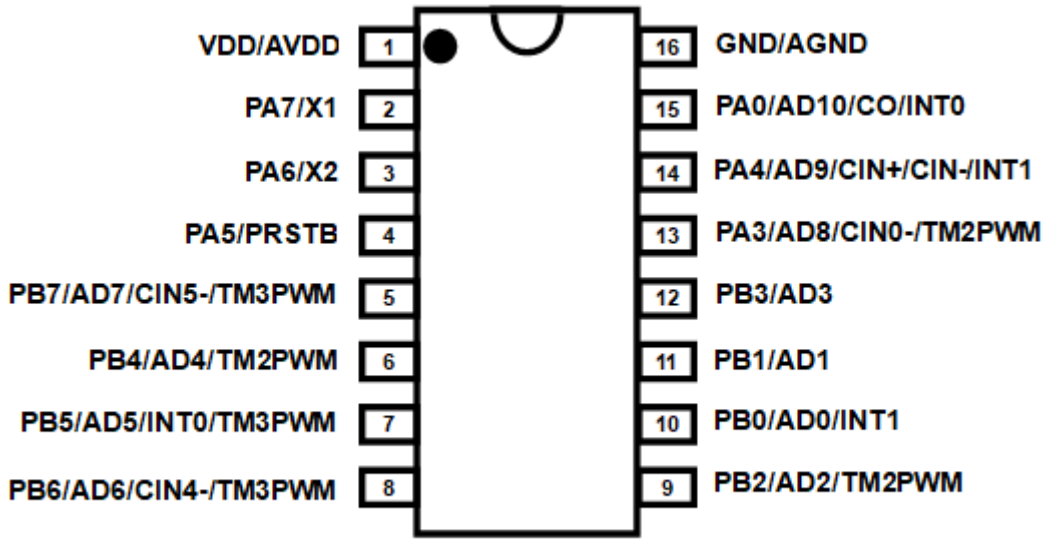



PFS122-M10 (MSOP10-118mil)
PFS122-EY10 (ESSOP10-150mil)

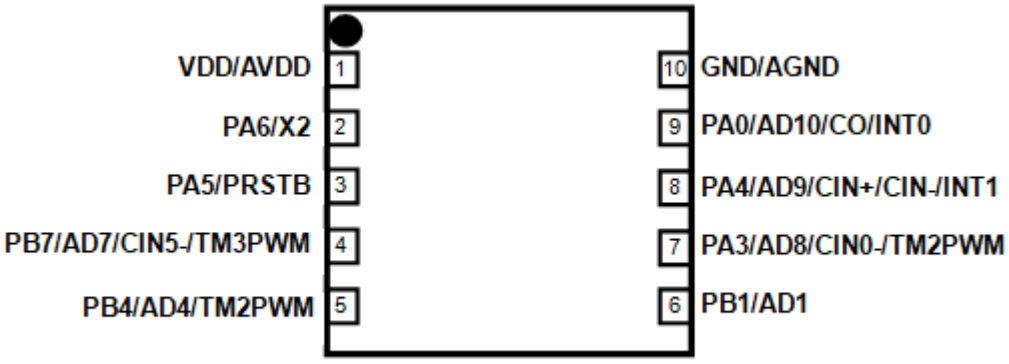
©Copyright 2020, PADAUK Technology Co. Ltd




PFS122-S14 (SOP14-150mil)

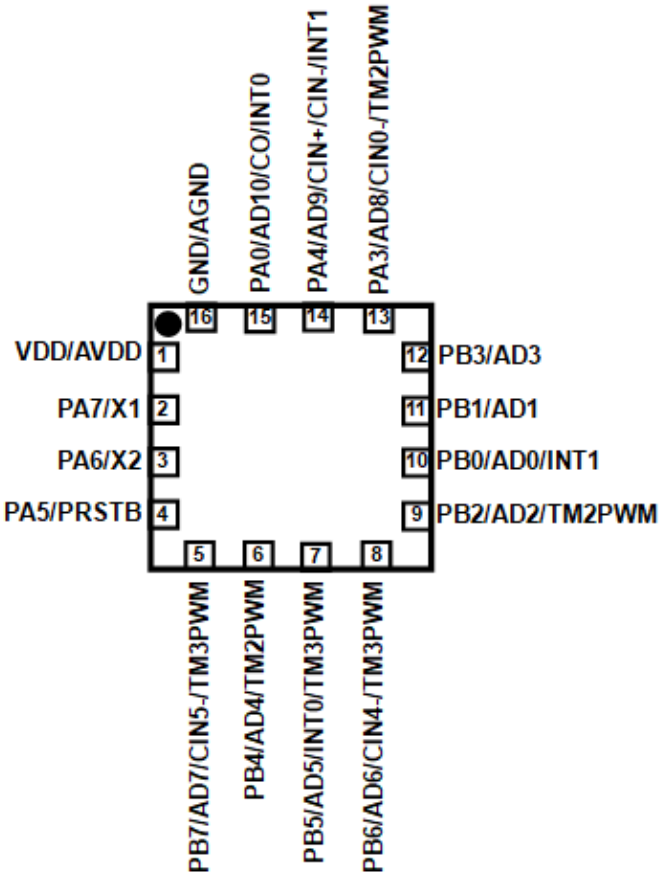



PFS122-S16A (SOP16-150mil)




PFS122-4N10 (DFN3*3-10pin-0.5pitch)

©Copyright 2020, PADAUK Technology Co. Ltd




PFS122-1J16A (QFN3*3-16pin-0.5pitch)

PFS122-2J16A (QFN4*4-16pin-0.65pitch)

| Pin Name | Pin Type & Buffer Type | Description |
|---|---|---|
| PA7 / X1 | IO ST / 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 / pull-low resistor.<br>(2) X1 is Crystal XIN(X1) when crystal oscillator is used.<br>If this pin is used for crystal oscillator, bit 7 of ***padier*** register must be programmed "0" to avoid leakage current. This pin can be used to wake-up system during sleep mode; however, wake-up function is also disabled if bit 7 of ***padier*** register is "0". |
| PA6 / X2 | IO ST / CMOS | 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 / pull-low resistor.<br>(2) X2 is Crystal XOUT(X2) when crystal oscillator is used.<br>If this pin is used for crystal oscillator, bit 6 of ***padier*** register must be programmed "0" to avoid leakage current. This pin can be used to wake-up system during sleep mode; however, wake-up function is also disabled if bit 6 of ***padier*** register is "0". |

©Copyright 2020, PADAUK Technology Co. Ltd

| Pin Name | Pin Type & Buffer Type | Description |
|---|---|---|
| PA5 / PRSTB | IO ST / CMOS | The functions of this pin can be:<br>(1) Bit 5 of port A. It can be configured as digital input, two-state output with pull-high / pull-low resistor by software independently.<br>(2) Hardware reset.<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". |
| PA4 / AD9 / CIN+ / CIN1- / INT1 | IO ST / CMOS / Analog | The functions of this pin can be:<br>(1) Bit 4 of port A. It can be configured as digital input, two-state output with pull-high / pull-low 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 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>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. |
| PA3 / AD8 / CIN0- / TM2PWM | IO ST / CMOS / Analog | The functions of this pin can be:<br>(1) Bit 3 of port A. It can be configured as digital input, two-state output with pull-high / pull-low resistor by software independently.<br>(2) Channel 8 of ADC analog input<br>(3) Minus input source 0 of comparator<br>(4) PWM output from Timer2<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 / AD10 / CO / INT0 | IO ST / CMOS / Analog | The functions of this pin can be:<br>(1) Bit 0 of port A. It can be configured as digital input, two-state output with pull-high / pull-low resistor by software independently.<br>(2) Channel 10 of ADC analog input<br>(3) Output of comparator<br>(4) 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>When this pin is configured as analog input, please use bit 0 of register ***padier*** to disable the digital input to prevent current leakage. The bit 0 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 |
|---|---|---|
| PB7 / AD7 / CIN5- / TM3PWM | IO ST / CMOS / Analog | The functions of this pin can be:<br>(1) Bit 7 of port B. It can be configured as digital input, two-state output with pull-high / pull-low resistor by software independently.<br>(2) Channel 7 of ADC analog input<br>(3) Minus input source 5 of comparator<br>(4) PWM output from Timer3<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. |
| PB6 / AD6 / CIN4- / TM3PWM | IO ST / CMOS / Analog | The functions of this pin can be:<br>(1) Bit 6 of port B. It can be configured as digital input, two-state output with pull-high / pull-low resistor by software independently.<br>(2) Channel 6 of ADC analog input<br>(3) Minus input source 4 of comparator<br>(4) PWM output from Timer3<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 / AD5 / INT0 / TM3PWM | IO ST / CMOS / Analog | The functions of this pin can be:<br>(1) Bit 5 of port B. It can be configured as digital input, two-state output with pull-high / pull-low resistor by software independently.<br>(2) Channel 5 of ADC analog input<br>(3) 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>(4) PWM output from Timer3<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 / AD4 / TM2PWM | IO ST / CMOS / Analog | The functions of this pin can be:<br>(1) Bit 4 of port B. It can be configured as digital input, two-state output with pull-high / pull-low resistor by software independently.<br>(2) Channel 4 of ADC analog input<br>(3) PWM output from Timer2<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 / AD3 | IO ST / CMOS / Analog | The functions of this pin can be:<br>(1) Bit 3 of port B. It can be configured as digital input, two-state output with pull-high / pull-low resistor by software independently.<br>(2) Channel 3 of ADC analog input<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 / AD2 / TM2PWM | IO ST / CMOS / Analog | The functions of this pin can be:<br>(1) Bit 2 of port B. It can be configured as digital input, two-state output with pull-high / pull-low resistor by software independently.<br>(2) Channel 2 of ADC analog input<br>(3) PWM output from Timer2<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 / AD1 | IO ST / CMOS / Analog | The functions of this pin can be:<br>(1) Bit 1 of port B. It can be configured as digital input, two-state output with pull-high / pull-low resistor by software independently.<br>(2) Channel 1 of ADC analog input<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. |
| PB0 / AD0 / INT1 | IO ST / CMOS / Analog | The functions of this pin can be:<br>(1) Bit 0 of port B. It can be configured as digital input, two-state output with pull-high / pull-low resistor by software independently.<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>When this pin is configured as analog input, please use bit 0 of register ***pbdier*** to disable the digital input to prevent current leakage. The bit 0 of ***pbdier*** register can be set to "0" to disable digital input; wake-up from power-down by toggling this pin is also disabled. |
| VDD / AVDD | VDD / 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 / AGND | GND / 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; **Analog**: Analog input pin; **CMOS**: CMOS voltage level

©Copyright 2020, PADAUK Technology Co. Ltd

# 4. Device Characteristics

## 4.1. AC/DC Device 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 (Ta=25°C) |
|---|---|---|---|---|---|---|
| $V_{DD}$ | Operating Voltage | 1.8* | 5.0 | 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>56K | <br>8M<br>4M<br>2M | Hz | <br>$V_{DD} \geqq$ 3.0V<br>$V_{DD} \geqq$ 2.2V<br>$V_{DD} \geqq$ 1.8V<br>$V_{DD}$ = 5.0V |
| $P_{cycle}$ | Program cycle | 1000 | | | cycles | |
| $I_{OP}$ | Operating Current | | 0.6<br>76 | | mA<br>uA | $f_{SYS}$=IHRC/16=1MIPS@5.0V<br>$f_{SYS}$=ILRC |
| $I_{PD}$ | Power Down Current<br>(by ***stopsys*** command) | | 0.9<br>0.6 | | uA<br>uA | $f_{SYS}$= 0Hz,$V_{DD}$=5.0V<br>$f_{SYS}$= 0Hz,$V_{DD}$=3.3V |
| $I_{PS}$ | Power Save Current<br>(by ***stopexe*** command) | | 3.1 | | uA | $V_{DD}$ =5.0V; $f_{SYS}$= ILRC<br>Only ILRC module is enabled. |
| $V_{IL}$ | Input low voltage for IO lines | 0 | | 0.2 $V_{DD}$ | V | |
| $V_{IH}$ | Input high voltage for IO lines | 0.7 $V_{DD}$ | | $V_{DD}$ | V | |
| $I_{OL}$ | IO lines Sink current | | | | | |
| | PB4, PB7 (Strong)<br>PB4, PB7 (Normal)<br>Other IOs | | 35<br>21<br>22 | | mA | $V_{DD}$=5.0V, $V_{OL}$=0.5V |
| $I_{OH}$ | IO lines Drive current | | | | | |
| | PB4, PB7 (Strong)<br>PB4, PB7 (Normal)<br>PA5<br>Other IOs | | 23<br>11<br>12<br>11 | | mA | $V_{DD}$=5.0V, $V_{OH}$=4.5V |
| $V_{IN}$ | Input voltage | -0.3 | | $V_{DD}$+0.3 | V | |
| $I_{INJ (PIN)}$ | Injected current on pin | | | 1 | mA | $V_{DD}$ +0.3$\geqq V_{IN} \geqq$ -0.3 |
| $R_{PH}$ | Pull-high Resistance | | 45<br>86<br>71 | | KΩ | PB1/PB4 @$V_{DD}$ =5.0V<br>PB7@$V_{DD}$ =5.0V<br>Other IO |
| $R_{PL}$ | Pull-low Resistance | | 45<br>86<br>71 | | KΩ | PB1/PB4 @$V_{DD}$ =5.0V<br>PB7 @$V_{DD}$ =5.0V<br>Other IO |
| $V_{BG}$ | Bandgap Reference Voltage | 1.145* | 1.20* | 1.255* | V | $V_{DD}$ =1.8V ~ 5.5V<br>-20°C <Ta<70°C* |

| Symbol | Description | Min | Typ | Max | Unit | Conditions (Ta=25°C) |
|---|---|---|---|---|---|---|
| $f_{IHRC}$ | Frequency of IHRC after calibration * | 15.76* | 16* | 16.24* | MHz | 25°C, $V_{DD}$ =2.0V~5.5V |
| | | 15.20* | | 16.80* | | $V_{DD}$ =2.0V~5.5V, -20°C <Ta<70°C* |
| | | 13.60* | | 18.40* | | $V_{DD}$ =1.8V~5.5V, -20°C <Ta<70°C* |
| $t_{INT}$ | Interrupt pulse width | 30 | | | ns | $V_{DD}$ = 5.0V |
| $V_{AD}$ | AD Input Voltage | 0 | | $V_{DD}$ | V | |
| ADrs | ADC resolution | | 12<br>10 | | bit | 0°C <Ta<50°C*<br>-20°C <Ta<70°C* |
| ADcs | ADC current consumption | | 0.9<br>0.8 | | mA | @5V<br>@3V |
| ADclk | ADC clock period | | 2 | | us | 1.8V ~ 5.5V |
| $t_{ADCONV}$ | ADC conversion time ($t_{ADCLK}$ is the period of the selected AD conversion clock) | | 16 | | $t_{ADCLK}$ | 12-bit resolution |
| AD DNL | ADC Differential Non-Linearity | | ±4* | | LSB | 12-bit resolution LSB |
| AD INL | ADC Integral Non-Linearity | | ±8* | | LSB | 12-bit resolution LSB |
| ADos | ADC offset | | 5* | | mV | @ $V_{DD}$ =3V |
| $V_{DR}$ | RAM data retention voltage* | 1.5 | | | V | in stop mode |
| $t_{WDT}$ | Watchdog timeout period | | 8k | | $T_{ILRC}$ | misc[1:0]=00 (default) |
| | | | 16k | | | misc[1:0]=01 |
| | | | 64k | | | misc[1:0]=10 |
| | | | 256k | | | misc[1:0]=11 |
| $t_{WUP}$ | Wake-up time period (fast) | | 45 | | $T_{ILRC}$ | Where $T_{ILRC}$ is the time period of ILRC |
| | Wake-up time period (slow) | | 3000 | | | |
| $t_{SBP}$ | System boot-up period from power-on for Slow boot-up | | 50 | | ms | $V_{DD}$ =5V |
| | System boot-up period from power-on for Fast boot-up | | 750 | | us | $V_{DD}$ =5V |
| $t_{RST}$ | External reset pulse width | 120 | | | us | @ $V_{DD}$ =5V |
| CPos | Comparator offset* | - | ±10 | ±20 | mV | |
| CPcm | Comparator input common mode* | 0 | | $V_{DD}$ -1.5 | V | |
| CPspt | Comparator response time* | | 100 | 500 | ns | Both Rising and Falling |
| CPmc | Stable time to change comparator mode | | 2.5 | 7.5 | us | |
| CPcs | Comparator current consumption | | 20 | | uA | $V_{DD}$ = 3.3V |

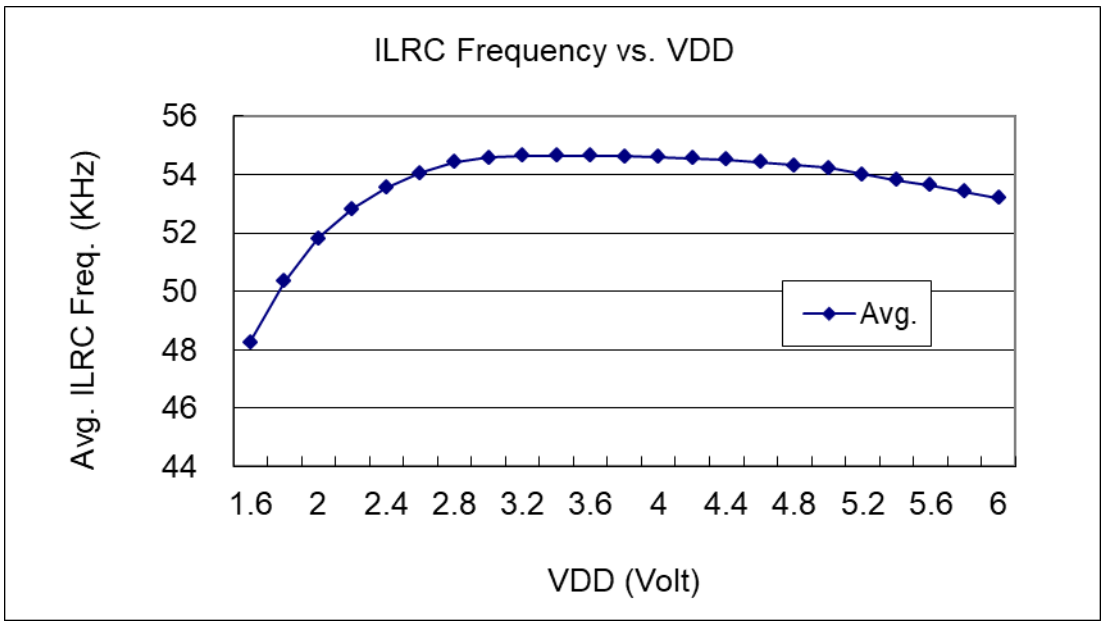
*These parameters are for design reference, not tested for each chip.

©Copyright 2020, PADAUK Technology Co. Ltd

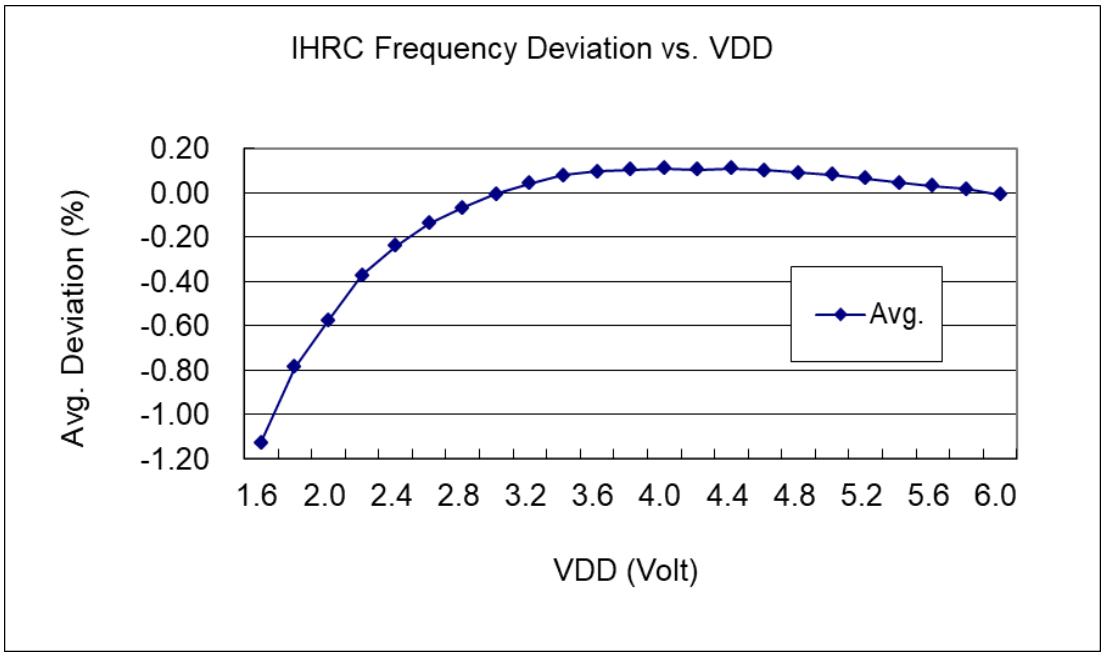
## 4.2. Absolute Maximum Ratings

- Supply Voltage ……..... 1.8V ~ 5.5V
  *If $V_{DD}$ is over the maximum rating, it may lead to a permanent damage of IC.
- Input Voltage …….. -0.3V ~ $V_{DD}$ + 0.3V
- Operating Temperature ..... -20°C ~ 70°C
- Junction Temperature …… 150°C
- Storage Temperature …….. -50°C ~ 125°C

## 4.3. Typical ILRC frequency vs. VDD

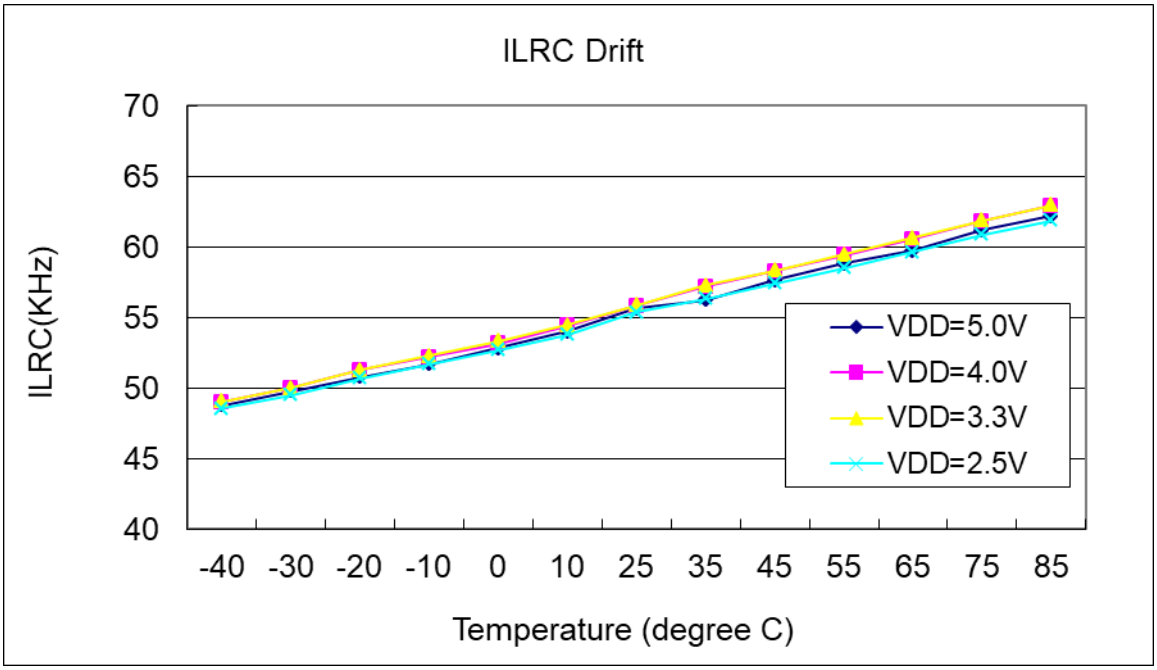



## 4.4. Typical IHRC frequency deviation vs. VDD (calibrated to 16MHz)

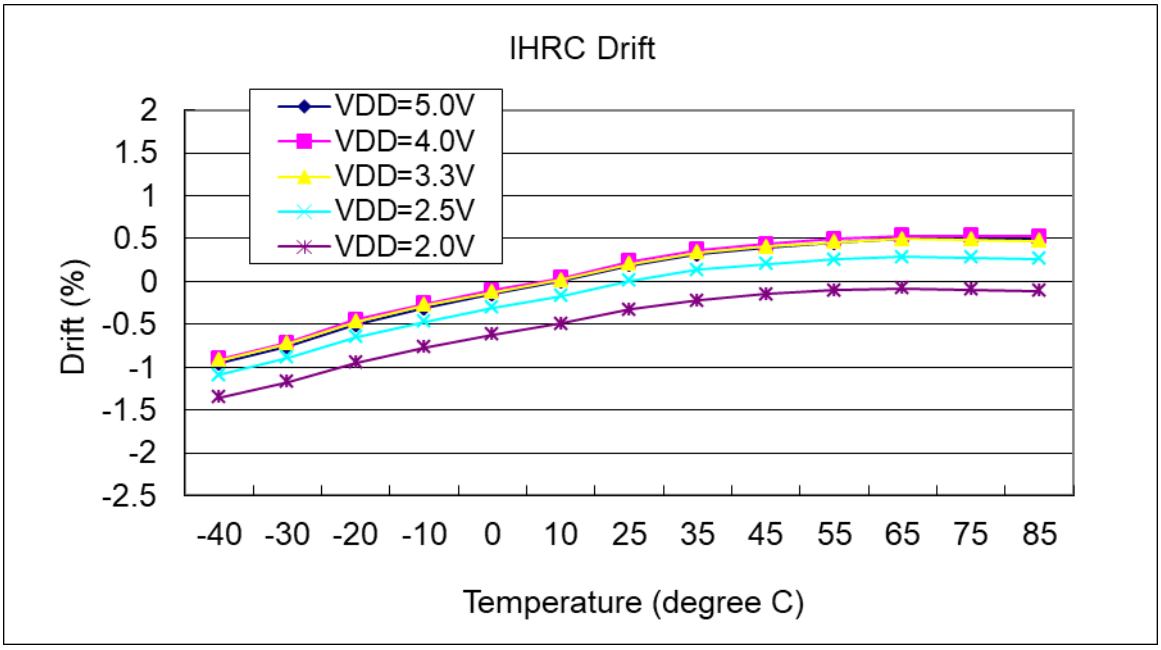



©Copyright 2020, PADAUK Technology Co. Ltd

### 4.5. Typical ILRC Frequency vs. Temperature




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

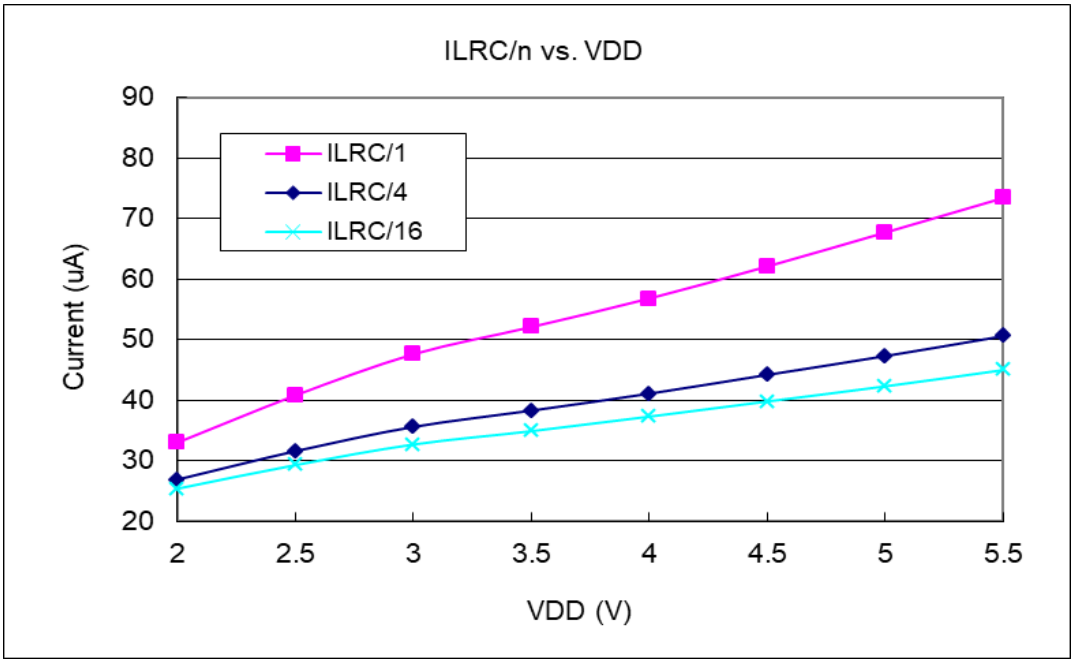


## 4.7. Typical operating current vs. VDD @ system clock = ILRC/n

Conditions:

**ON**: ILRC, Bandgap, LVR; **OFF**: IHRC, EOSC, T16, TM2, TM3, ADC modules;

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

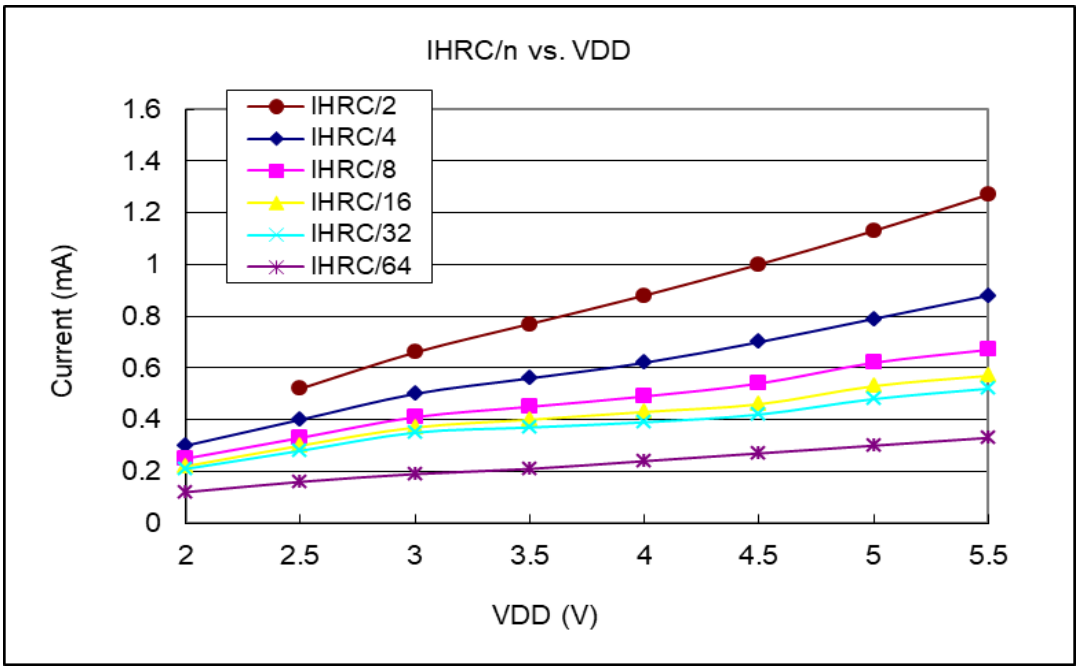



## 4.8. Typical operating current vs. VDD @ system clock = IHRC/n

Conditions:

**ON**: IHRC, Bandgap, LVR; **OFF**: ILRC, EOSC, T16, TM2, TM3, ADC modules;

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

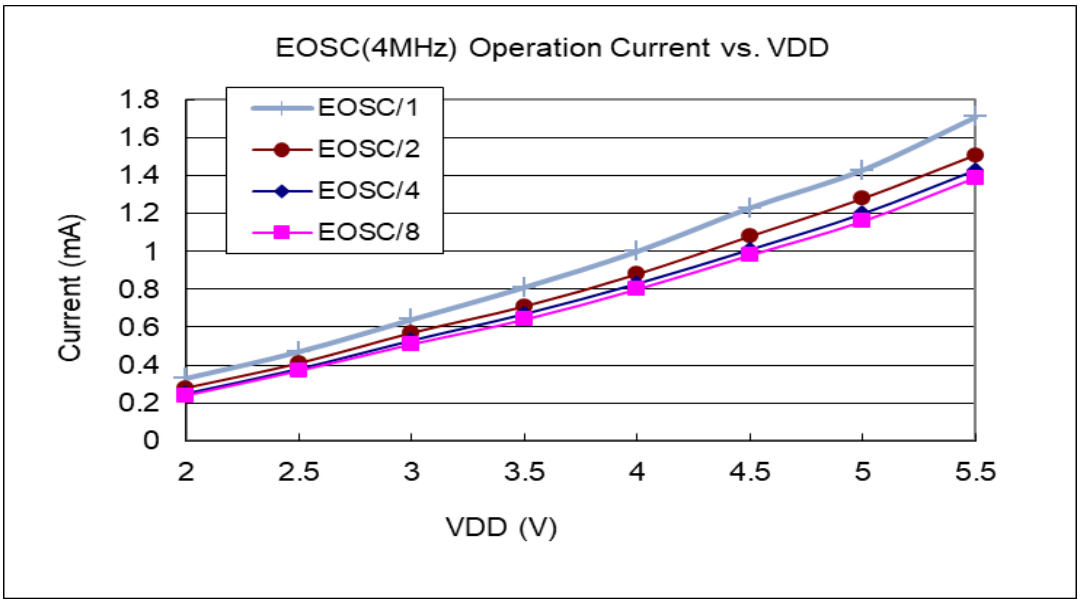


## 4.9. Typical operating current vs. VDD @ system clock = 4MHz EOSC / n

Conditions:

**ON**: EOSC[6,5] = [1,1], Bandgap, LVR; **OFF**: IHRC, ILRC, T16, TM2, TM3, ADC modules;

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

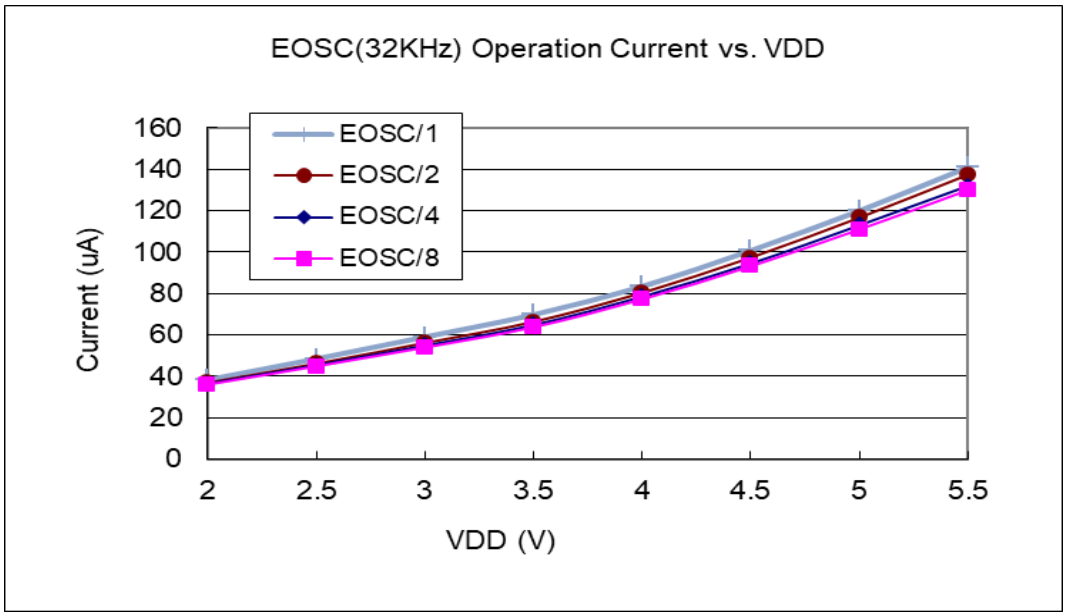



## 4.10. Typical operating current vs. VDD @ system clock = 32KHz EOSC / n

Conditions:

**ON**: EOSC[6,5] = [0,1], Bandgap, LVR; **OFF**: IHRC, ILRC, T16, TM2, TM3, ADC modules;

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

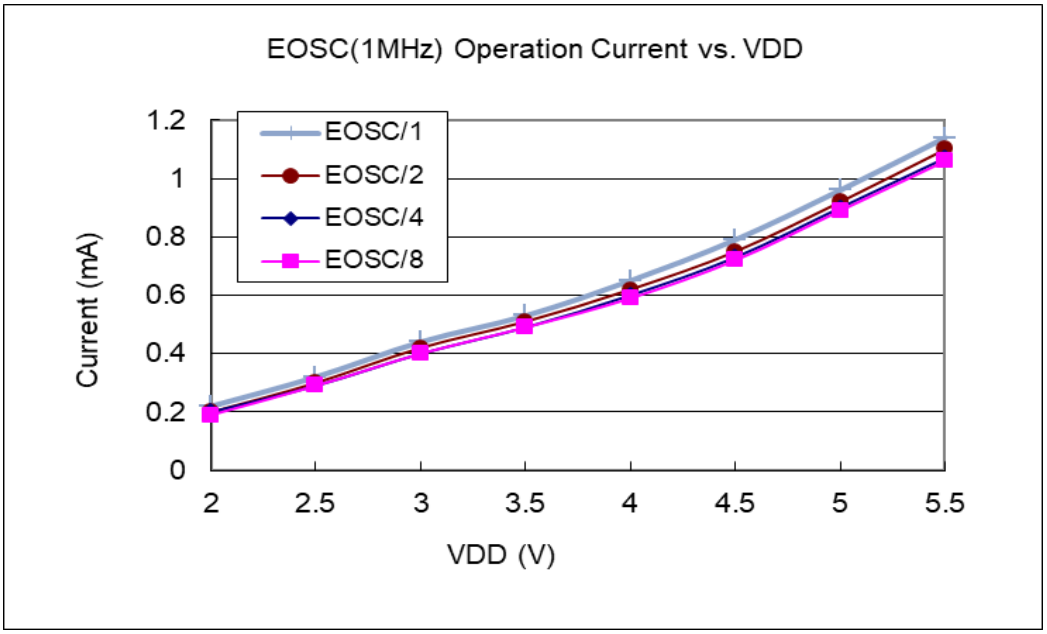


## 4.11.Typical operating current vs. VDD @ system clock = 1MHz EOSC / n

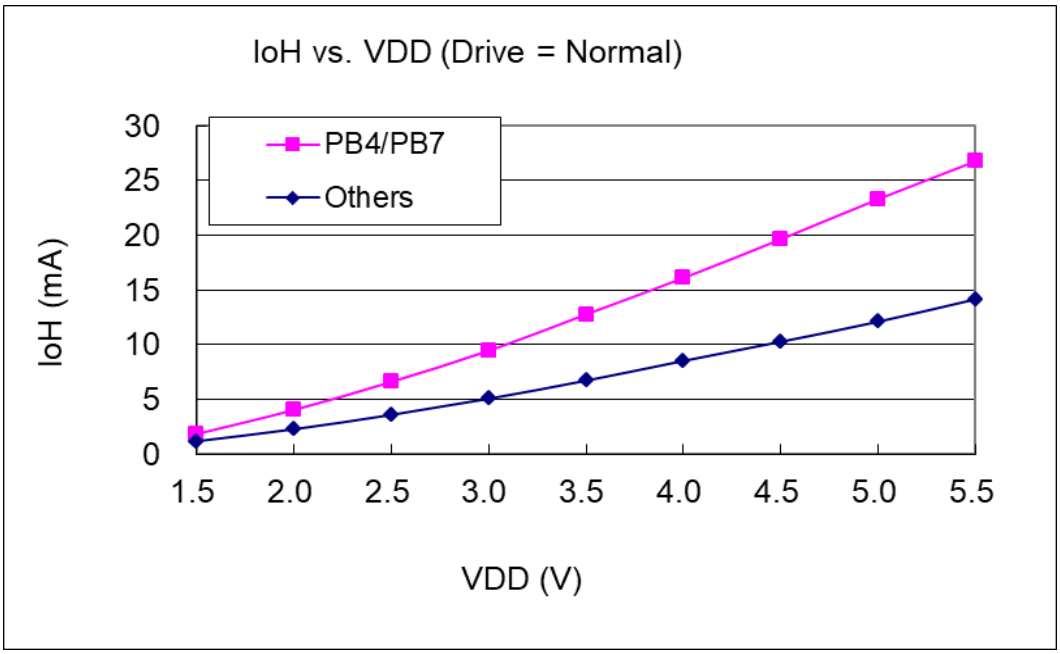
Conditions:

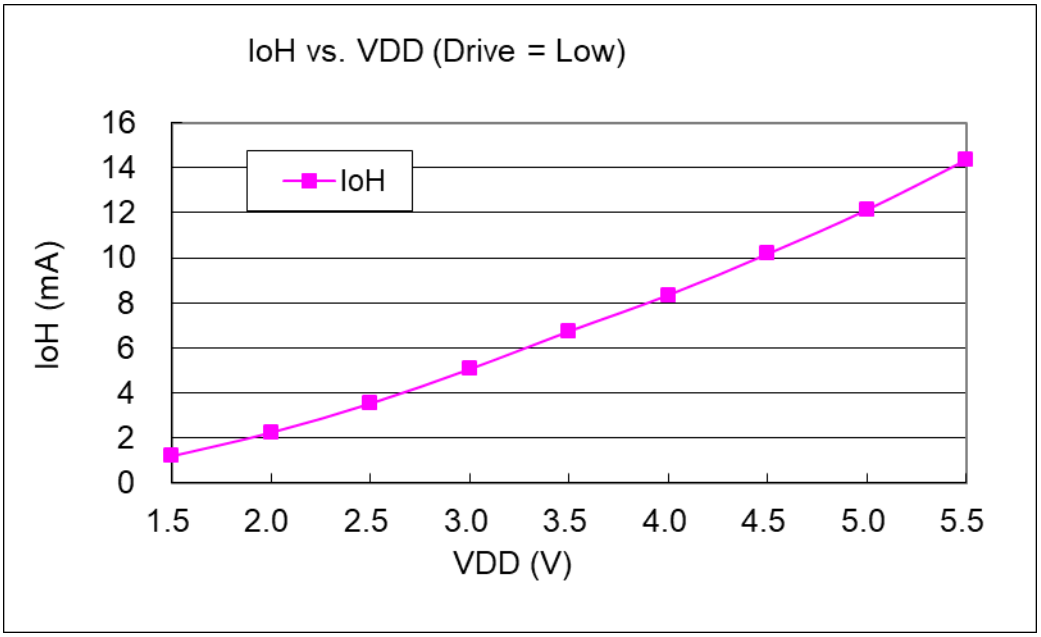
**ON**: EOSC[6,5] = [1,0], Bandgap, LVR; **OFF**: IHRC, ILRC, T16, TM2, TM3, ADC modules;

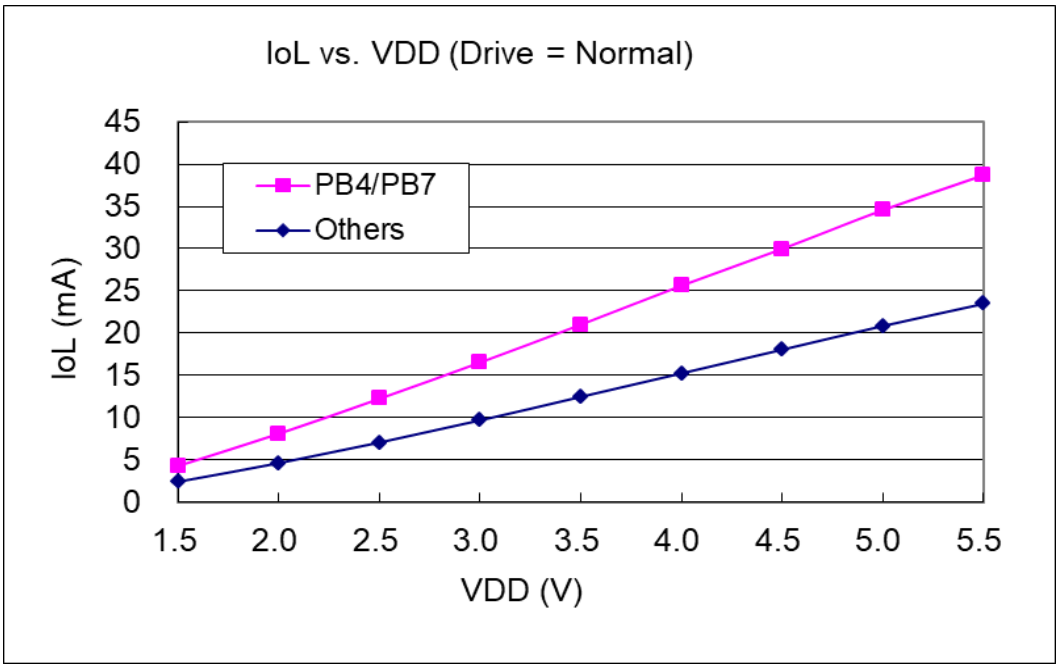
**IO**: PA0:0.5Hz output toggle and no loading, **others**: input and no floating

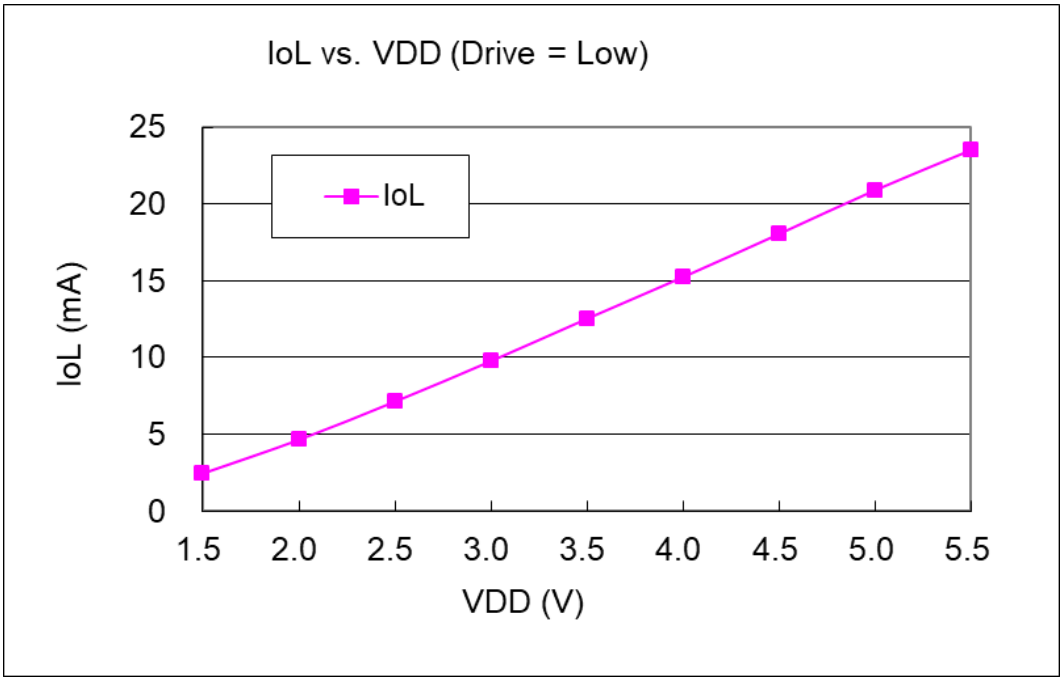



## 4.12.Typical IO driving current ($I_{OH}$) and sink current ($I_{OL}$)

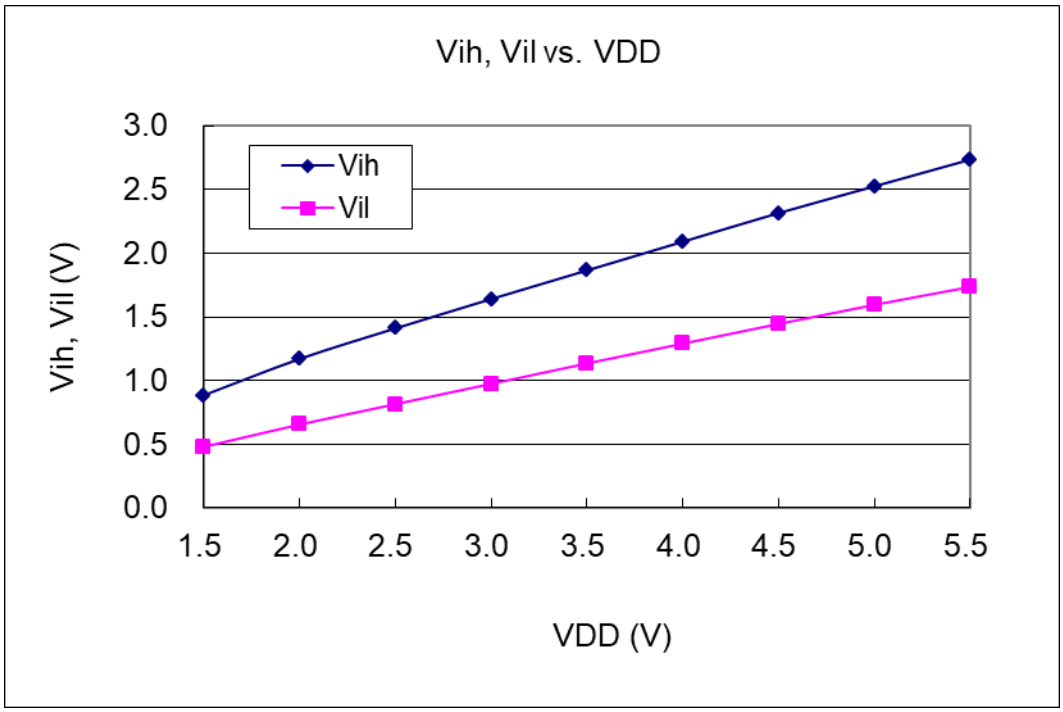



IoH vs. VDD (Drive = Low)
IoH
IoH (mA)
16
14
12
10
8
6
4
2
0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
5.5
VDD (V)

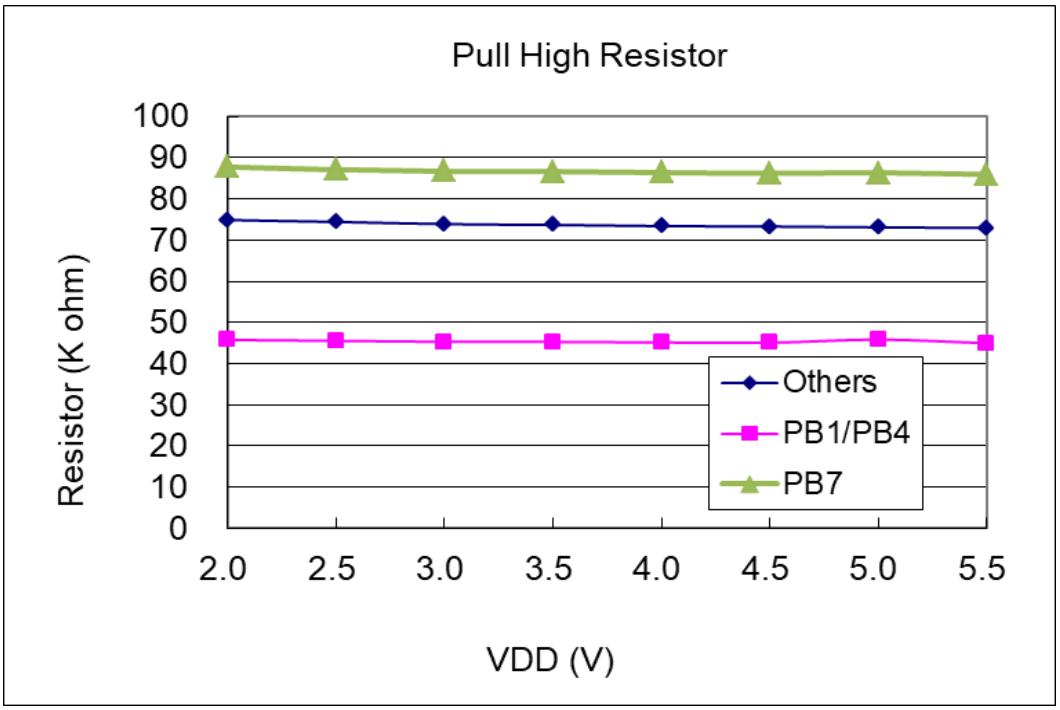

IoL vs. VDD (Drive = Normal)
PB4/PB7
Others
IoL (mA)
45
40
35
30
25
20
15
10
5
0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
5.5
VDD (V)

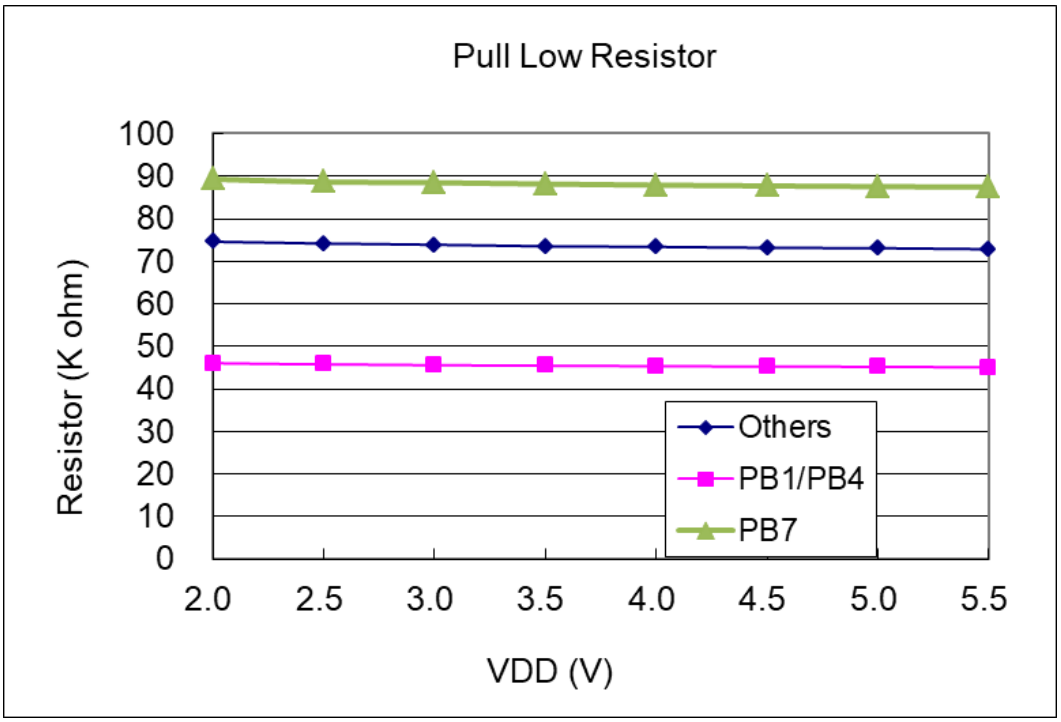



## 4.13. Typical IO input high/low threshold voltage ($V_{IH}/V_{IL}$)

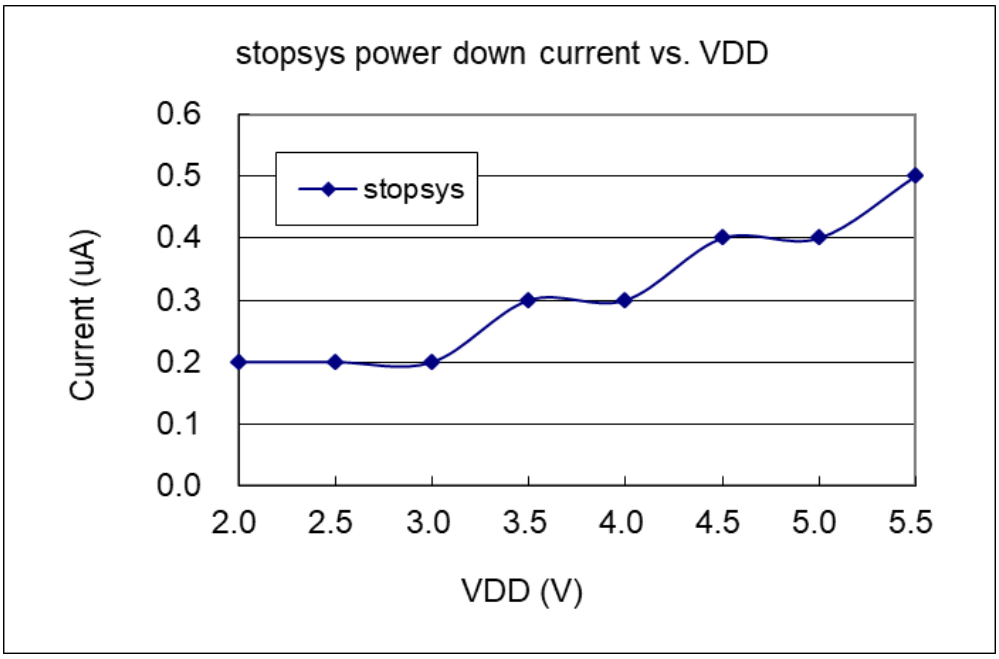


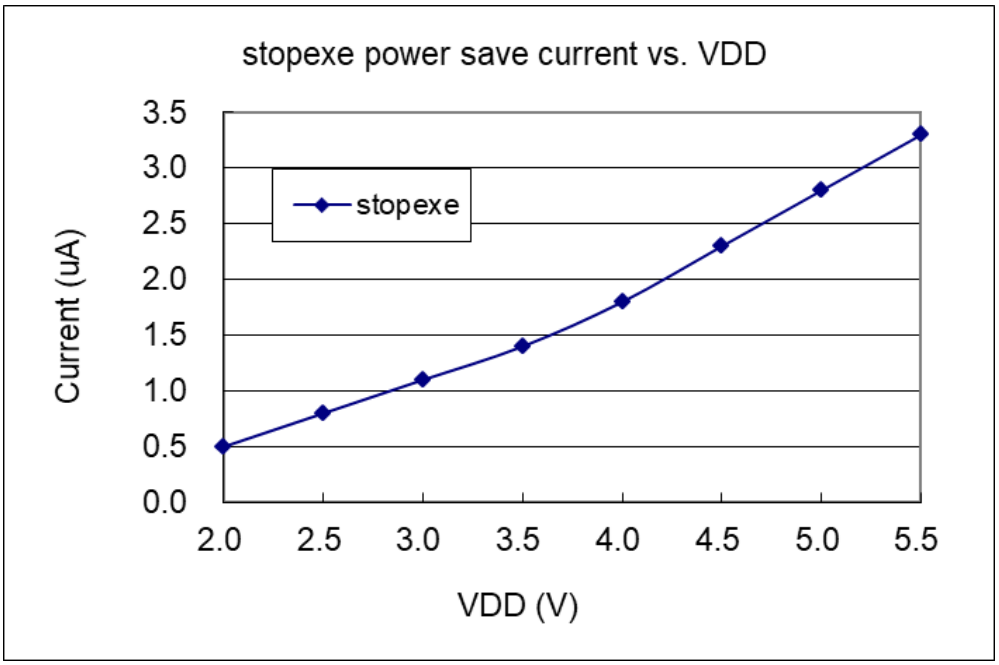
### 4.14. Typical resistance of IO pull high/low device



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



# 5. Functional Description

## 5.1. Program Memory - MTP

The MTP (Multiple Time Programmable) program memory is used to store the program instructions to be executed. The MTP program memory may contains the data, tables and interrupt entry. After reset, the program will start from the initial address 0x000 which is GOTO FPPA0 instruction usually. The interrupt entry is 0x10 if used, the last 32 addresses are reserved for system using, like checksum, serial number, etc. The MTP program memory for PFS122 is 2KW that is partitioned as Table 1. The MTP memory from address 0X7E0 to 0x7FF is for system using, address space from 0x001 to 0x00F and from 0x011 to 0X7DF are user program spaces.

| Address | Function |
|---|---|
| 0x000 | GOTO FPPA0 instruction |
| 0x001 | User program |
| • | • |
| 0x00F | User program |
| 0x010 | Interrupt entry address |
| 0x011 | User program |
| • | • |
| 0x7DF | User program |
| 0X7E0 | System Using |
| • | • |
| 0x7FF | System Using |

Table 1: Program Memory Organization

## 5.2. Boot Procedure

POR (Power-On-Reset) is used to reset PFS122 when power up. The boot up time can be optional fast or normal. Customer must ensure the stability of supply voltage after power up no matter which option is chosen, the power up sequence is shown in the Fig. 1 and $t_{SBP}$ is the boot up time.

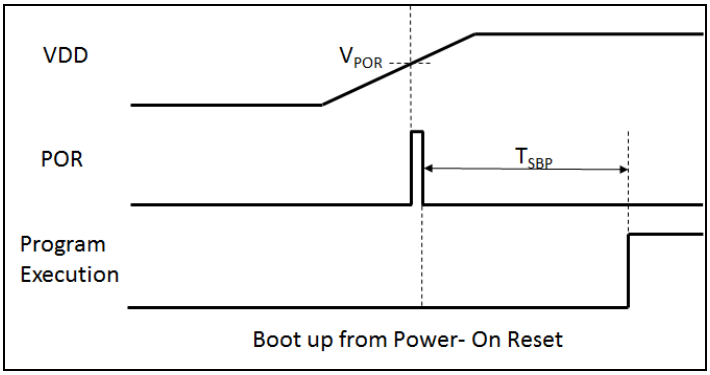
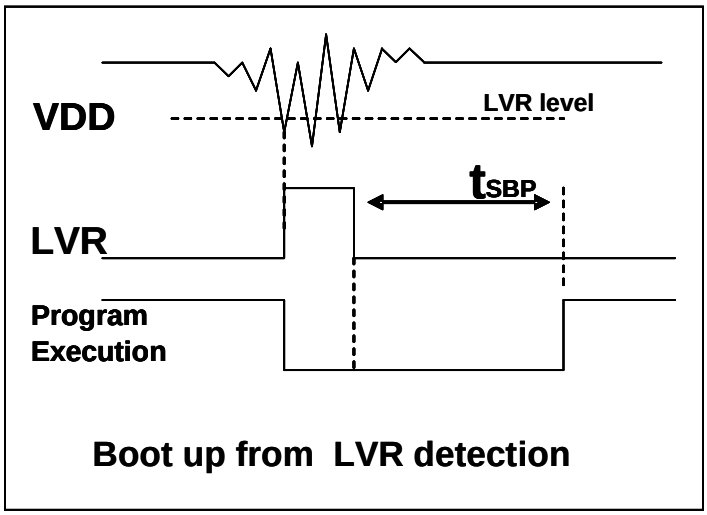
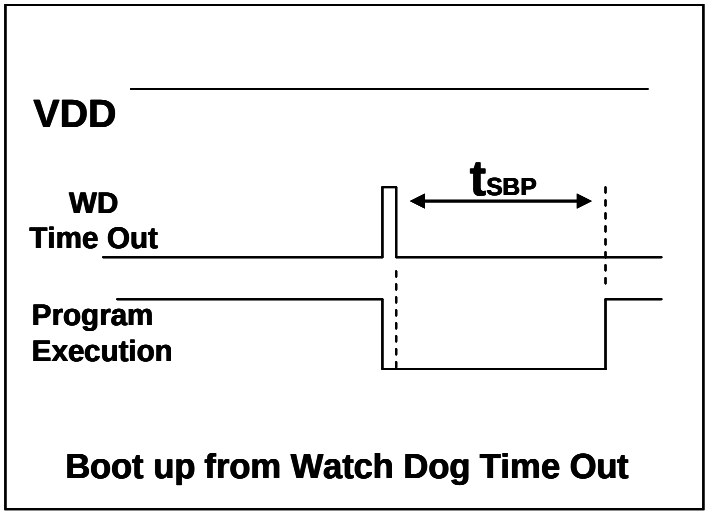



Fig.1: Power-Up Sequence

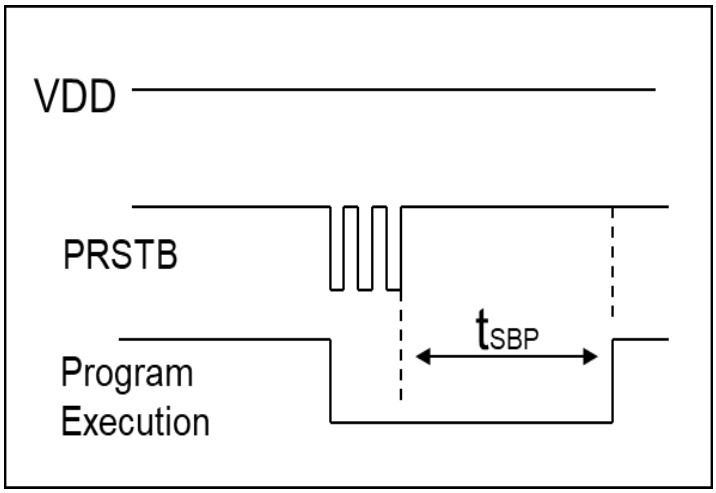
### 5.2.1. Timing charts for reset conditions




Boot up from LVR detection




Boot up from Watch Dog Time Out



## 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. Since the data width is 8-bit, all the 128 bytes data memory of PFS122 can be accessed by indirect access mechanism.

## 5.4. Oscillator and clock

There are three oscillator circuits provided by PFS122: external crystal oscillator (EOSC), internal high RC oscillator (IHRC) and internal low RC oscillator (ILRC), and these three oscillators are enabled or disabled by registers eoscr.7, clkmd.4 and clkmd.2 independently. User can choose one of these three oscillators as system clock source and use ***clkmd*** register to target the desired frequency as system clock to meet different applications.

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

Table 2: Three oscillation circuits

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

After boot-up, the IHRC and ILRC oscillators are enabled. The frequency of IHRC can be calibrated to eliminate process variation by ***ihrcr*** register; normally it is calibrated to 16MHz. Please refer to the measurement chart for IHRC frequency verse $V_{DD}$ and IHRC frequency verse temperature. The frequency of ILRC will vary by process, supply voltage and temperature, please refer to DC specification and do not use for accurate timing application.

### 5.4.2. Chip calibration

The IHRC frequency and bandgap reference voltage may be different chip by chip due to manufacturing variation, PFS122 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**=2.5 ~ 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 starting the system. The program code for IHRC frequency calibration is executed only one time that occurs in writing the codes into MTP 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 PFS122 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, $V_{DD}$=2.5V

After boot up, CLKMD = 0x1C：

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

©Copyright 2020, PADAUK Technology Co. Ltd

**(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 by Boot-up Time
- System CLK = ILRC or IHRC/64 (by Boot-upTime)
- Watchdog timer is enabled, ILRC is enabled, PA5 is in input mode

### 5.4.4. External Crystal Oscillator

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

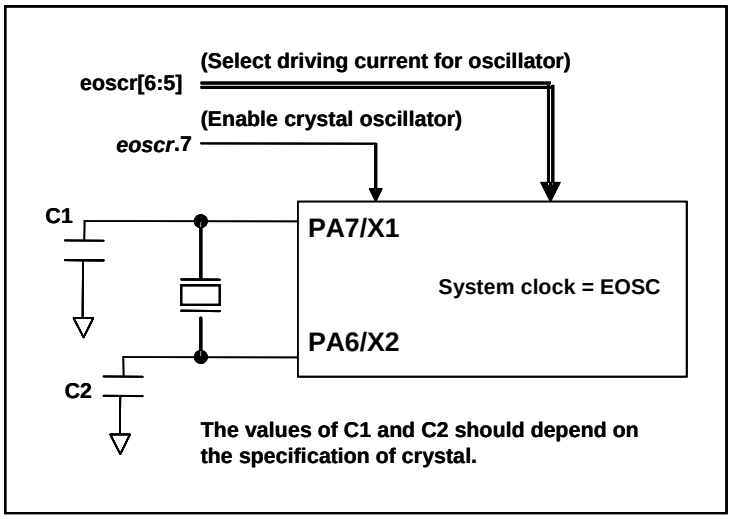



Fig.2: Connection of crystal oscillator

Besides crystal, external capacitor and options of PFS122 should be fine tuned in *eoscr* (0x0a) register to have good sinusoidal waveform. The *eoscr*.7 is used to enable crystal oscillator module, *eoscr*.6 and *eoscr*.5 are used to set the different driving current to meet the requirement of different frequency of crystal oscillator:

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

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

©Copyright 2020, PADAUK Technology Co. Ltd

resonator, please refer to its specification for proper values of C1 and C2.

| Frequency | C1 | C2 | Measured Start-up time | Conditions |
|---|---|---|---|---|
| 4MHz | 4.7pF | 4.7pF | 6ms | (*eoscr[*6:5]=11, *misc*.6=0) |
| 1MHz | 10pF | 10pF | 11ms | (*eoscr[*6:5]=10, *misc*.6=0) |
| 32KHz | 22pF | 22pF | 450ms | (*eoscr*[6:5]=01, *misc*.6=0) |

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

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

```
void    FPPA0 (void)
{
    . ADJUST_IC   SYSCLK=IHRC/16, IHRC=16MHz, VDD=5V
$   EOSCR   Enable, 4MHz;          // EOSCR = 0b110_00000;

$   T16M  EOSC, /1, BIT13;         // T16 receive 2^14=16384 clocks of crystal EOSC,
                                   // Intrq.T16 =>1, crystal EOSC Is stable
    WORD    count =    0;
    stt16  count;
    Intrq.T16   =    0;
    do
    {   nop; }while(!Intrq.T16);   // count from 0x0000 to 0x2000, then set INTRQ.T16
    clkmd=    0xB4;                // switch system clock to EOSC;

    Clkmd.4 = 0;                   // disable IHRC
    ...
```

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

### 5.4.5. System Clock and LVR level

The clock source of system clock comes from EOSC, IHRC and ILRC, the hardware diagram of system clock in the PFS122 is shown as Fig.3.

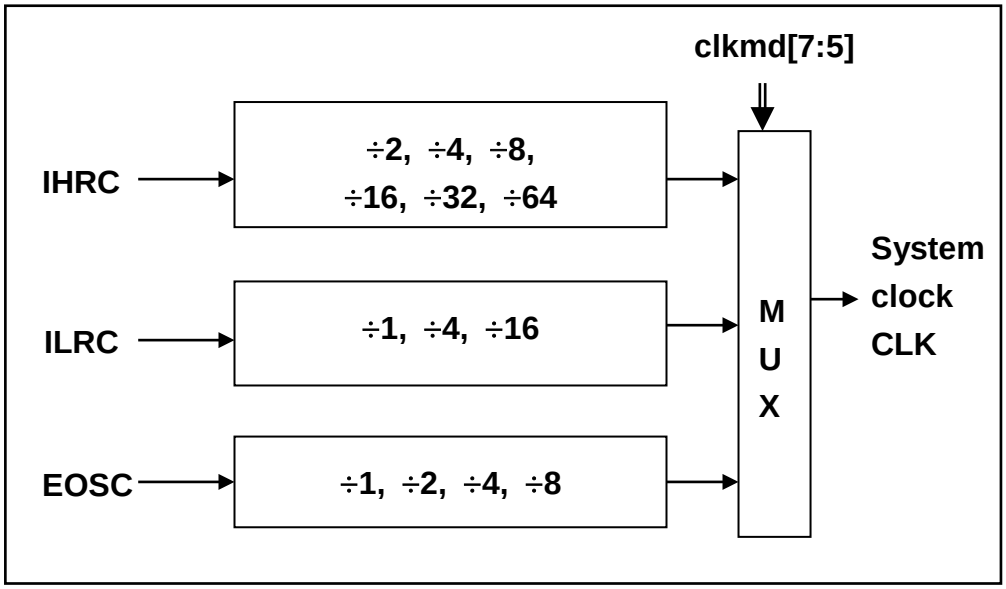



Fig.3: Options of System Clock

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.

©Copyright 2020, PADAUK Technology Co. Ltd

### 5.4.6. System Clock Switching

After IHRC calibration, user may want to switch system clock to a new frequency or may switch system clock at any time to optimize the system performance and power consumption. Basically, the system clock of PFS122 can be switched among IHRC, ILRC and EOSC by setting the ***clkmd*** register at any time; system clock will be the new one after writing to ***clkmd*** register immediately. Please notice that the original clock module can NOT be turned off at the same time as writing command to ***clkmd*** register. The examples are shown as below and more information about clock switching, please refer to the "Application Note" -> "IC Introduction" -> "Register Introduction" -> CLKMD".

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

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

**Case 2:** Switching system clock from ILRC to EOSC

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

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

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

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

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

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

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

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

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

©Copyright 2020, PADAUK Technology Co. Ltd

## 5.5. Comparator

One hardware comparator is built inside the PFS122; 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.

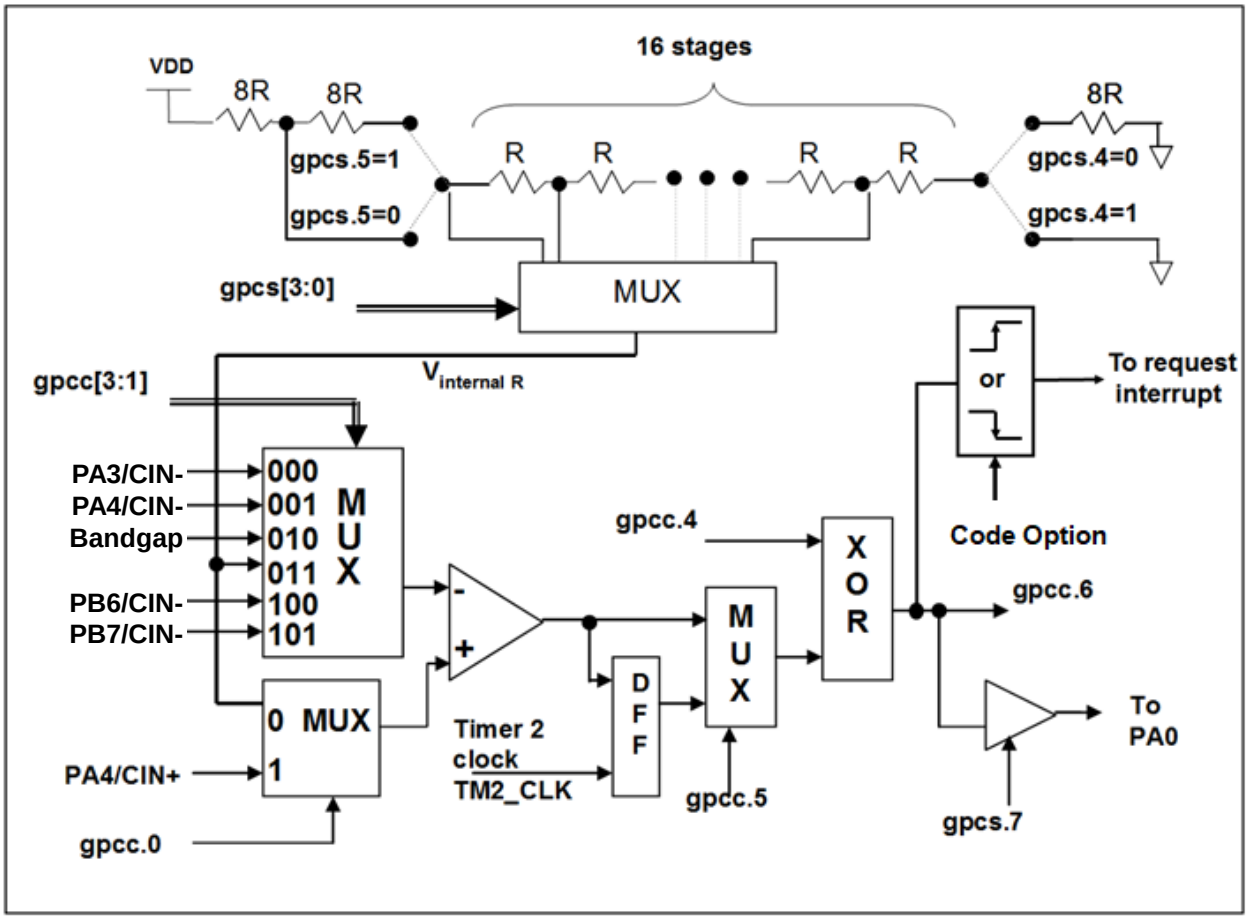



Fig.4: Hardware diagram of comparator

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

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

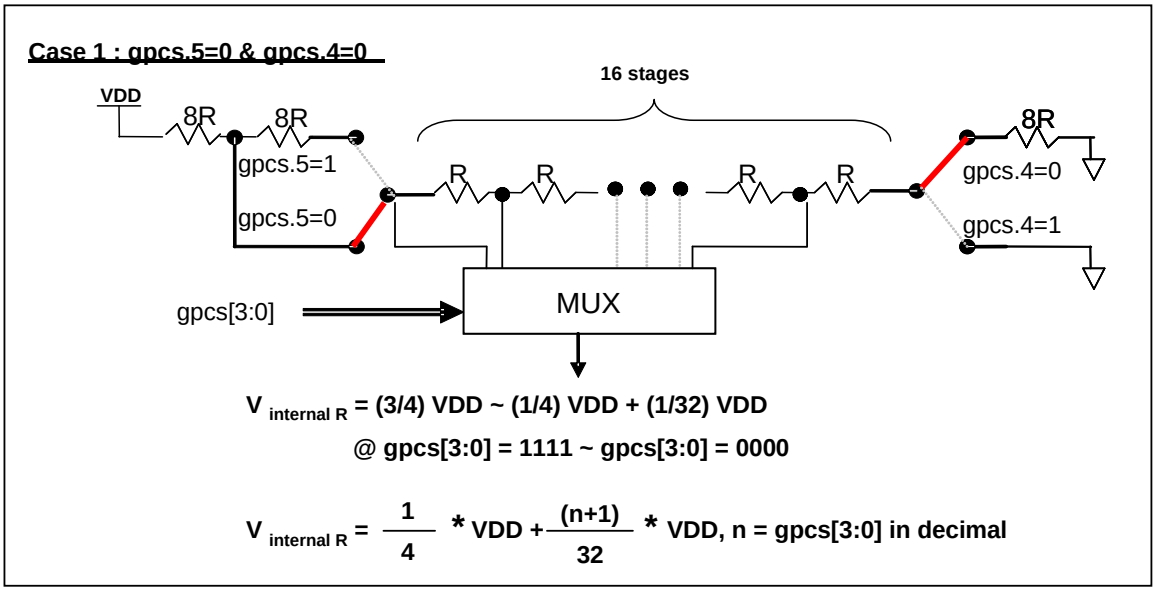



Fig.5: $V_{internal\ R}$ hardware connection if gpcs.5=0 and gpcs.4=0

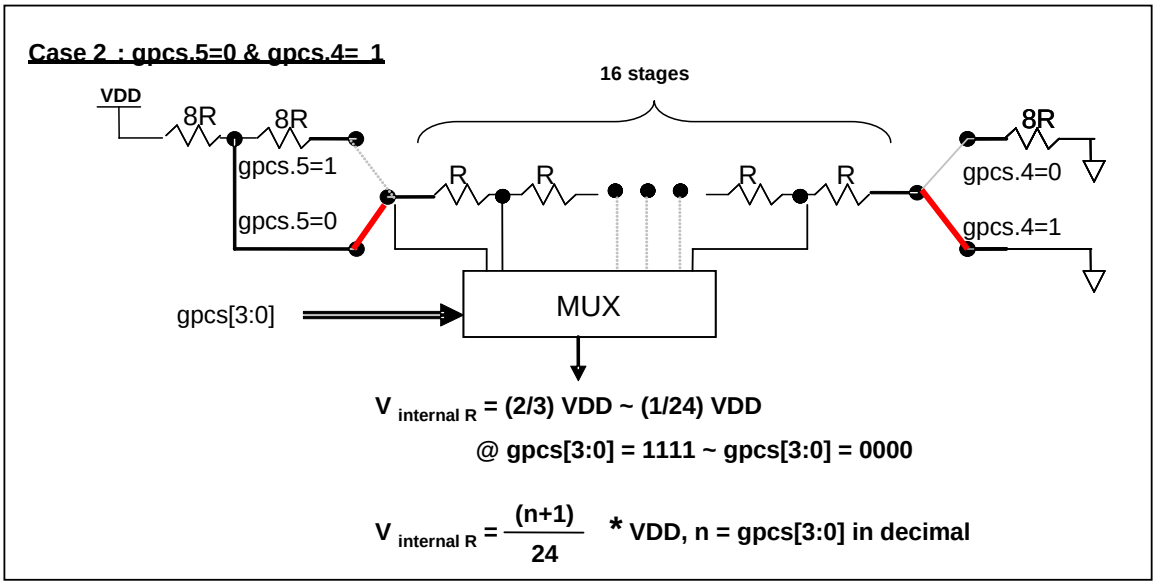



Fig.6: $V_{internal\ R}$ hardware connection if gpcs.5=0 and gpcs.4=1

©Copyright 2020, PADAUK Technology Co. Ltd

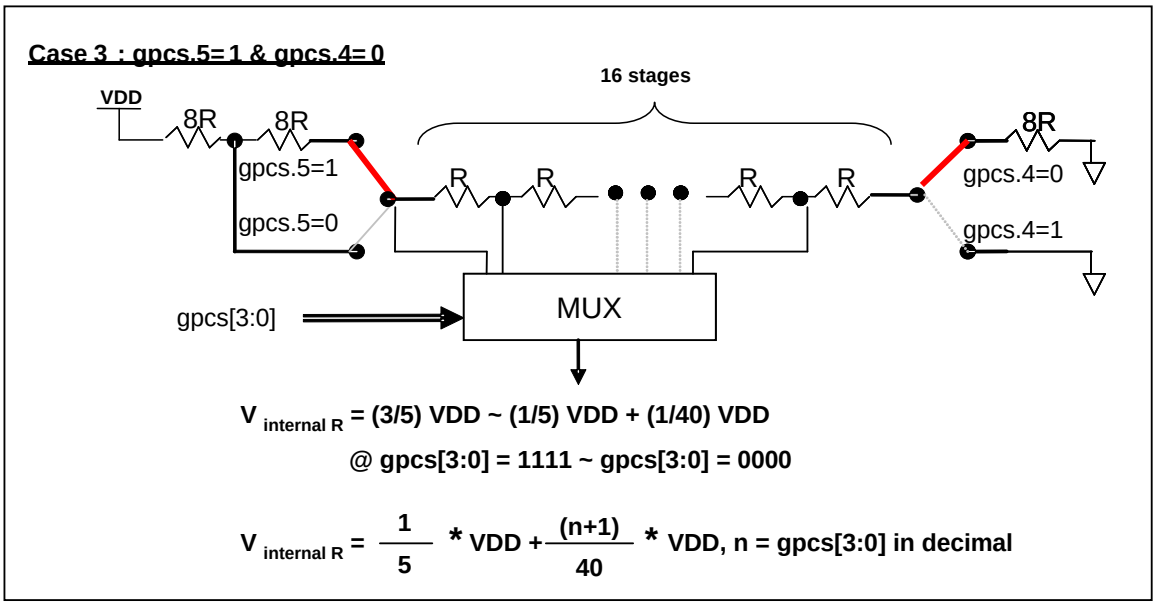



Fig.7: $V_{internal\ R}$ hardware connection if gpcs.5=1 and gpcs.4=0

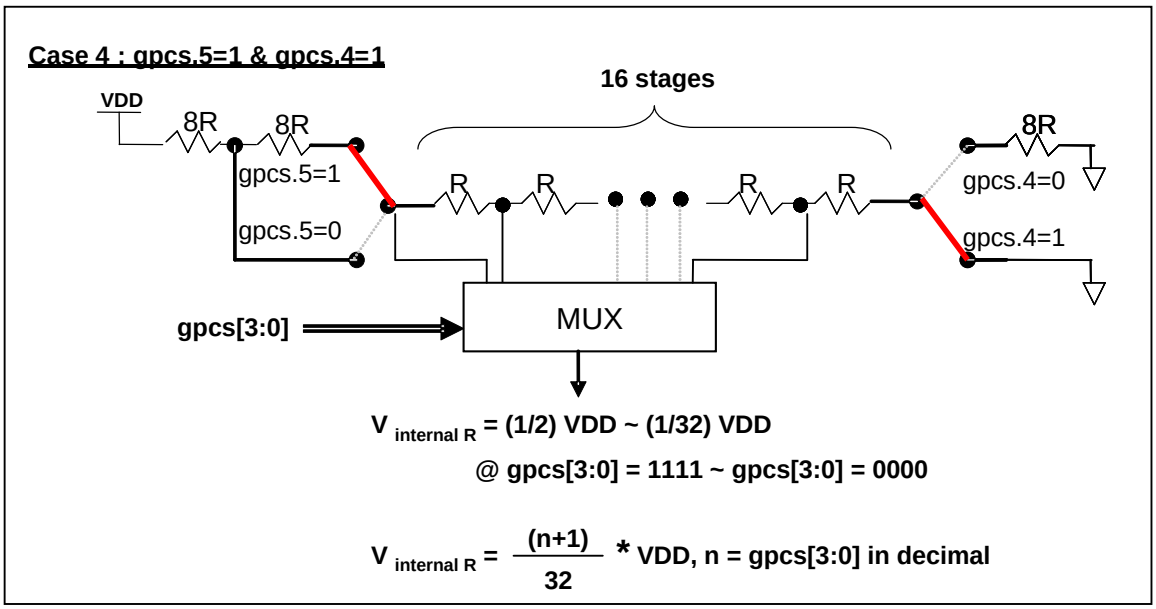



Fig.8: $V_{internal\ R}$ hardware connection if gpcs.5=1 and gpcs.4=1

### 5.5.2 Using the comparator

Case I:

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

```
gpcs    = 0b0_0_00_1001;        // V_internal R = V_DD*(18/32)
gpcc    = 0b1_0_0_0_000_0;      // enable comp, - input: PA3, + input: V_internal R
padier  = 0bxxxx_0_xxx;         // disable PA3 digital input to prevent leakage current
```

*or*

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

Case 2:

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

```
gpcs    = 0b1_0_10_1101;        // output to PA0,V_internal R = V_DD*(22/40)
gpcc    = 0b1_0_0_1_011_1;      // Inverse output, - input: V_internal R, + input: PA4
padier  = 0bxxx_0_xxxx;         // disable PA4 digital input to prevent leakage current
```

*or*

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

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 $V_{DD}$, and 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 VDD can be calculated by using the following equations:

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

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

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

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

*Case 1:*

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

## 5.6 16-bit Timer (Timer16)

A 16-bit hardware timer (Timer16) is implemented in the PFS122, the clock sources of Timer16 may come from system clock (CLK), clock of external crystal oscillator (EOSC), internal high RC oscillator (IHRC), internal low RC oscillator (ILRC), PA4 and PA0. A multiplex is used to select clock output for the clock source. Before sending clock to the counter16, a pre-scaling logic with divided-by-1, 4, 16, and 64 is used for wide range counting.

The 16-bit counter performs up-counting operation only. The counter initial values can be stored from memory by ***stt16*** instruction and the counting values can be loaded to memory by ***ldt16*** instruction. A selector is used to select the interrupt condition of Timer16, whenever overflow occurs, the Timer16 interrupt can be triggered. The hardware diagram of Timer16 is shown as Fig.9. The interrupt source of Timer16 comes from one of bit 8 to 15 of 16-bit counter, and the interrupt type can be rising edge trigger or falling edge trigger which is specified in the bit 4 of ***integs*** register (IO address 0x0C).

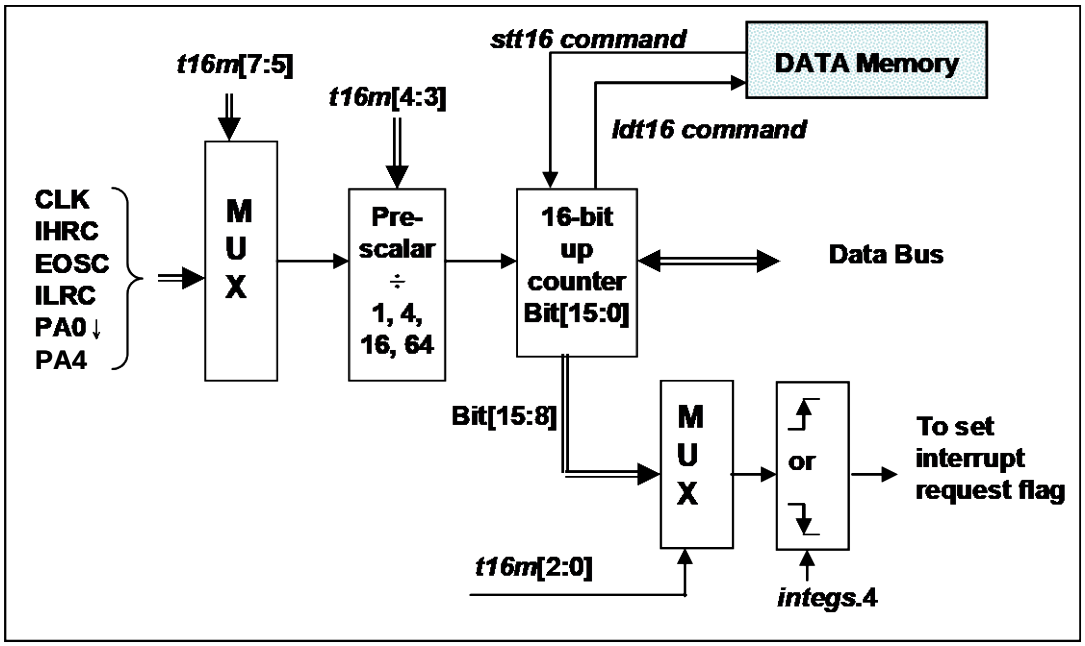



Fig.9: Hardware diagram of Timer16

When using the Timer16, the syntax for Timer16 has been defined in the .INC file. There are three parameters to define the Timer16; 1st parameter is used to define the clock source of Timer16, 2nd parameter is used to define the pre-scalar and the last one is to define the interrupt source. The detail description is shown as below:

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

User can define the parameters of T16M based on system requirement, some examples are shown below and more examples please refer to "Application Note → IC Introduction → Register Introduction → T16M" in IDE utility.

```
$ T16M SYSCLK, /64, BIT15;
    // choose (SYSCLK/64) as clock source, every 2^16 clock to set INTRQ.2=1
    // if using System Clock = IHRC / 2 = 8 MHz
    // SYSCLK/64 = 8 MHz/64 = 125KHz, about every 512 mS to generate INTRQ.2=1

$ T16M EOSC, /1, BIT13;
    // choose (EOSC/1) as clock source, every 2^14 clocks to generate INTRQ.2=1
    // if EOSC=32768 Hz, 32768 Hz/(2^14) = 2Hz, every 0.5S to generate INTRQ.2=1

$ T16M PA0_F, /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
```

If Timer16 is operated at free running, the frequency of interrupt can be described as below:

$$F_{INTRQ_T16M} = F_{clock\ source} \div P \div 2^{n+1}$$

Where, F is the frequency of selected clock source to Timer16;

P is the selection of t16m [4:3]; (1, 4, 16, 64)

N is the $n^{th}$ bit selected to request interrupt service, for example: n=10 if bit 10 is selected.

©Copyright 2020, PADAUK Technology Co. Ltd

## 5.7 8-bit Timer (Timer2/Timer3) with PWM generation

Two 8-bit hardware timers (Timer2 and Timer3) with PWM generation are implemented in the PFS122. The following descriptions thereinafter are for Timer2 only. It is because Timer3 have same structure with Timer2. Please refer to Fig.10 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), external crystal oscillator (EOSC), 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.11 shows the timing diagram of Timer2 for both period mode and PWM mode.

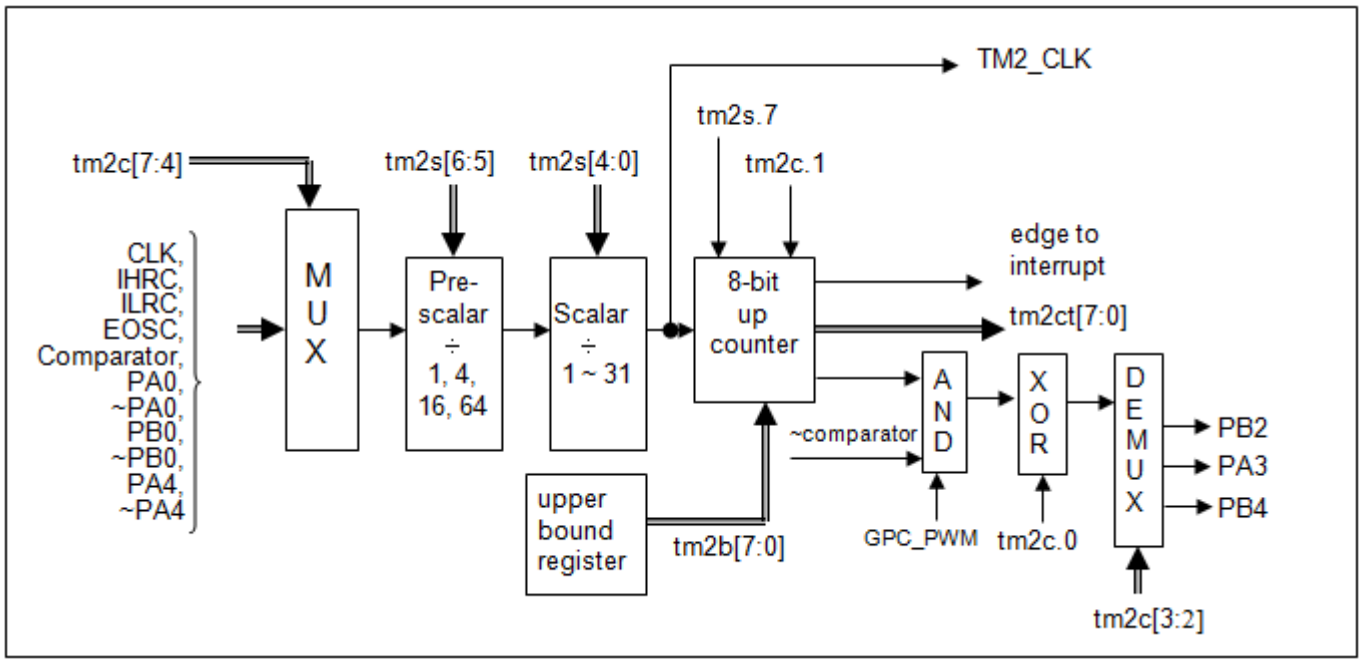
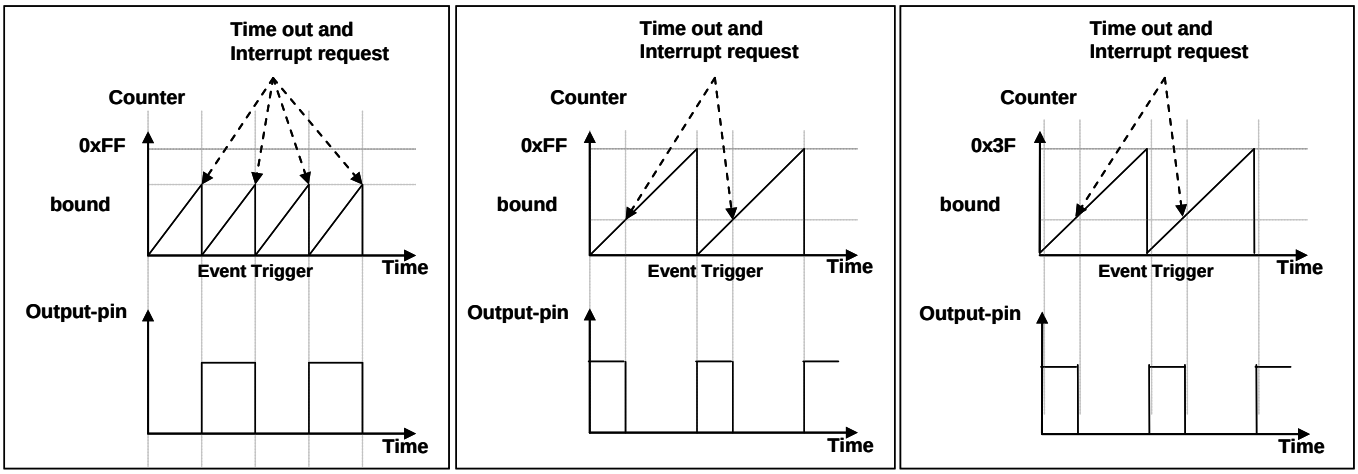



Fig.10: Timer2 hardware diagram




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

Fig.11: 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. 12.

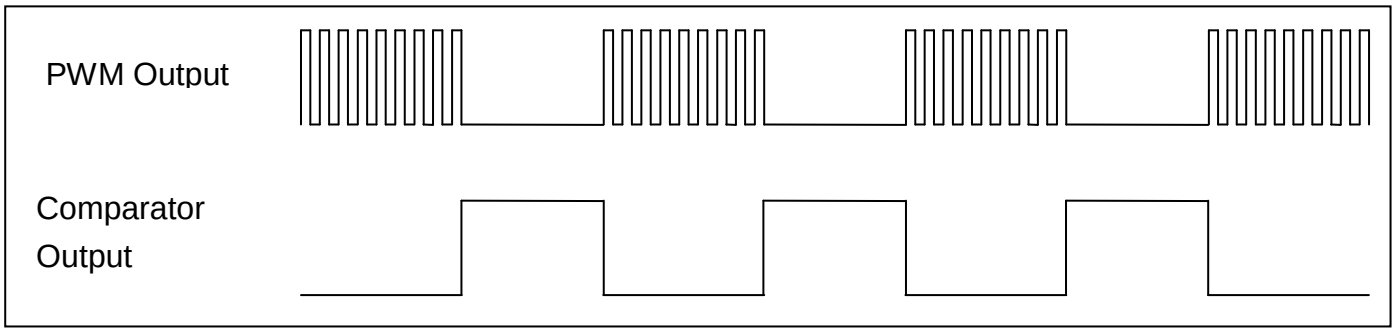



Fig.12: Comparator controls the output of PWM waveform

### 5.7.1 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)

©Copyright 2020, PADAUK Technology Co. Ltd

Example 1:

tm2c = 0b0001_1000, Y=8MHz

tm2b = 0b0111_1111, K=127

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

➔ frequency of output = 8MHz ÷ [ 2 × (127+1) × 1 × (0+1) ] = 31.25KHz

Example 2:

tm2c = 0b0001_1000, Y=8MHz

tm2b = 0b0111_1111, K=127

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

➔ frequency = 8MHz ÷ ( 2 × (127+1) × 64 × (31+1) ) =15.25Hz

Example 3:

tm2c = 0b0001_1000, Y=8MHz

tm2b = 0b0000_1111, K=15

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

➔ frequency = 8MHz ÷ ( 2 × (15+1) × 1 × (0+1) ) = 250KHz

Example 4:

tm2c = 0b0001_1000, Y=8MHz

tm2b = 0b0000_0001, K=1

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

➔ frequency = 8MHz ÷ ( 2 × (1+1) × 1 × (0+1) ) =2MHz

The sample program for using the Timer2 to generate periodical waveform 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;
    }
}
```

### 5.7.2 Using the Timer2 to generate 8-bit PWM waveform

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

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

**Duty of Output = [( K＋1 ) ÷ 256]×100%**

Where, Y = tm2c[7:4] : frequency of selected clock source

K = tm2b[7:0] : bound register in decimal

S1= tm2s[6:5] : pre-scalar (S1= 1, 4, 16, 64)

S2 = tm2s[4:0] : scalar register in decimal (S2= 0 ~ 31)

Example 1:

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

Example 2:

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

Example 3:

tm2c = 0b0001_1010, Y=8MHz
tm2b = 0b1111_1111, K=255
tm2s = 0b0_00_00000, S1=1, S2=0
➔ 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 = 0b0_00_00000, S1=1, S2=0
➔ frequency of output = 8MHz ÷ ( 256 × 1 × (0+1) ) = 31.25KHz
➔ duty of output = [(9+1) ÷ 256] × 100% = 3.9%

©Copyright 2020, PADAUK Technology Co. Ltd

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
    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;
    }
}
```

### 5.7.3 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)

Users can set Timer2 to be 7-bit PWM mode instead of 6-bit mode by using ***TMx_Bit*** code option. At that time, the calculation factors of the above equations become 128 instead of 64.

Example 1:

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

Example 2:

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

©Copyright 2020, PADAUK Technology Co. Ltd

Example 3:

tm2c = 0b0001_1010, Y=8MHz
tm2b = 0b0011_1111, K=63
tm2s = 0b1_00_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 = 0b1_00_00000, S1=1, S2=0
➔ frequency = 8MHz ÷ ( 64 × 1 × (0+1) ) = 125KHz
➔ duty = [(0+1) ÷ 64] × 100% =1.5%

### 5.7.4 Complementary PWM with Dead Zones

User can get complementary PWM with dead zones by employing TM2 and TM3. Here provide an example in which duty cycle and dead time are adjustable.

```
//------- These two parameters need be defined when T(PWM) = 256 us ----------
#define   PWM_pulse          70    //  70 us; Adjust it for a different duty cycle of TM2/TM3.
#define   dead_zone          30    //  30 us; Adjust it for the best dead time.
//--------Parameters for switching duty cycle ----------------------------------
#define   PWM_Pulse_a        100   //  100 us; Adjust it for a different duty cycle of TM2/TM3
#define   PWM_Pulse_b        160   //  160 us; Adjust it for a different duty cycle of TM2/TM3
#define   t_delay            500   //   500 us; Switching time of duty cycle

void    FPPA0 (void)
{
  // SYSCLK must quicker than Timer2's clock. Here set SYSCLK=2MHz to capture Tm2ct = 0.
  .ADJUST_IC SYSCLK=IHRC/8, IHRC=16MHz, VDD=3.3V, Init_ram;

  //******Generate complementary PWM with dead zones in a fixed-duty cycle***************
  //------Set the counter upper bound, duty cycle and TMXCT ----------
  $ TM2S 8BIT,/4,/4                    //  16MHz /4 /4 /256   = 1MHz / 256 =    256 us
  TM2B      =      PWM_pulse - 1;

  $ TM3S 8BIT,/4,/4                    //  16MHz /4 /4 /256
  TM3B    =    PWM_pulse + 2 * dead_zone - 1;
  TM2CT   =    0;
```

```
TM3CT   =    0;

//------Timer PWM output control --------------------
$ TM3C    IHRC, PB5, PWM, Inverse;        //  Inverse output
.delay      dead_zone*2 - 2;              //  "*2": SYSCLK = 2MHz
                                          //  "-2": executing "$ TM3C XXXX" needs two
                                          //   instructions
$ TM2C    IHRC, PB4, PWM;
//***Note: Do not change the sequence of the control part's program*****
 //-------Following codes can be for reference when user needs switch duty cycle ----------
 //------ Switching PWM_pulse --------------------
 While (1)
 {
      While(tm2ct!=0)    {}                //  Wait till tm2ct=0 to avoid noise
      TM2B      =        PWM_Pulse_a - 1;
      TM3B      =        PWM_Pulse_a + 2 * dead_zone - 1;
      .delay       t_delay*2;

       While(tm2ct!=0)   {}
       TM2B      =        PWM_Pulse_b - 1;
       TM3B      =        PWM_Pulse_b + 2 * dead_zone - 1;
       .delay       t_delay*2;
  }
}
```

The following figures show the waveforms at different condition.

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

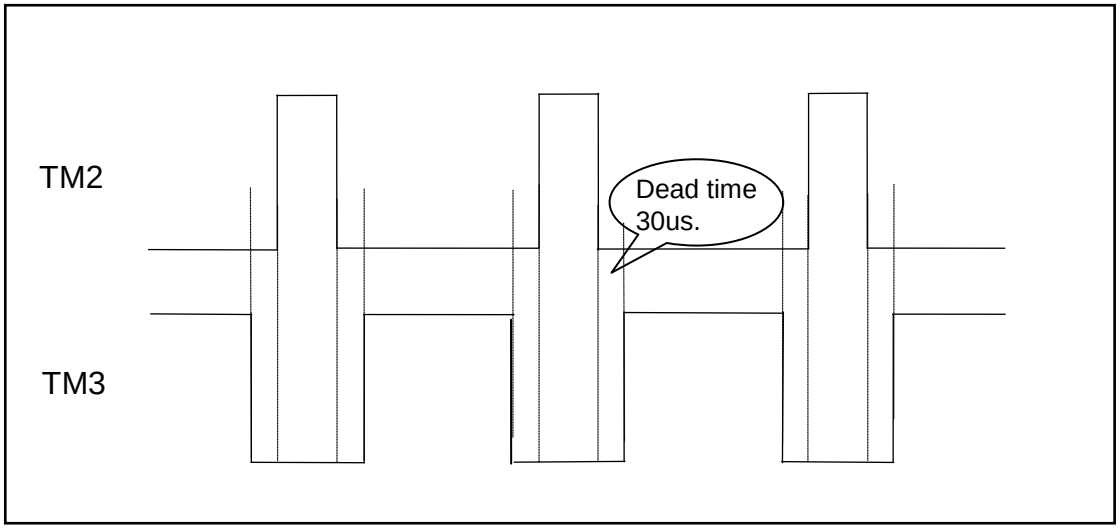



Fig. 13: Two complementary PWM waveforms with dead zones

2. PWM waveforms when switching two duty cycles:

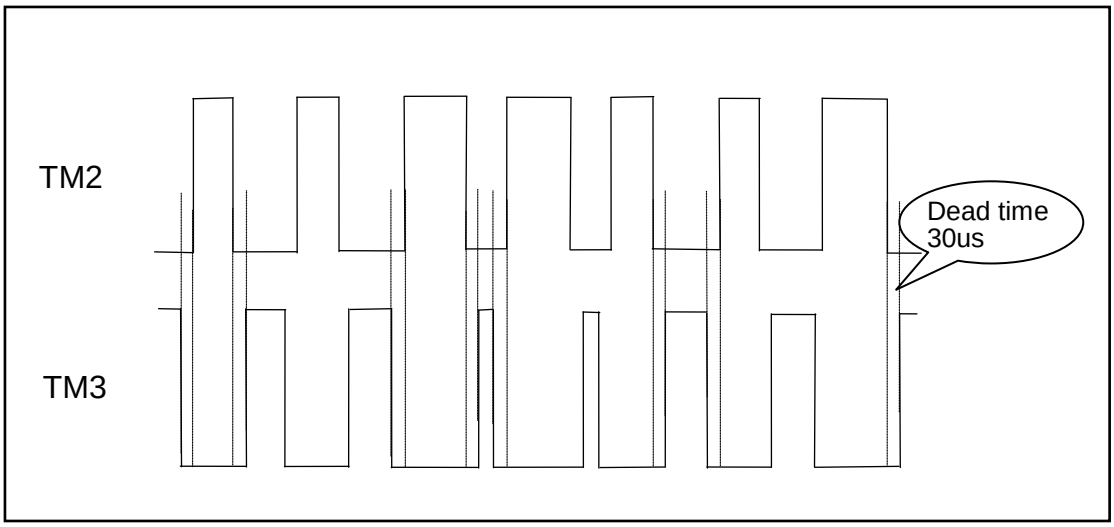



Fig. 14: Two complementary PWM waveforms with dead zones

**Note:** This example just illustrate a method for generating complementary PWM with dead zones and switching duty cycle. If users try to switch duty cycle by adjusting PWM_pulse: such as when the present PWM_pulse = 70, directly let PWM_pulse_a = 100 and PWM_pulse_b = 160. Then the new value must not be re-assigned to tm2b register until tm2ct is 0.

This method can effectively deal with the problems such as first duty cycle inaccuracy and possible dead zone time reduction or dead zone disappear caused by assigning new value to tm2b when tm2ct is not 0. Please handle it carefully and consult FAE when necessary according to the practical application specifications.

©Copyright 2020, PADAUK Technology Co. Ltd

## 5.8 WatchDog Timer

The watchdog timer (WDT) is a counter with clock coming from ILRC. WDT can be cleared by power-on-reset or by command ***wdreset*** at any time. There are four different timeout periods of watchdog timer to 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 save 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, PFS122 will be reset to restart the program execution. The relative timing diagram of watchdog timer is shown as Fig.15.

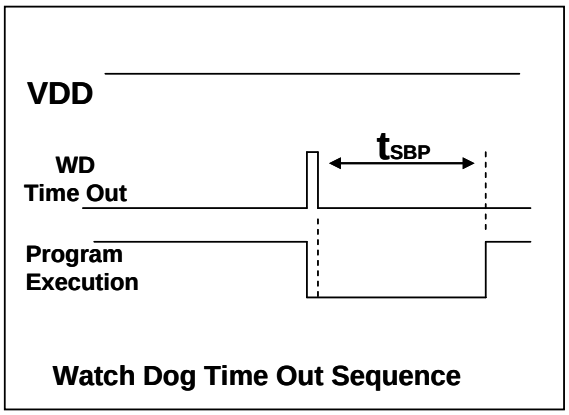



Fig.15: Sequence of Watch Dog Time Out

## 5.9 Interrupt

There are seven interrupt lines for PFS122:

- External interrupt PA0/PB5
- External interrupt PB0/PA4
- ADC interrupt
- Timer16 interrupt
- GPC interrupt
- Timer2 interrupt
- Timer3 interrupt

Every interrupt request line has its own corresponding interrupt control bit to enable or disable it; the hardware diagram of interrupt function is shown as Fig.16. 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***

©Copyright 2020, PADAUK Technology Co. Ltd

instruction (enable global interrupt) to enable interrupt operation and ***disgint*** instruction (disable global interrupt) to disable it.

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

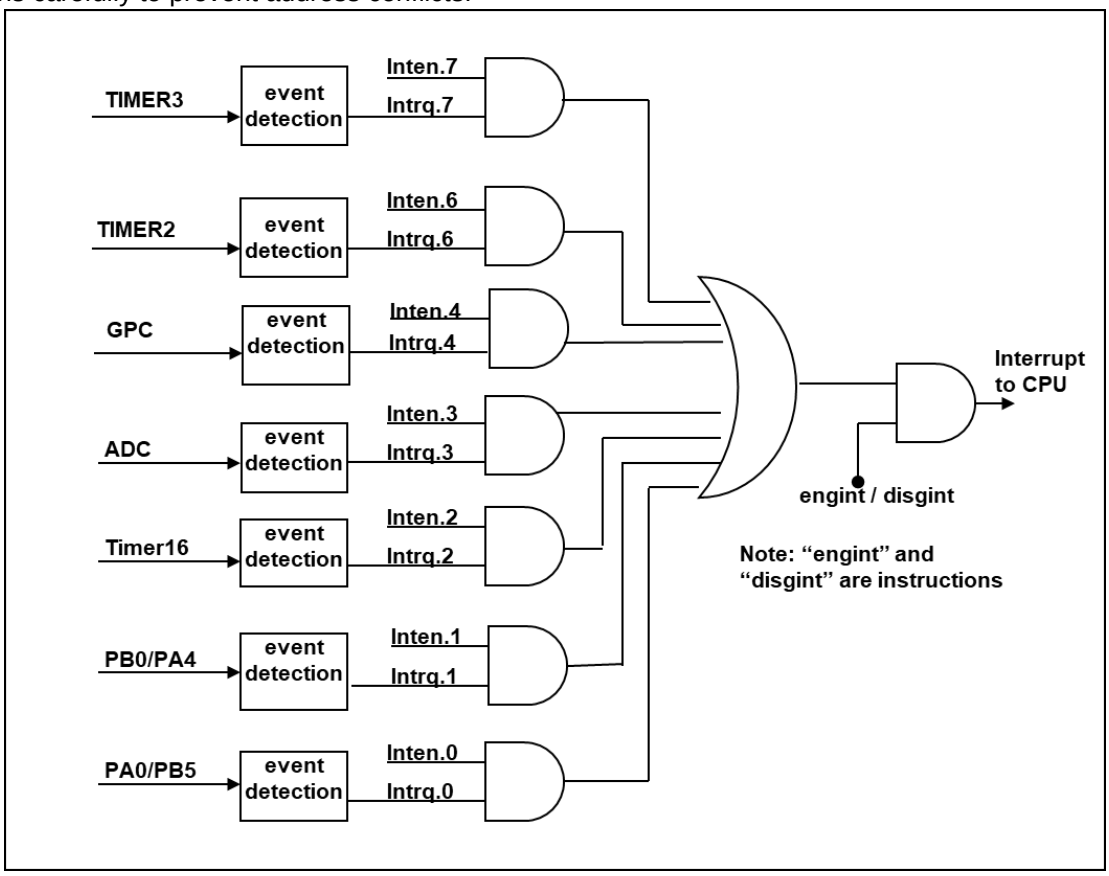



Fig.16: Hardware diagram of interrupt controller

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.

©Copyright 2020, PADAUK Technology Co. Ltd

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

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

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

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

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

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

## 5.10 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 periodically and Power-Down mode is used in the system which needs power down deeply with seldom wake-up.

### 5.10.1 Power-Save mode ("*stopexe*")

Using "***stopexe***" instruction to enter the Power-Save mode, only system clock is disabled, remaining all the oscillator modules active. For CPU, it stops executing; however, for Timer16, counter keep counting if its clock source is not the system clock. The wake-up sources for "stopexe" can be IO-toggle or Timer16 counts to set values when the clock source of Timer16 is IHRC or ILRC modules, or 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 and EOSC oscillator modules: No change, keep active if it was enabled.
- ILRC oscillator modules: must remain enabled, need to start with ILRC when be wakening up.
- System clock: Disable, therefore, CPU stops execution.
- MTP memory is turned off.
- Timer counter: Stop counting if system clock is selected 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. Comparator wake-up: It need setting *GPCC*.7=1 and *GPCS*.6=1 to enable the comparator wake-up function at the same time.

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

```
$ T16M      ILRC, /1, BIT8          // Timer16 setting
$ INTEGS    BIT_R, xxx;             // BITx 0 to 1 will trigger (default)

...
WORD        count =     0;
STT16       count;
stopexe;
...
```

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

### 5.10.2 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 PFS122 detail when "***stopsys***" command is issued:

- All the oscillator modules are turned off.
- MTP memory is turned off.
- The contents of SRAM and registers remain unchanged.
- Wake-up sources: IO toggle in digital mode. (PxDIER bit is 1)

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

```
CLKMD     =     0xF4;      //    Change clock from IHRC to ILRC
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
```

### 5.10.3 Wake-up

After entering the Power-Down or Power-Save modes, the PFS122 can be resumed to normal operation by toggling IO pins, Wake-up from timer are 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 | Timer wake-up |
| STOPSYS | Yes | No |
| STOPEXE | 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 PFS122, registers ***pxdier*** 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*** register, and the time for fast wake-up is about 45 ILRC clocks from IO toggling.

| Suspend mode | Wake-up mode | Wake-up time ($t_{WUP}$) from IO toggle |
|---|---|---|
| STOPEXE suspend<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 |

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

©Copyright 2020, PADAUK Technology Co. Ltd

## 5.11 IO Pins

All the pins can be independently set into two states output or input by configuring the data registers (***pa, pb***), control registers (***pac, pbc***) and pull-high registers (***paph, pbph)*** or pull-low registers (***papl, pbpl***). All these pins have Schmitt-trigger input buffer and output driver with CMOS level. When it is set to output low, the pull-high / pull-low 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 6 shows the configuration table of bit 0 of port A. The hardware diagram of IO buffer is also shown as Fig.17.

| *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-low / pull-high rseistor |
| 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 6: PA0 Configuration Table

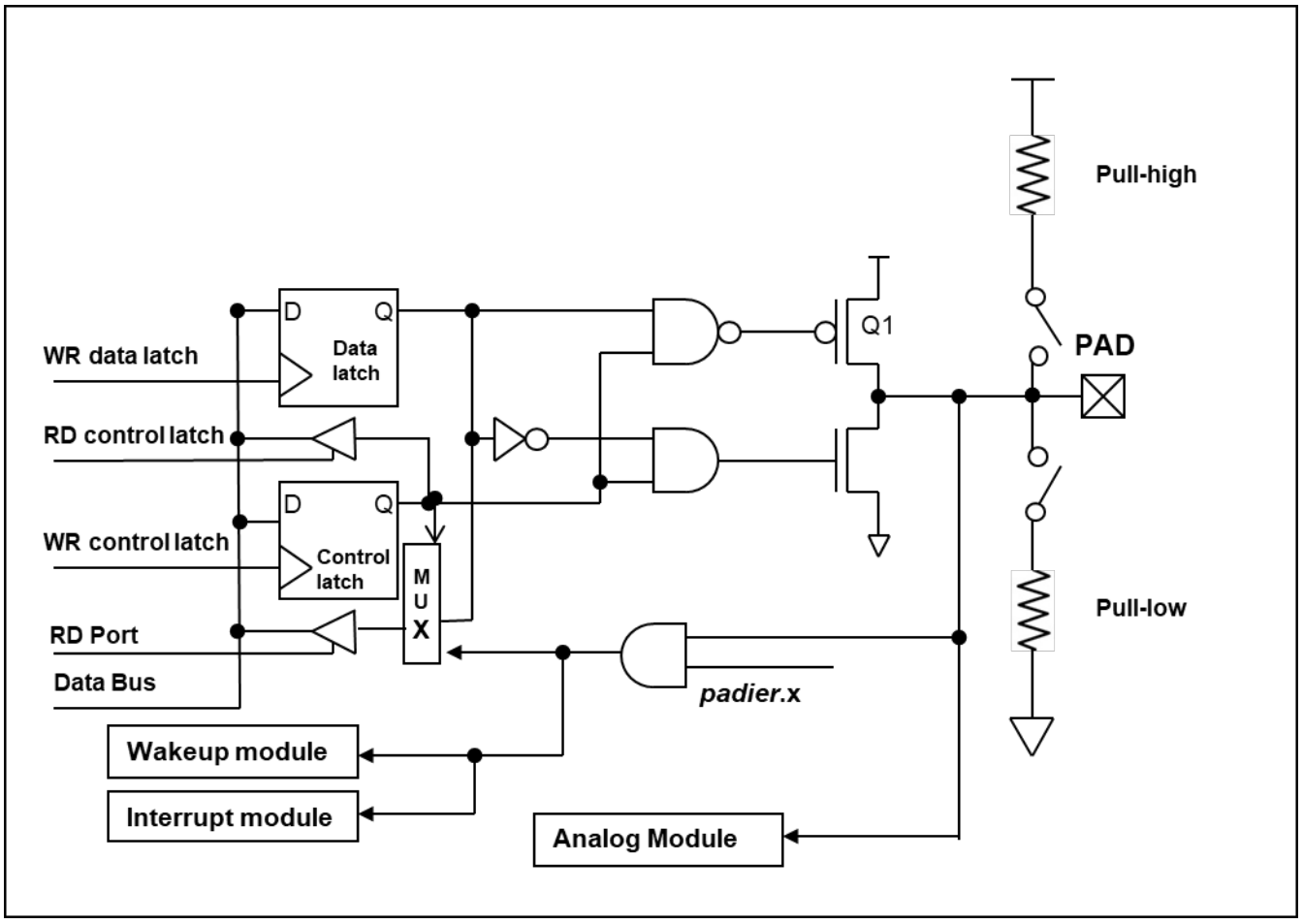



Fig. 17: Hardware diagram of IO buffer

PB4 and PB7 can adjust their drive and sink current by code option ***PB4_PB7_Drive.***

All the IO pins have the same structure. The corresponding bits in registers ***padier*** / ***pbdier*** should be set to low to prevent leakage current for those pins are selected to be analog function. When PFS122 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 ***pxdier*** to high***.*** The same reason, ***padier***.0 should be set high when PA0 is used as external interrupt pin, and so for other external interrupt pins: PB0, PA4 and PB5.

## 5.12 Reset and LVR

### 5.12.1 Reset

There are many causes to reset the PFS122, once reset is asserted, most of all the registers in PFS122 will be set to default values, system should be restarted once abnormal cases happen, or by jumping program counter to address 0x0. The data memory is in uncertain state when reset comes from power-up and LVR; however, the content will be kept when reset comes from PRSTB pin or WDT timeout.

### 5.12.2 LVR reset

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

## 5.13 Analog-to-Digital Conversion (ADC) module

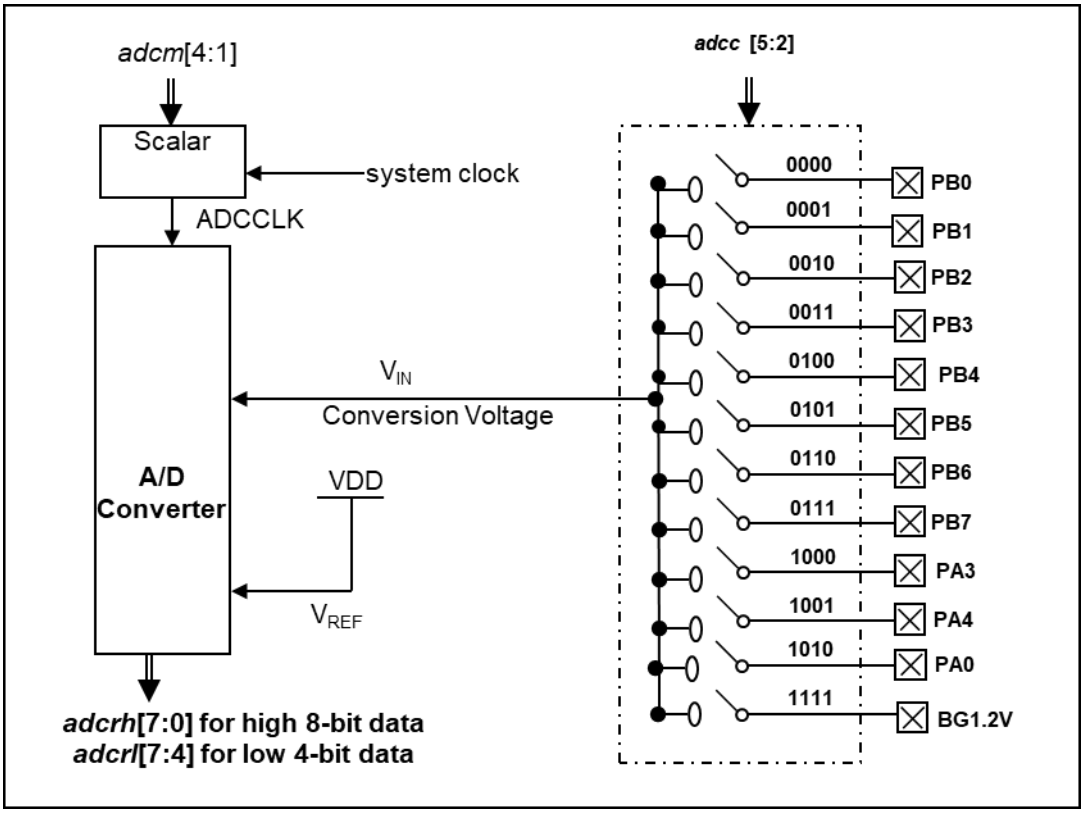



Fig. 18: ADC Block Diagram

There are 6 registers when using the ADC module, which are:

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

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

(1) Configure the AD conversion clock by ***adcm*** register
(2) Configure the pin as analog input by ***padier***, ***pbdier*** register
(3) Select the ADC input channel by ***adcc*** register
(4) Enable the ADC module by ***adcc*** register
(5) 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'
(6) Read the ADC result registers

### 5.13.1 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.19, 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.

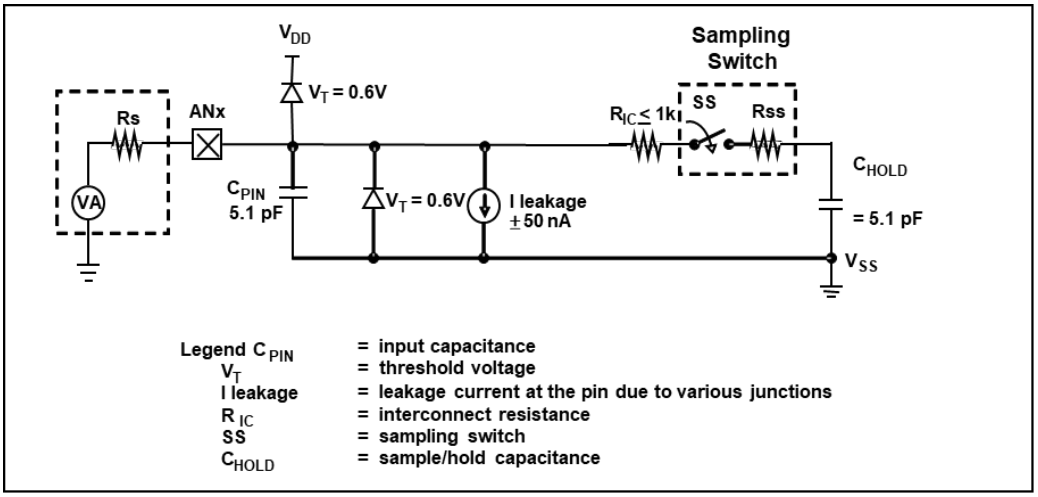



Fig.19: Analog Input Model

### 5.13.2 ADC clock selection

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.

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.

### 5.13.3 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 1.2V. For those external pins defined as analog input, to avoid leakage current from the digital circuit of the shared IO ports, please always remember to disable the digital input function (set the corresponding bit of ***padier or pbdier*** register to be 0).

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

### 5.13.4 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 resistor
PBPL    =   0B_XXXX_0000;   //  PB0 ~ PB3 without pull-low resistor
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
```

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

```
$ ADCM  /16;                //  recommend /16 @System Clock=8MHz
$ ADCM  /8;                 //  recommend /8 @System Clock=4MHz
```

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;
```

### 5.13.5 How to calculate ADC input voltage $V_{IN}$

For PFS122, only VDD but not 1.2V bandgap voltage can be selected as the $V_{REF}$ of the ADC. When VDD is not regulated, users have to use the reading of 1.2V bandgap voltage to deduce the input voltage ($V_{IN}$) by the ratio of the readings. The principle is as below:

$V_{BG} / V_{DD} = N_{BG} / 4096$ ....(1)

$V_{IN} / V_{DD} = N_{IN} / 4096$ ....(2)

Where $V_{IN}$ is the analog input voltage

$V_{BG}$ is the 1.2V bandgap voltage

$N_{IN}$ is the corresponding ADC reading of $V_{IN}$

$N_{BG}$ is the corresponding ADC reading of $V_{BG}$

$V_{DD}$ is the $V_{DD}$ at the measuring instant

4096 is the full swing reading when $V_{IN}=V_{DD}$ (12bit: $2^{12} = 4096$)

(2)/(1) we get

$V_{IN}/V_{BG} = N_{IN}/N_{BG}$

And so

$V_{IN} = N_{IN} / N_{BG} * V_{BG}$

It means users can firstly get the readings for $V_{IN}$ and $V_{BG}$ respectively in a very short period that VDD remains unchanged. And then use multiplication and division program module or use look-up table method to finally get the $V_{IN}$ voltage.

If necessary, please contact FAE for demo code reference.

©Copyright 2020, PADAUK Technology Co. Ltd

# 6. IO Registers

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

| Bit | Reset | R/W | Description |
|---|---|---|---|
| 7 - 4 | - | - | Reserved. Please do not use. |
| 3 | 0 | R/W | OV (Overflow Flag). This bit is set to be 1 whenever the sign operation is overflow. |
| 2 | 0 | R/W | AC (Auxiliary Carry Flag). 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 | 0 | R/W | C (Carry Flag). 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 | 0 | R/W | Z (Zero Flag). 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. |

## 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="4">7 - 5</td><td rowspan="4">111</td><td rowspan="4">R/W</td><td colspan="2">System clock (CLK) 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>010: reserved<br>011: EOSC÷4<br>100: EOSC÷2<br>101: EOSC<br>110: ILRC÷4<br>111: ILRC (default)</td><td>000: IHRC÷16<br>001: IHRC÷8<br>010: ILRC÷16 (ICE does NOT Support.)<br>011: IHRC÷32<br>100: IHRC÷64<br>101: EOSC÷8<br>11x: reserved.</td></tr>
<tr><td>4</td><td>1</td><td>R/W</td><td colspan="2">Internal High RC 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">Internal Low RC 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>

©Copyright 2020, PADAUK Technology Co. Ltd

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

| Bit | Reset | R/W | Description |
|---|---|---|---|
| 7 | 0 | R/W | Enable interrupt from Timer3. 0 / 1: disable / enable |
| 6 | 0 | R/W | Enable interrupt from Timer2. 0 / 1: disable / enable |
| 5 | 0 | - | Reserved |
| 4 | 0 | R/W | Enable interrupt from comparator. 0 / 1: disable / enable |
| 3 | 0 | R/W | Enable interrupt from ADC. 0 / 1: disable / enable |
| 2 | 0 | R/W | Enable interrupt from Timer16 overflow. 0 / 1: disable / enable |
| 1 | 0 | R/W | Enable interrupt from PB0/PA4. 0 / 1: disable / enable |
| 0 | 0 | R/W | Enable interrupt from PA0/PB5. 0 / 1: disable / enable |

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

| Bit | Reset | R/W | Description |
|---|---|---|---|
| 7 | - | R/W | Interrupt Request from Timer3, this bit is set by hardware and cleared by software.<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 | - | - | Reserved |
| 4 | - | R/W | Interrupt Request from comparator, 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.<br>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 |

©Copyright 2020, PADAUK Technology Co. Ltd

### 6.6. Timer16 mode Register (*t16m*), IO address = 0x06

| Bit | Reset | R/W | Description |
|---|---|---|---|
| 7 - 5 | 000 | R/W | Timer16 Clock source selection.<br>000: disable<br>001: CLK (system clock)<br>010: reserved<br>011: PA4 falling edge (from external pin)<br>100: IHRC<br>101: EOSC<br>110: ILRC<br>111: PA0 falling edge (from external pin) |
| 4 - 3 | 00 | R/W | Timer16 clock pre-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 the selected bit status 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 |

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

| Bit | Reset | R/W | Description |
|---|---|---|---|
| 7 | 0 | WO | Enable external crystal oscillator. 0 / 1 : Disable / Enable |
| 6 - 5 | 00 | WO | External crystal oscillator selection.<br>00 : reserved<br>01 : Low driving capability, for lower frequency, ex: 32KHz crystal oscillator<br>10 : Middle driving capability, for middle frequency, ex: 1MHz crystal oscillator<br>11 : High driving capability, for higher frequency, ex: 4MHz crystal oscillator |
| 4 - 1 | - | - | Reserved. Please keep 0 for future compatibility. |
| 0 | 0 | WO | Power-down the Bandgap and LVR hardware modules. 0 / 1: normal / power-down. |

©Copyright 2020, PADAUK Technology Co. Ltd

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

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

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

| Bit | Reset | R/W | Description |
|---|---|---|---|
| 7 | 1 | WO | Enable PA7 digital input and wake-up event. 1 / 0 : enable / disable<br>This bit should be set to low to prevent leakage current when external crystal oscillator is used. If this bit is set to low, PA7 can NOT be used to wake-up the system. |
| 6 | 1 | WO | Enable PA6 digital input and wake-up event. 1 / 0 : enable / disable<br>This bit should be set to low to prevent leakage current when external crystal oscillator is used. If this bit is set to low, PA6 can NOT be used to wake-up the system. |
| 5 | 1 | WO | Enable PA5 digital input and wake-up event. 1 / 0 : enable / disable<br>This bit can be set to low to disable PA5 digital input and wake-up function. |
| 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 PA0 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. |
| 0 | 1 | WO | Enable PA0 digital input, wake-up event and interrupt request.<br>1 / 0 : enable / disable<br>This bit can be set to low to prevent leakage current when PA0 is assigned as AD input, and to disable wake-up from PA0 toggling and interrupt request from this pin. |

©Copyright 2020, PADAUK Technology Co. Ltd

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

| Bit | Reset | R/W | Description |
|---|---|---|---|
| 7 - 6 | 11 | WO | Enable PB7~PB6 digital input and wake-up event.<br>1 / 0 : enable / disable<br>These bits can be set to low to prevent leakage current when PB7~PB6 are assigned as AD inputs. When disable is selected, the wake-up function and interrupt requests from these pins are also disabled. |
| 5 | 1 | WO | Enable PB5 digital input, wake-up event and interrupt request.<br>1 / 0 : enable / disable<br>This bit can be set to low to prevent leakage current when PB5 is assigned as AD input, and to disable wake-up from PB5 toggling and interrupt request from this pin. |
| 4 - 1 | 1111 | WO | Enable PB47~PB1 digital input and wake-up event.<br>1 / 0 : enable / disable<br>These bits can be set to low to prevent leakage current when PB4~PB1 are assigned as AD inputs. When disable is selected, the wake-up function and interrupt requests from these pins are also disabled. |
| 0 | 1 | WO | Enable PB0 digital input, wake-up event and interrupt request.<br>1 / 0 : enable / disable<br>This bit can be set to low to prevent leakage current when PB0 is assigned as AD input, and to disable wake-up from PB0 toggling and interrupt request from this pin. |

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

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

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

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

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

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

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

| Bit | Reset | R/W | Description |
|---|---|---|---|
| 7 - 0 | 0x00 | R/W | PA pull-low register. 0 / 1: Disable / Enable. |

### 6.15. Port B Data Register (*pb*), IO address = 0x15

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

©Copyright 2020, PADAUK Technology Co. Ltd

### 6.16. Port B Control Register (*pbc*), IO address = 0x16

| 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 |

### 6.17. Port B Pull-High Register (*pbph*), IO address = 0x17

| 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 |

### 6.18. Port B Pull-Low Register (*pbpl*), IO address = 0x18

| Bit | Reset | R/W | Description |
|---|---|---|---|
| 7 -0 | 0x00 | R/W | PB pull-low register. 0 / 1: Disable / Enable. |

### 6.19. ADC Control Register (*adcc*), IO address = 0x20

| 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>Write “1” to start conversion<br>Read “1” to indicate the ADC is ready or end of conversion. |
| 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<br>Others: reserved |
| 0 - 1 | - | - | Reserved. (keep 0 for future compatibility) |

©Copyright 2020, PADAUK Technology Co. Ltd

### 6.20. ADC Mode Register (*adcm*), IO address = 0x21

| Bit | Reset | R/W | Description |
|---|---|---|---|
| 7 - 4 | - | - | Reserved. (keep 0 for future compatibility) |
| 3 - 1 | 000 | R/W | 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. |

### 6.21. ADC Result High Register (*adcrh*), IO address = 0x22

| 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. |

### 6.22. ADC Result Low Register (*adcrl*), IO address = 0x23

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

### 6.23. MISC Register (*misc*), IO address = 0x26

| 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. The wake-up time is 3000 ILRC clocks (Not for fast boot-up)<br>1: Fast wake up. The wake-up time is 45 ILRC clocks. |
| 4 | - | - | Reserved. |
| 3 | - | - | Reserved. |
| 2 | 0 | WO | Disable LVR function. 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 |

©Copyright 2020, PADAUK Technology Co. Ltd

### 6.24. Comparator Control Register (*gpcc*), IO address = 0x2b

| 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<br>011 : $V_{internal\ R}$<br>100 : PB6 (not for ICE)<br>101 : PB7 (not for ICE)<br>11X: reserved |
| 0 | 0 | R/W | Selection the plus input (+) of comparator.<br>0 : $V_{internal\ R}$<br>1 : PA4 |

### 6.25. Comparator Selection Register (*gpcs*), IO address = 0x2c

| 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)Reserved.<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) |

©Copyright 2020, PADAUK Technology Co. Ltd

### 6.26. Timer2 Control Register (*tm2c*), IO address = 0x30

| Bit | Reset | R/W | Description |
|---|---|---|---|
| 7 - 4 | 0000 | R/W | Timer2 clock selection.<br>0000 : disable<br>0001 : CLK<br>0010 : IHRC or IHRC *2 (by code option TMx_source)<br>0011 : EOSC<br>0100 : ILRC<br>0101 : comparator output (ICE does NOT support.)<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)<br>Others: reserved<br>**Notice:** In ICE mode and IHRC is selected for Timer2 clock, the clock sent to Timer2 does NOT be stopped, Timer2 will keep counting when ICE is in halt state. |
| 3 - 2 | 00 | R/W | Timer2 output selection.<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 |

### 6.27. Timer2 Counter Register (*tm2ct*), IO address = 0x31

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

### 6.28. Timer2 Scalar Register (*tm2s*), IO address = 0x32

| Bit | Reset | R/W | Description |
|---|---|---|---|
| 7 | 0 | WO | PWM resolution selection.<br>0 : 8-bit<br>1 : 6-bit or 7-bit (by code option TMx_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. |

©Copyright 2020, PADAUK Technology Co. Ltd

### 6.29. Timer2 Bound Register (*tm2b*), IO address = 0x33

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

### 6.30. Timer3 Control Register (*tm3c*), IO address = 0x34

| Bit | Reset | R/W | Description |
|---|---|---|---|
| 7 - 4 | 0000 | R/W | Timer3 clock selection.<br>0000 : disable<br>0001 : CLK<br>0010 : IHRC or IHRC *2 (by code option TMx_source)<br>0011 : EOSC<br>0100 : ILRC<br>0101 : comparator output (ICE does NOT support.)<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)<br>Others: reserved<br>**Notice:** In ICE mode and IHRC is selected for Timer3 clock, the clock sent to Timer3 does NOT be stopped, Timer3 will keep counting when ICE is in halt state. |
| 3 - 2 | 00 | R/W | Timer3 output selection.<br>00 : disable<br>01 : PB5<br>10 : PB6<br>11 : PB7 |
| 1 | 0 | R/W | Timer3 mode selection. 0 / 1 : period mode / PWM mode |
| 0 | 0 | R/W | Enable to inverse the polarity of Timer3 output.<br>0 / 1: disable / enable |

### 6.31. Timer3 Counter Register (*tm3ct*), IO address = 0x35

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

©Copyright 2020, PADAUK Technology Co. Ltd

### 6.32. Timer3 Scalar Register (*tm3s*), IO address = 0x36

| Bit | Reset | R/W | Description |
|---|---|---|---|
| 7 | 0 | WO | PWM resolution selection.<br>0 : 8-bit<br>1 : 6-bit or 7-bit (by code option TMx_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. |

### 6.33. Timer3 Bound Register (*tm3b*), IO address = 0x37

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

©Copyright 2020, PADAUK Technology Co. Ltd

## 7. Instructions

| Symbol | Symbol |
|---|---|
| **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<br>(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 0~0x7F (0~127) is allowed |

©Copyright 2020, PADAUK Technology Co. Ltd

## 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* pb, a;<br>Result: pb ← 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>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.<br>Example: *stt16* word;<br>Result: 16-bit timer ←word<br>Affected flags: 『N』Z 『N』C 『N』AC 『N』OV<br>Application Example:<br>---<br>word T16val ; // declare a RAM word<br>…<br>*mov* a, 0x34 ;<br>*mov* lb@ T16val , a ; // move 0x34 to T16val (LSB)<br>*mov* a, 0x12 ;<br>*mov* hb@ T16val , a ; // move 0x12 to T16val (MSB)<br>*stt16* T16val ; // initial T16 with 0x1234<br>…<br>--- |
| *Idxm a, index* | Move data from specified memory to ACC by indirect method. It needs 2T to execute this instruction.<br>Example: idxm a, index;<br>Result: a ← [index], where index is declared by word.<br>Affected flags: 『N』Z 『N』C 『N』AC 『N』OV<br>Application Example:<br>---<br>word RAMIndex ; // declare a RAM pointer<br>…<br>mov a, 0x5B ; // assign pointer to an address (LSB)<br>mov lb@RAMIndex, a ; // save pointer to RAM (LSB)<br>mov a, 0x00 ; // assign 0x00 to an address (MSB), should be 0<br>mov hb@RAMIndex, a ; // save pointer to RAM (MSB)<br>…<br>idxm a, RAMIndex ; // move memory data in address 0x5B to ACC<br>--- |
| *Idxm index, a* | Move data from ACC to specified memory by indirect method. It needs 2T to execute this instruction.<br>Example: idxm index, a;<br>Result: [index] ← a; where index is declared by word.<br>Affected flags: 『N』Z 『N』C 『N』AC 『N』OV<br>Application Example:<br>---<br>word RAMIndex ; // declare a RAM pointer<br>…<br>mov a, 0x5B ; // assign pointer to an address (LSB)<br>mov lb@RAMIndex, a ; // save pointer to RAM (LSB)<br>mov a, 0x00 ; // assign 0x00 to an address (MSB), should be 0<br>mov hb@RAMIndex, a ; // save pointer to RAM (MSB)<br>…<br>mov a, 0xA5 ;<br>idxm RAMIndex, a ; // mov 0xA5 to memory in address 0x5B<br>--- |

| | |
|---|---|
| *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 |

## 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 |

©Copyright 2020, PADAUK Technology Co. Ltd

## 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 |

©Copyright 2020, PADAUK Technology Co. Ltd

## 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 |

©Copyright 2020, PADAUK Technology Co. Ltd

| | |
|---|---|
| *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>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>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* pb.5 ;<br>Result: set bit 5 of port B 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>...<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>---<br>Application Example2 (serial input) :<br>---<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>--- |
| *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 |

©Copyright 2020, PADAUK Technology Co. Ltd

## 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>EX: *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 |

©Copyright 2020, PADAUK Technology Co. Ltd

| | |
|---|---|
| *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 that is 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 |
| *wdreset* | Reset Watchdog timer.<br>Example: *wdreset* ;<br>Result: Reset Watchdog timer.<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>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 CPU<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 CPU<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 |

## 7.8. Summary of Instructions Execution Cycle

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

## 7.9. Summary of affected flags by Instructions

| Instruction | Z | C | AC | OV | Instruction | Z | C | AC | OV | Instruction | Z | C | AC | OV |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| *mov* a, I | - | - | - | - | *mov* M, a | - | - | - | - | *mov* a, M | Y | - | - | - |
| *mov* a, IO | Y | - | - | - | *mov* IO, a | - | - | - | - | *ldt16* word | - | - | - | - |
| *stt16* word | - | - | - | - | *idxm* a, index | - | - | - | - | *idxm* index, a | - | - | - | - |
| *xch* M | - | - | - | - | *pushaf* | - | - | - | - | *popaf* | Y | Y | Y | Y |
| *add* a, I | Y | Y | Y | Y | *add* a, M | Y | Y | Y | Y | *add* M, a | Y | Y | Y | Y |
| *addc* a, M | Y | Y | Y | Y | *addc* M, a | Y | Y | Y | Y | *addc* a | Y | Y | Y | Y |
| *addc* M | Y | Y | Y | Y | *nadd a, M* | Y | Y | Y | Y | *nadd M, a* | 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 | - | - | - |
| *comp a, M* | Y | Y | Y | Y | *comp M, a* | Y | Y | Y | Y | *set0* IO.n | - | - | | |
| *set1* IO.n | - | | - | - | *set0* M.n | - | - | - | - | *set1* M.n | - | - | - | - |
| *swapc IO.n* | - | Y | - | - | *ceqsn* a, I | Y | Y | Y | Y | *ceqsn* a, M | Y | Y | Y | Y |
| *cneqsn a,M* | Y | Y | Y | Y | *cneqsn a, I* | Y | Y | Y | Y | *t0sn* IO.n | - | - | | |
| *t1sn* IO.n | | | - | - | *t0sn* M.n | - | - | - | - | *t1sn* M.n | - | - | - | - |
| *izsn* a | Y | Y | Y | Y | *dzsn* a | Y | Y | Y | Y | *izsn* M | Y | Y | Y | Y |
| *dzsn* M | Y | Y | Y | Y | *call* label | | | | | *goto* label | - | - | - | - |
| *ret* I | - | - | - | - | *ret* | - | - | - | - | *reti* | - | - | - | - |
| *nop* | - | - | - | - | *pcadd* a | - | - | - | - | *engint* | - | - | - | - |
| *disgint* | - | - | - | - | *stopsys* | - | - | - | - | *stopexe* | - | - | - | - |
| *reset* | - | - | - | - | *wdreset* | - | - | - | - | | | | | |

## 7.10. BIT definition

Bit access of RAM is only available for address from 0x00 to 0x7F.

©Copyright 2020, PADAUK Technology Co. Ltd

## 8. Code Options

| Option | Selection | Description |
|---|---|---|
| Security | Enable | MTP content is protected and program cannot be read back |
| | Disable | MTP content is not protected so program can be read back |
| LVR | 4.0V | Select LVR = 4.0V |
| | 3.5V | Select LVR = 3.5V |
| | 3.0V | Select LVR = 3.0V |
| | 2.7V | Select LVR = 2.7V |
| | 2.5V | Select LVR = 2.5V |
| | 2.2V | Select LVR = 2.2V |
| | 2.0V | Select LVR = 2.0V |
| | 1.8V | Select LVR = 1.8V |
| Boot-up_Time | Slow | Please refer to $t_{WUP}$ and $t_{SBP}$ in Section 4.1 |
| | Fast | Please refer to $t_{WUP}$ and $t_{SBP}$ in Section 4.1 |
| Interrupt Src0 | PA.0 | INTEN/ INTRQ.Bit0 is from PA.0 |
| | PB.5 | INTEN/ INTRQ.Bit0 is from PB.5 |
| Interrupt Src1 | PB.0 | INTEN/ INTRQ.Bit1 is from PB.0 |
| | PA.4 | INTEN/ INTRQ.Bit1 is from PA.4 |
| PB4_PB7_Drive | Normal | PB4 & PB7 Drive / Sink Current is Normal |
| | Strong | PB4 & PB7 Drive / Sink Current is Strong (ICE does NOT support.) |
| Comparator Edge | All_Edge | The comparator will trigger an interrupt on both 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 |
| GPC_PWM | Disable | Comparator does not control all PWM outputs |
| | Enable | Comparator controls all PWM outputs (ICE does NOT support.) |
| TMx_Source | 16MHZ | When tm2c[7:4]= 0010, TM2 clock source = IHRC = 16MHZ<br>When tm3c[7:4]= 0010, TM3 clock source = IHRC = 16MHZ |
| | 32MHZ | When tm2c[7:4]= 0010, TM2 clock source = IHRC*2 = 32MHZ<br>When tm3c[7:4]= 0010, TM3 clock source = IHRC*2 = 32MHZ<br>(ICE does NOT support.) |
| TMx_Bit | 6 Bit | When tm2s.7=1, TM2 PWM resolution is 6 Bit<br>When tm3s.7=1, TM3 PWM resolution is 6 Bit |
| | 7 Bit | When tm2s.7=1, TM2 PWM resolution is 7 Bit<br>When tm3s.7=1, TM3 PWM resolution is 7 Bit<br>(ICE does NOT support.) |

©Copyright 2020, PADAUK Technology Co. Ltd

# 9. Special Notes

This chapter is to remind user who use PFS122 series IC in order to avoid frequent errors upon operation.

## 9.1. Warning

User must read all application notes of the IC by detail before using it. Please download the related application notes from the following link:
http://www.padauk.com.tw/tw/technical/index.aspx

## 9.2. Using IC

### 9.2.1. IO pin usage and setting

(1) IO pin is set to be digital input

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

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

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

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

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

(4) PA7 and PA6 as external crystal oscillator

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

Note: Please read the PMC-APN013 carefully. According to PMC-APN013, the crystal oscillator should be used reasonably. If the following situations happen to cause IC start-up slowly or non-startup, PADAUK

Technology is not responsible for this: the quality of the user's crystal oscillator is not good, the usage conditions are unreasonable, the PCB cleaner leakage current, or the PCB layouts are unreasonable.

### 9.2.2. Interrupt

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

Step1: Set INTEN register, enable the interrupt control bit

Step2: Clear INTRQ register

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

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

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

* Use DISGINT in the main program to disable all interrupts
* When interrupt service routine starts, use PUSHAF instruction to save ALU and FLAG register. POPAF instruction is to restore ALU and FLAG register before RETI as below:

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

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

(3) There are two sets of external IO pin interrupt source. Every set is decided by code option Interrupt Src0 and Interrupt Src1 corresponding to the unique interrupt pin. Please comply with the ***inten / intrq / integs*** register when selecting IO pin.

### 9.2.3. System clock switching

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

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

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

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

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

### 9.2.4. Watchdog

Watchdog will be inactive once ILRC is disabled.

©Copyright 2020, PADAUK Technology Co. Ltd

### 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

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.

©Copyright 2020, PADAUK Technology Co. Ltd

### 9.2.8. Programming Writing

Please use PDK5S-P-003 to program. PDK3S-P-002 or older versions do not support programming PFS122.
Jumper connection: Please follow the instruction inside the writer software to connect the jumper.
Please select the following program mode according to the actual situation.

Normal Programming Mode

Range of application:

- Single-Chip-Package IC with programming at the writer IC socket or on the handler.
- Multi-Chip-Package(MCP) with PFS122. Be sure its connected IC and devices will not be damaged by the following voltages, and will not clam the following voltages.

**The voltage conditions in normal programming mode:**

(1) VDD is 7.5V, and the maximum supply current is up to about 20mA.
(2) PA5 is 5.5V.
(3) The voltages of other program pins (except GND) are the same as VDD.

**Important Cautions：**

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

Limited-Voltage Programming Mode

Range of application:

- On-Board writing. Its peripheral circuits and devices will not be damaged by the following voltages, and will not clam the following voltages. Please refer to On-Board Writing for more details.
- Multi-Chip-Package(MCP) with PFS122. Please be sure that its connected IC and devices will not be damaged by the following voltages, and will not clam the following voltages.

**The voltage conditions in Limited-Voltage programming mode:**

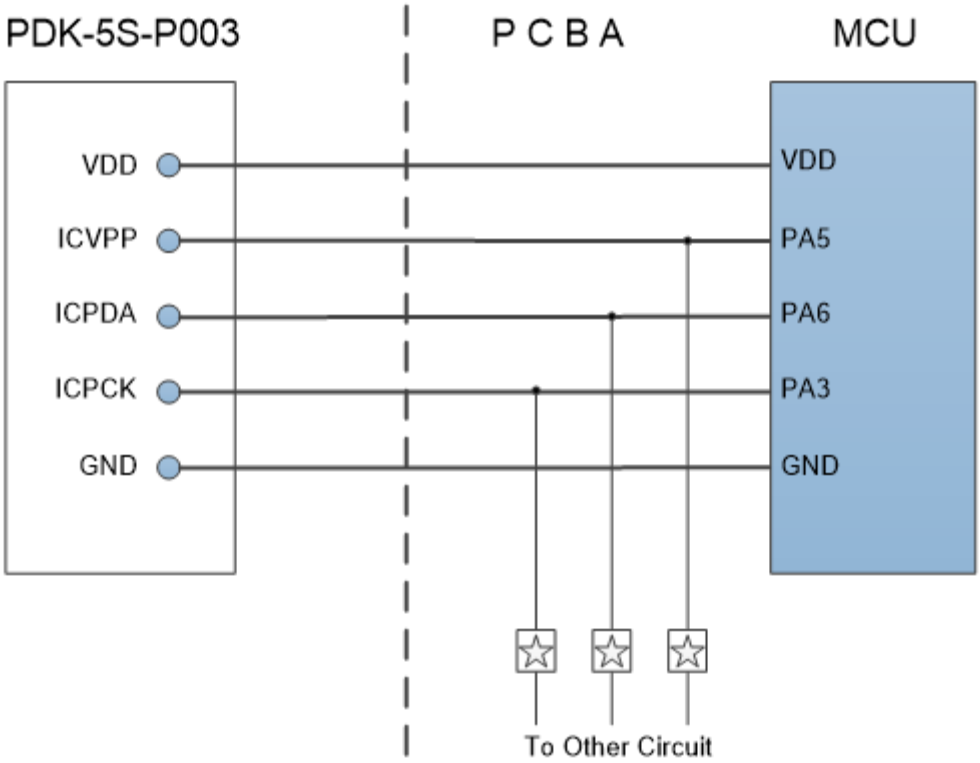
(1) VDD is 5.0V, and the maximum supply current is up to about 20mA.
(2) PA5 is 5.0V.
(3) The voltage of other program pins (except GND) is the same as VDD.

Please select "MTP On-Board VDD Limitation" or "On-Board Program" on the writer screen to enable the limited-voltage programming mode. (Please refer to the file of Writer “PDK5S-P-003 UM”).

©Copyright 2020, PADAUK Technology Co. Ltd

On-board Writing

PFS122 can support On-board writing. On-Board Writing is known as the situation that the IC has to be programmed when the IC itself and other peripheral circuits and devices have already been mounted on the PCB. Five wires of PDK5S-P-003 are used for On-Board Writing: ICPCK, ICPDA, VDD, GND and ICVPP. They are used to connect PA3, PA6, VDD, GND and PA5 of the IC correspondingly.




The above figure shows the connection for PFS122 on-board writing. In this figure, ☆can be either resistors or capacitors. They are used to isolate the programming signal wires from the peripheral circuit. it should be≧10KΩfor resistance while ≦220pF for capacitance.

**Notice:**

- In general, the limited-voltage programming mode is used in On-board Writing, Please refers to the 13.2 for more detail about limited-voltage programming mode.
- Any zener diode ≦5.0V, or any circuitry which clam the 5.0V to be created SHOULD NOT be connected between VDD and GND of the PCB.
- Any capacitor ≧500uF SHOULD NOT be connected between VDD and GND of the PCB.
- In general, the writing signal pins PA3, PA5 and PA6 should not be considered as strong output pins.

©Copyright 2020, PADAUK Technology Co. Ltd

## 9.3. Using ICE

(1) PDK5S-I-S01/2(B) supports PFS122 MCU emulation, the following items should be noted when using PDK5S-I-S01/2(B) to emulate PFS122:

- PDK5S-I-S01/2(B) doesn't support the instruction NMOV/SWAP/NADD/COMP with RAM.
- PDK5S-I-S01/2(B) doesn't support SYSCLK=ILRC/16.
- PDK5S-I-S01/2(B) doesn't support the function ***Tm2.gpcrs***/***Tm3.gpcrs***.
- PDK5S-I-S01/2(B) doesn't support the code options: PB4_PB7_Drive, GPC_PWM, TMx_source and TMx_bit.
- PDK5S-I-S01/2(B) doesn't support PAPL, PBPL.
- When using GPCC output, PA3 will be influenced.
- When simulating PWM waveform, please check the waveform during program running. When the ICE is suspended or single-step running, its waveform may be inconsistent with the reality.
- The ILRC frequency of the PDK5S-I-S01/2(B) simulator is different from the actual IC and is uncalibrated, with a frequency range of about 34K~38KHz.
- Fast Wakeup time is different from PDK5S-I-S01/2(B): 128 SYSCLK, PFS122: 45 ILRC
- Watch dog time out period is different from PDK5S-I-S01/2(B):

| WDT period | PDK5S-I-S01/2(B) | PFS122 |
|---|---|---|
| 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 * $T_{ILRC}$ |

©Copyright 2020, PADAUK Technology Co. Ltd