# P232CS14 / P232CD14
# P232CS20 / P232CD20

# P232 Family
# ADC-Type Enhanced
# Field Programmable Processor Array
# (FPPA$^{TM}$)

## Preliminary Data Sheet

*Version 0.21 – February 28, 2011*

**10F-2, No. 1, Sec. 2, Dong-Da Road, Hsin-Chu 300, Taiwan, R.O.C.**

TEL: 886-3-532-7598 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.**

# Table of Contents

Preliminary Information, subject to change without notice

Preliminary Information, subject to change without notice

## Revision History:

| Revision | Date | Description |
|---|---|---|
| 0.11 | 2010/03/02 | 1st version |
| 0.12 | 2010/06/18 | 1. Correct EOSCR Read/Write register to Write only register<br>2. Correct PCDIDR register description<br>3. Correct Fig.10 Timer2 Hardware Diagram<br>Add ~PA0, ~PA3, ~PA4 clock sources<br>4. Modify Fig.4-14~4-24, misc.6=1 to disable XTAL high drive mode<br>5. Add comparator functional description and responsive time chart |
| 0.13 | 2010/07/08 | Amend band-gap reference voltage specification |
| 0.20 | 2010/10/01 | 1. Amend operating voltage range<br>2. Amend adjustable frequency of IHRC<br>3. Amend ADC requirement<br>4. Amend band-gap reference voltage range |
| 0.21 | 2011/02/28 | 1. Add description of PA7, PA6, PA5 Pin (3)<br>2. Amend Device Characteristics of $f_{IHRC}$, $f_{ILRC}$ (4-1)<br>3. Add Typical measurement of ILRC frequency vs. Temperature (4-4)<br>4. Amend Typical measurement of IHRC frequency vs. Temperature (4-25)<br>5. Correct Fig. 8 Options of System Clock (5-7-4)<br>6. Correct Fig. 11 Timing diagram of Timer2 in PWM Mode (5-9) |

# 1. Features

## 1-1. High Performance RISC CPU Array

- Patented Field Programmable Processor Array (FPPA™) Technology
- Two parallel processing units
- 2Kx16 bits OTP program memory for both FPP units
- 200 Bytes data RAM for both FPP units
- 100 powerful instructions
- All instructions are 1T except indirect memory access, including branch instructions
- One cycle for branch instructions to reduce overhead and easy timing calculation
- Programmable stack pointer to provide adjustable stack level
- Direct and indirect addressing modes for data and instructions
- Bit-manipulation instructions
- All data memories are available for use as an index pointer
- Support security function to protect OTP data
- Separated IO and memory space to reduce firmware overhead in space exchange

## 1-2. System Functions

- Clock sources : internal high RC oscillator (IHRC), internal low RC oscillator (ILRC) and crystal oscillator
- Built-in Power On Reset and Low Voltage Detector
- Built-in precise internal high RC oscillator, low drift with voltage
- One hardware 16-bit timer
- One hardware 8-bit timer with PWM generator
- Up to 10-channel 12-bit resolution ADC with 1-channel for internal band-gap reference voltage
- Built-in Internal band-gap to provide 1.20±0.20 volt reference voltage
- Provide two start up speed modes and two wake-up speed modes
- Provide one hardware comparator
- High noise immunity (high EFT)
- 18 IO pins with 12mA driving / sink capability
- Four configurable Low Voltage Detector (LVD) levels to ensure workable voltage
- Operating voltage range: 2.5V ~ 5.5V
- Operating temperature range: -40°C ~ 85°C
- Operating frequency and voltage for both crystal mode and IHRC mode

  DC ~ 8MHz@VDD≧4V DC ~ 4MHz@VDD≧3V

  DC ~ 2MHz@VDD≧2.5V
- Low power consumption

  $I_{operating}$ ~ 1.2mA@1MIPS, VDD=5.0V; $I_{operating}$ ~ 40uA@VDD=3.3V, ILRC ~ 16KHz

  $I_{standby}$ ~ 1uA@VDD=5.0V; $I_{standby}$ ~ 0.5uA@VDD=3.3V

- Package types:

  P232CS14: SOP14 (150mil), P232CD14: DIP14 (300mil)

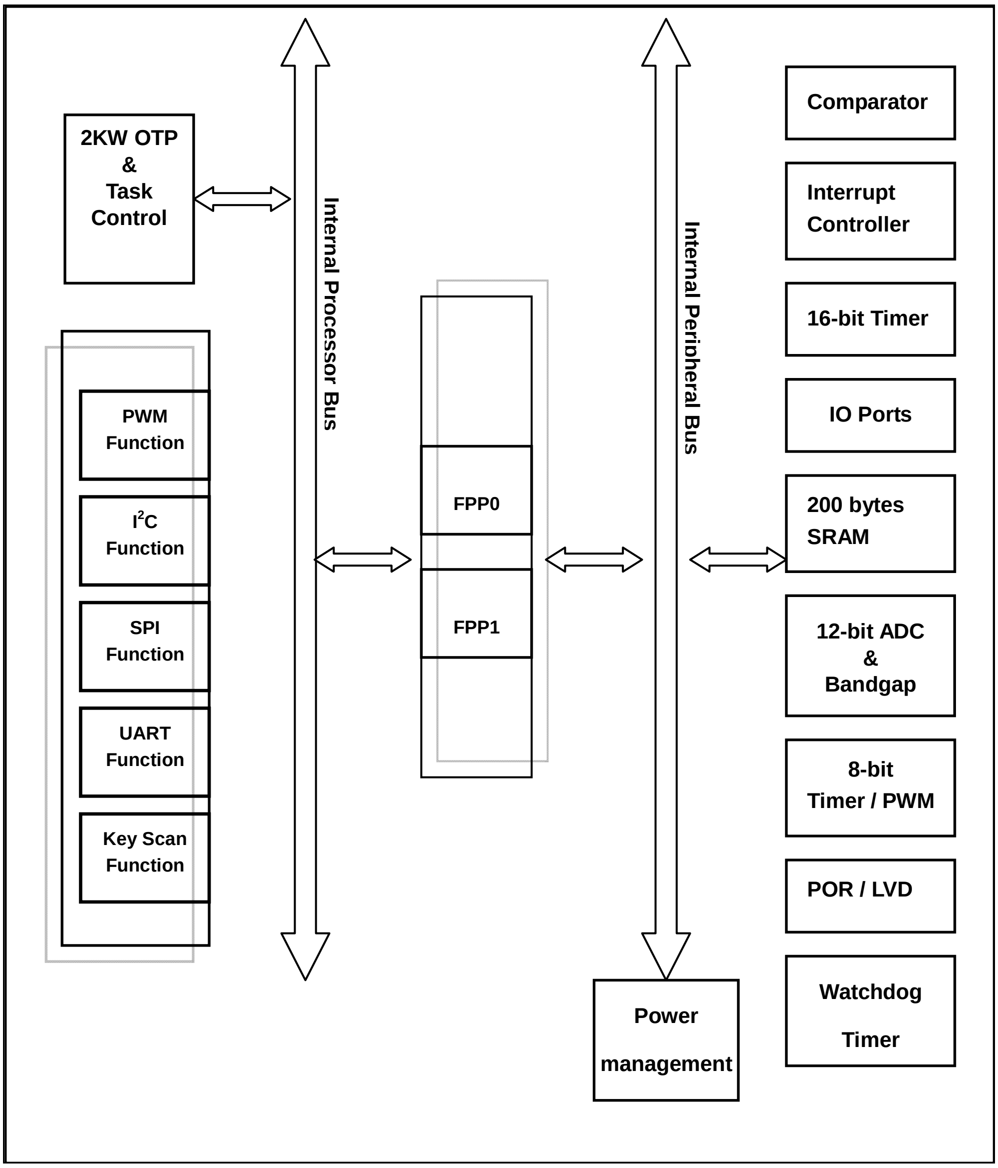
  P232CS20: SOP20 (300mil), P232CD20: DIP20 (300mil)

Preliminary Information, subject to change without notice

## 2. General Description and Block Diagram

The P232 family is an ADC-Type of PADAUK's parallel processing, fully static, OTP-based CMOS 2x8 bit processor array that can execute two peripheral functions in parallel. It employs RISC architecture based on patent pending FPPA™ (Field Programmable Processor Array) technology and all the instructions are executed in one cycle except that some instructions are two cycles that handle indirect memory access.

2Kx16 bits OTP program memory and 200 bytes data SRAM are inside for two FPP units using, one up to 10 channels 12-bit ADC is built inside the chip with one channel for internal band-gap reference voltage; one comparator is also provided. P232 also provides two hardware timers: one is 16-bit timer and the other one is 8-bit with PWM generation.

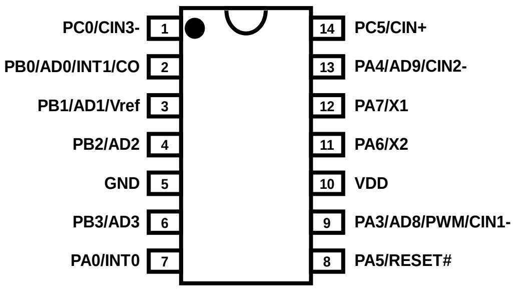


Preliminary Information, subject to change without notice

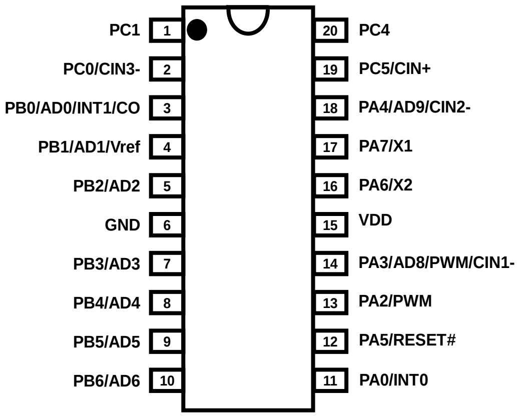
## 3. P232 Family and Pin Description

**P232CS14 (SOP14-150mil)**
**P232CD14 (DIP14-300mil)**

**P232CS20 (SOP20-300mil)**
**P232CD20 (DIP20-300mil)**



P232CS14 (SOP14-150mil)
P232CD14 (DIP14-300mil)



P232CS20 (SOP20-300mil)
P232CD20 (DIP20-300mil)

### Pin Description

| Pin Name | Pin Type & Buffer Type | Description |
|---|---|---|
| PA7/X1 | IO<br>ST / CMOS | This pin can be used as (1) Bit 7 of port A when an internal RC oscillator is used and can be configured as input/output, with pull-up resistor, open-drain output mode by software. (2) X1 when a crystal oscillator is used.<br>When this pin is configured for crystal oscillator, it will be switched to analog pin to prevent leakage current automatically. The bit 7 of ***padidr*** register can be set to high to disable digital input; wake-up from power-down by toggling this pin is also disabled.<br>For the application needs High EFT. Please use PA7 pin as output pin or add a resistor (at least 1Kohm) in series when use as input pin. |
| PA6/X2 | IO<br>ST / CMOS | This pin can be used as (1) Bit 6 of port A when an external crystal oscillator is not used and can be configured as input/output, with pull-up resistor, open-drain output mode by software. (2) X2 when a crystal oscillator is used.<br>When this pin is configured for crystal oscillator, it will be switched to analog pin to prevent leakage current automatically. The bit 6 of ***padidr*** register can be set to high to disable digital input, wake-up from power-down by toggling this pin is also disabled.<br>For the application needs High EFT. Please use PA6 pin as output pin or add a resistor (at least 1Kohm) in series when use as input pin. |

| Pin Name | Pin Type &<br>Buffer Type | Description |
|---|---|---|
| PA5/RESET# | IO (OC)<br>ST / CMOS | This pin can be used as (1) hardware reset of this chip. (2) Bit 5 of port A. Please note that this pin will drive low level only for output mode and does NOT have pull-up or pull-down resistor inside. Please add pull up resistor (input mode) outside or drive low (output) to prevent leakage current if no use. The bit 5 of ***padidr*** register can be set to high to disable wake-up from power-down by toggling this pin.<br>Please avoid using PA5 pin as input pin. If you can only use PA5 pin as input pin, be sure to have a resistor (at least 100ohm) in series. |
| PA4/AD9/CIN2- | IO<br>ST / CMOS /<br>Analog | This pin can be used as (1) Bit 4 of port A. It can be configured as digital input, two-state output and open-drain output mode with pull-up resistor by software independently (2) Channel 9 of ADC analog input (3) Minus input source of comparator.<br>When this pin is configured as analog input, please use bit 4 of register ***padidr*** to disable the digital input to prevent current leakage. The bit 4 of ***padidr*** register can be set to high to disable digital input, wake-up from power-down by toggling this pin is also disabled. |
| PA3/AD8/PWM/<br>CIN1- | IO<br>ST / CMOS /<br>Analog | This pin can be used as (1) Bit 3 of port A. It can be configured as digital input, two-state output and open-drain output mode with pull-up resistor independently by software (2) Channel 8 of ADC analog input (3) Minus input source of comparator (4) PWM output from Timer2.<br>When this pin is configured as analog input, please use bit 3 of register ***padidr*** to disable the digital input to prevent current leakage. The bit 3 of ***padidr*** register can be set to high to disable digital input; wake-up from power-down by toggling this pin is also disabled. |
| PA2/PWM | IO<br>ST / CMOS | This pin can be used as (1) Bit 2 of port A. It can be configured as digital input, two-state output and open-drain output mode with pull-up resistor independently by software (2) PWM output from Timer2.<br>The bit 2 of ***padidr*** register can be set to high to disable wake-up from power-down by toggling this pin. |
| PA0/INT0 | IO<br>ST / CMOS | This pin can be used as (1) Bit 0 of port A. It can be configured as digital input, two-state output and open-drain output mode with pull-up resistor independently by software (2) 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>The bit 0 of ***padidr*** register can be set to high to disable wake-up from power-down by toggling this pin. |

| Pin Name | Pin Type & Buffer Type | Description |
|---|---|---|
| PB6/AD6<br>PB5/AD5<br>PB4/AD4<br>PB3/AD3<br>PB2/AD2 | IO<br>ST / CMOS /<br>Analog | These pins can be used as (1) Bit 6~2 of port B. These five pins can each be configured as analog input, digital input, and two-state output mode with pull-up resistor independently by software. (2) Channel 6~2 of ADC analog input. When any of these five pins acts as analog inputs, please use register ***pbdidr*** to disable the digital input to prevent current leakage. If the control bit in ***pbdidr*** register is set to high to disable digital input, wake-up from power-down by toggling the corresponding pin is also disabled. |
| PB1/AD1/Vref | IO<br>ST / CMOS /<br>Analog | This pin can be used as (1) Bit 1 of port B. It can be configured as analog input, digital input, and two-state output mode with pull-up resistor independently by software. (2) Channel 1 of ADC analog input (3) Reference high input Vref of ADC.<br>When this pin acts as analog input, please use bit 1 of register ***pbdidr*** to disable the digital input to prevent current leakage. If bit 1 of ***pbdidr*** register is set to high to disable digital input, wake-up from power-down by toggling this pin is also disabled. |
| PB0/AD0/INT1/<br>CO | IO<br>ST / CMOS /<br>Analog | This pin can be used as (1) Bit 0 of port B. It can be configured as analog input, digital input, and two-state output mode with pull-up resistor independently by software. (2) Channel 0 of analog input. When this pin acts as analog input, please use bit 0 of register ***pbdidr*** to disable the digital input to prevent current leakage. (3) External interrupt line 1. Both rising edge and falling edge are accepted to request interrupt service and configurable by register setting. (4) Comparator output.<br>If bit 0 of ***pbdidr*** register is set to high to disable digital input, wake-up from power-down by toggling this pin is also disabled. |
| PC5/CIN+ | IO<br>ST / CMOS /<br>Analog | This pin can be used as (1) Bit 5 of port C. It can be configured as analog input, digital input, and two-state output mode with pull-up resistor independently by software. (2) Plus or minus input source of comparator.<br>When this pin acts as analog input, please use bit 1 of register ***pcdidr*** to disable the digital input to prevent current leakage. If bit 1 of ***pcdidr*** register is set to high to disable digital input, wake-up from power-down by toggling this pin is also disabled. |
| PC4<br>PC1 | IO<br>ST / CMOS | Bit 4 and 1 of port C. These pins can be configured as digital input mode and two-state output mode with pull-up resistor independently by software. If the control bit in ***pcdidr*** register is set to high to disable digital input, wake-up from power-down by toggling the corresponding pin is also disabled. |

| Pin Name | Pin Type &<br>Buffer Type | Description |
|---|---|---|
| PC0/CIN3- | IO<br>ST / CMOS /<br>Analog | This pin can be used as (1) Bit 0 of port C. It can be configured as analog input, digital input, and two-state output mode with pull-up resistor independently by software. (2) Minus input source of comparator.<br>When this pin acts as analog input, please use register ***pcdidr*** to disable the digital input to prevent current leakage. If bit 0 of ***pcdidr*** register is set to high to disable digital input, wake-up from power-down by toggling this pin is also disabled. |
| VDD | VDD | Positive power |
| GND | GND | Ground |

**Notes:** IO: Input/Output; ST: Schmitt Trigger input; OC: Open Collector; Analog: Analog input pin;
CMOS: CMOS voltage level

# 4. Device Characteristics

## 4-1. AC/DC Device Characteristics

| Symbol | Description | Min | Typ | Max | Unit | Conditions (Ta=25℃) |
|---|---|---|---|---|---|---|
| $V_{DD}$ | Operating Voltage | 2.5 | 5.0 | 5.5 | V | |
| $I_{OP}$ | Operating Current | | 1.2<br>3.5<br>150<br>40 | | mA<br>mA<br>uA<br>uA | $f_{SYS}$=1MIPS@5.0V<br>$f_{SYS}$=8MIPS@5.0V<br>$f_{SYS}$=5.0V，use ILRC ~ 32KHz<br>$f_{SYS}$=3.3V，use ILRC ~ 16KHz |
| $I_{PD}$ | Power Down Current | | <1.0<br><0.5 | | uA<br>uA | $f_{SYS}$= 0Hz,VDD=5.0V<br>$f_{SYS}$= 0Hz,VDD=3.3V |
| $V_{IL}$ | Input low voltage for IO lines | 0 | | 0.2$V_{DD}$ | V | |
| $V_{IH}$ | Input high voltage for IO lines | 0.8 $V_{DD}$ | | $V_{DD}$ | V | |
| $I_{OL}$ | IO lines sink current | 8 | 12 | 18 | mA | $V_{DD}$=5.0V, $V_{OL}$=0.5V |
| $I_{OH}$ | IO lines drive current | -7.5 | -11.5 | -17 | mA | $V_{DD}$=5.0V, $V_{OH}$=4.5V |
| $R_{PH}$ | Pull-high Resistance | | 75 | | KΩ | $V_{DD}$=5.0V |
| $V_{BRD}$ | Low Voltage Detect Voltage (Brown-out voltage) | 3.8<br>2.85<br>2.35<br>2.1 | 4.0<br>3.0<br>2.5<br>2.2 | 4.2<br>3.15<br>2.65<br>2.3 | V | |
| $f_{SYS}$ | System clock<br>IHRC & crystal oscillator<br>IHRC & crystal oscillator<br>IHRC & crystal oscillator<br>Internal low RC oscillator | <br>0<br>0<br>0<br> | <br><br><br><br>32K | <br>8M<br>4M<br>2M<br> | Hz | <br>$V_{DD}$ = 4.0V<br>$V_{DD}$ = 3.0V<br>$V_{DD}$ = 2.5V<br>$V_{DD}$ = 5.0V |
| $t_{INT}$ | Interrupt pulse width | 30 | | | ns | $V_{DD}$ = 5.0V |
| $V_{ADC}$ | Workable ADC operating Voltage | 2.5 | | 5.0 | V | |
| $V_{AD}$ | AD Input Voltage | 0 | | VDD | V | |
| ADrs | ADC resolution | | | 12 | bit | |
| ADcs | ADC current consumption | | 0.9<br>0.8 | | mA | @5V<br>@3V |
| ADclk | ADC clock period | | 2 | | us | 2.5V ~ 5.5V |
| $t_{ADCONV}$ | ADC conversion time ($T_{ADCLK}$ is the period of the selected AD conversion clock) | | 13<br>14<br>15<br>16<br>17 | | $T_{ADCLK}$ | 8-bit resolution<br>9-bit resolution<br>10-bit resolution<br>11-bit resolution<br>12-bit resolution |
| AD DNL | ADC Differential NonLinearity | | ±2* | | LSB | |
| AD INL | ADC Integral NonLinearity | | ±4* | | LSB | |
| ADos | ADC offset* | | 10<br>6.5<br>4 | | mV | VDD=5V<br>VDD=4V<br>VDD=3.3V |
| $V_{BG}$ | Bandgap Reference Voltage | 1.00 | 1.20 | 1.40 | V | |
| $V_{DR}$ | RAM data retention voltage* | 1.5 | | | V | P232 in stop mode. |

| Symbol | Description | Min | Typ | Max | Unit | Conditions (Ta=25℃) |
|---|---|---|---|---|---|---|
| $f_{IHRC}$ | Frequency of IHRC after calibration@25°C * | 15.84<br>15.52<br>14.72<br>14.08 | 16<br>16<br>16<br>16 | 16.16<br>16.48<br>17.28<br>17.92 | MHz | ±1% @VDD=5V, 25°C<br>±3% @VDD=2.5V~5.5V, 25°C<br>±8% @VDD=2.5V~5.5V, 0°C~70°C<br>±12% @VDD=2.5V~5.5V, -40°C~85°C |
| $f_{ILRC}$ | Frequency of ILRC | | 32<br>16 | | KHz | ±35% @VDD=5V & -40°C<Ta<85°C<br>±35% @VDD=3.3V & -40°C<Ta<85°C |
| CPos | Comparator offset* | - | ±10 | ±20 | mV | |
| CPcm | Comparator input common mode* | 0 | | VDD-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 | | 40<br>20 | | uA | @5.0V<br>@3.3V |
| $t_{WDT}$ | Watchdog timeout period | | 16<br>32 | | ms | @VDD=5V, ILRC~32KHz<br>@VDD=3.3V, ILRC~16KHz |
| $t_{SBP}$ | System boot-up period (Fast) | | 20 | | us | @VDD=5V, IHRC=16MHz |
| | System boot-up period (Normal) | | 32 | | ms | @VDD=5V, ILRC~32KHz |
| $t_{WUP}$ | System wake up period (Fast) | | 20 | | us | @IHRC=16MHz |
| | System wake up period (Normal) | | 32<br>64 | | ms | @VDD=5V, ILRC~32KHz<br>@VDD=3.3V, ILRC~16KHz |
| $t_{RST}$ | External reset pulse width | 120 | | | us | @VDD=5V |

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

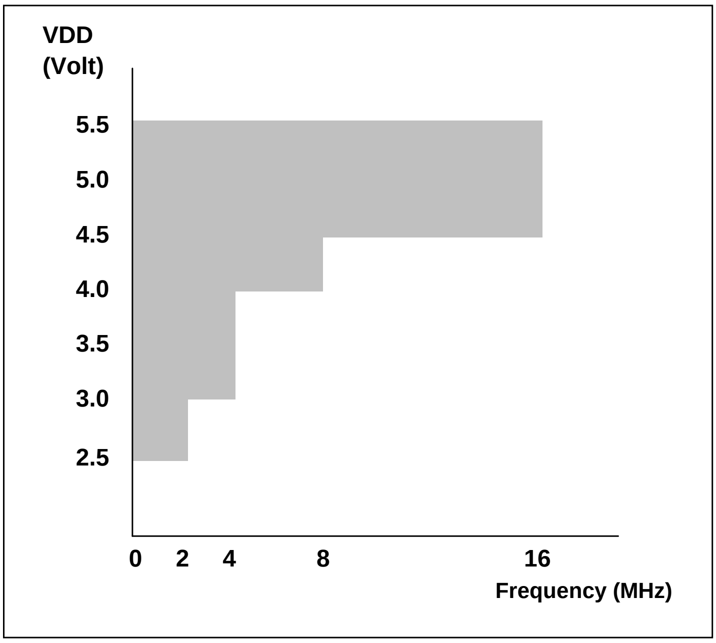
** Response time is measured with comparator input at (VDD-1.5)/2 -100mV, and (VDD-1.5)/2+100mV

## 4-2. Absolute Maximum Ratings

- Supply Voltage ………………………………...... 2.5V ~ 5.5V
- Input Voltage ………………………………….. -0.3V ~ VDD + 0.3V
- Operating Temperature ……………………… -40℃ ~ 85℃
- Junction Temperature ……………………….. 150℃
- Storage Temperature ………………………… -50℃ ~ 125℃

Preliminary Information, subject to change without notice

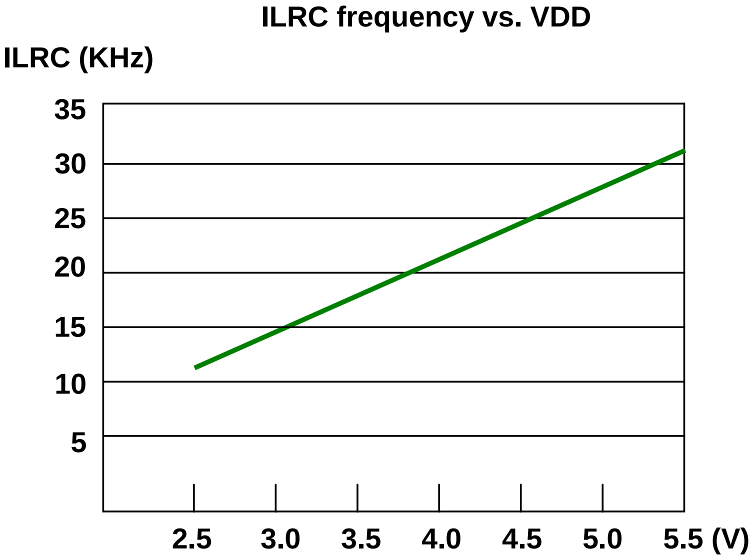
## 4-3. Typical operating frequency vs. VDD



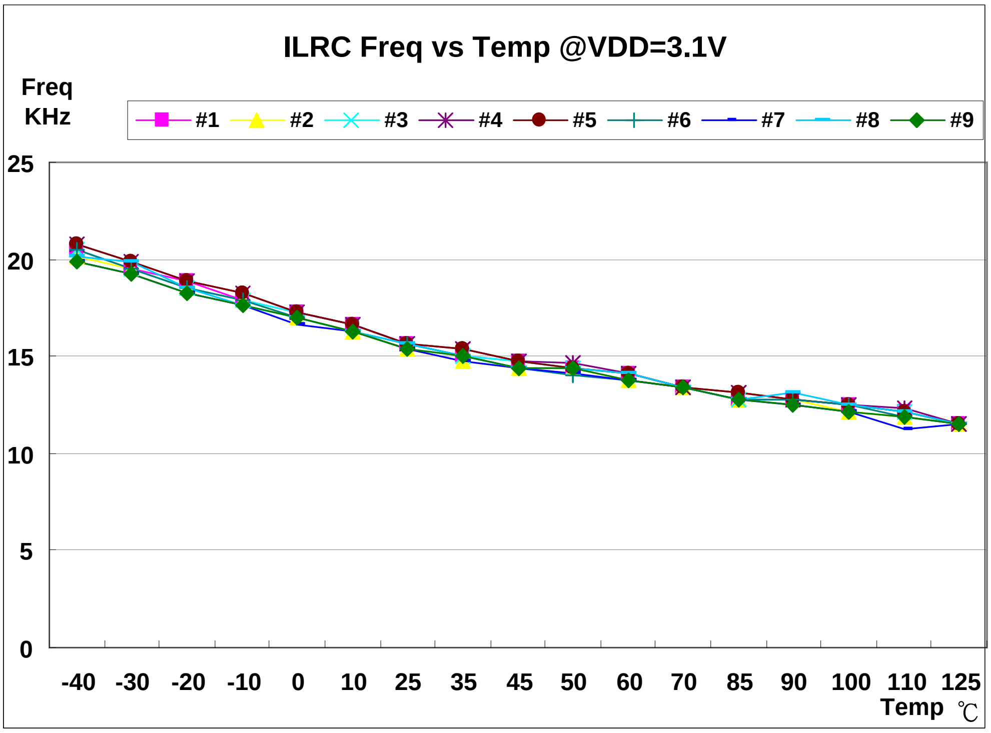
The allowable operating frequency of P232 is
2MHz for 2.5V,
4MHz for 3.0V,
8MHz for 4.0V and
16MHz for 4.5V.
Here, the operating frequency is the frequency of system clock.

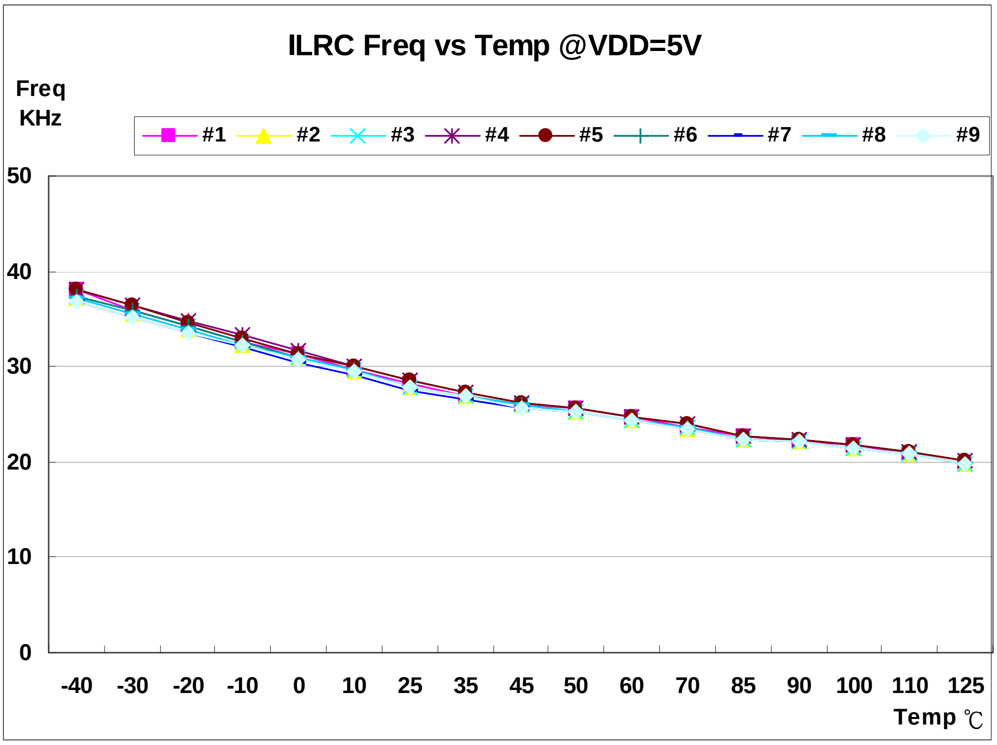
**Note: The shaded region indicates the workable region of frequency and voltage**

## 4-4. Typical measurement of ILRC frequency vs. VDD and temperature



Preliminary Information, subject to change without notice

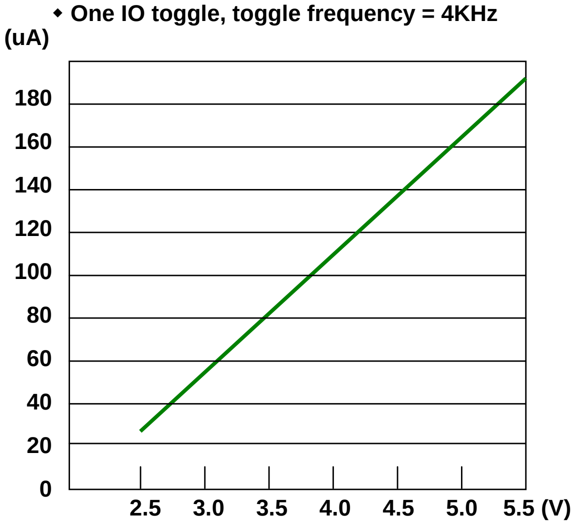




Preliminary Information, subject to change without notice

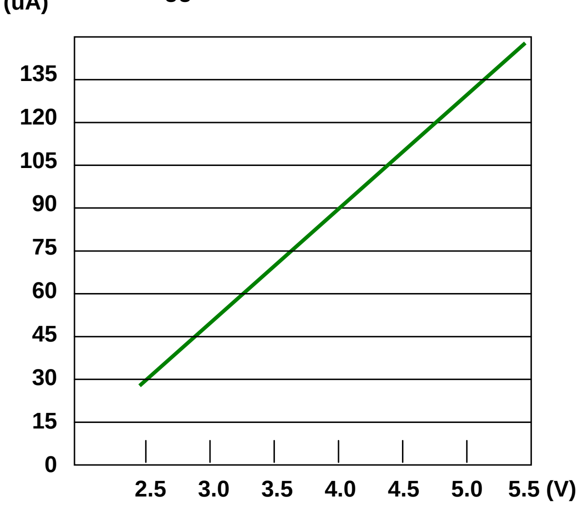
## 4-5. Typical operating current vs. VDD @ system clock = ILRC/1

Measurement conditions:
- system CLK = ILRC ~ 32KHz @ 5V
- eoscr.0 = 1, IHRC and LVD disabled
- 2 FPPA running
- One IO toggle, toggle frequency = 4KHz



Note: ILRC frequency will change with voltage

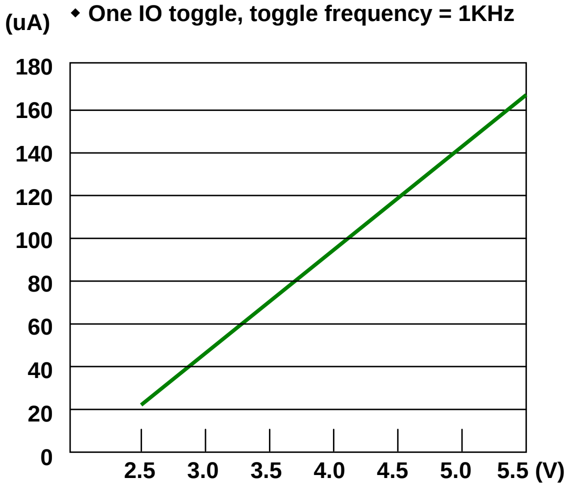
Measurement conditions:
- system CLK = ILRC ~ 32KHz @5V
- eoscr.0 = 1, IHRC and LVD disabled
- 1 FPPA running
- No IO toggle



Note: ILRC frequency will change with voltage

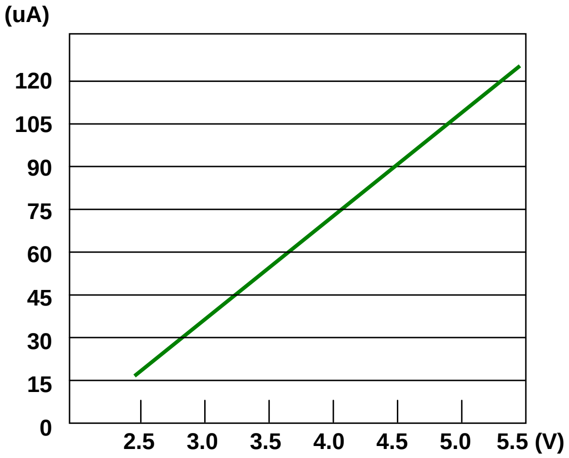
## 4-6. Typical operating current vs. VDD @ system clock = ILRC/4

Measurement conditions:
- system CLK = ILRC/4 ~ 8KHz @5V
- eoscr.0 = 1, IHRC and LVD disabled
- 2 FPPA running
- One IO toggle, toggle frequency = 1KHz



Note: ILRC frequency will change with voltage

Measurement conditions:
- system CLK = ILRC/4 ~ 8KHz @5V
- eoscr.0 = 1, IHRC and LVD disabled
- 1 FPPA running
- No IO toggle



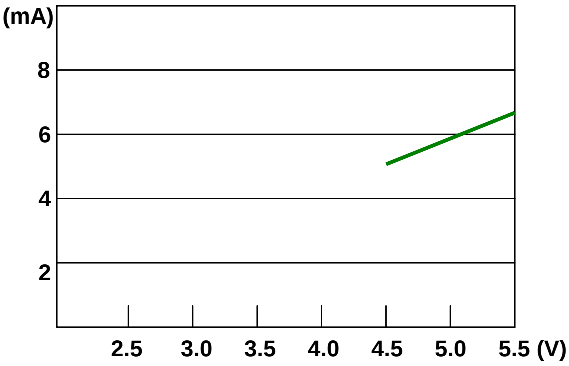
Note: ILRC frequency will change with voltage

Preliminary Information, subject to change without notice

## 4-7. Typical operating current vs. VDD @ system clock = IHRC/1

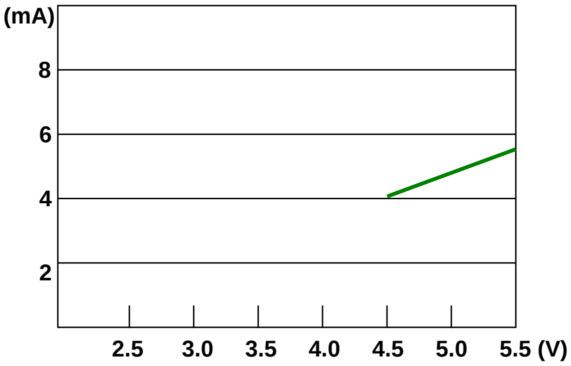
**Measurement conditions:**

- IHRC=16MHz, system CLK = IHRC/1 = 16MHz
- eoscr.0 = 0, IHRC and LVD enabled
- 2 FPPA running
- One IO toggle, toggle frequency = 2MHz



**Measurement conditions:**

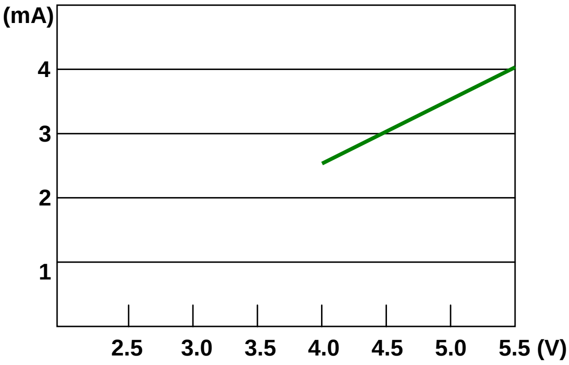
- IHRC=16MHz, system CLK = IHRC/1 = 16MHz
- eoscr.0 = 0, IHRC and LVD enabled
- 1 FPPA running
- NO IO toggle



## 4-8. Typical operating current vs. VDD @ system clock = IHRC/2

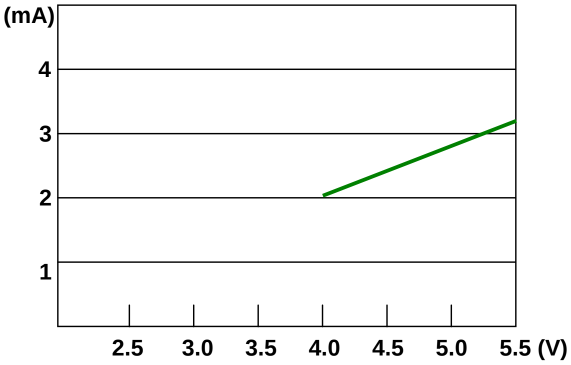
**Measurement conditions:**

- IHRC=16MHz, system CLK = IHRC/2 = 8MHz
- eoscr.0 = 0, IHRC and LVD enabled
- 2 FPPA running
- One IO toggle, toggle frequency = 1MHz



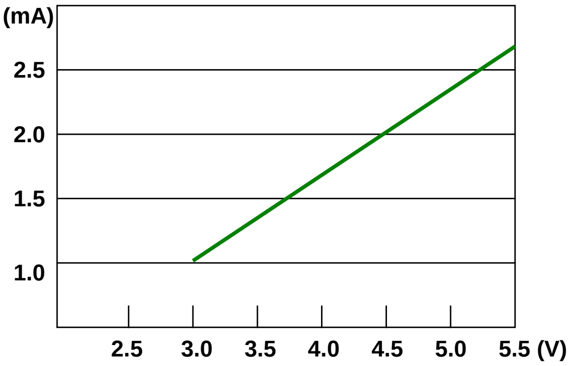
**Measurement conditions:**

- IHRC=16MHz, system CLK = IHRC/2 = 8MHz
- eoscr.0 = 0, IHRC and LVD enabled
- 1 FPPA running
- No IO toggle

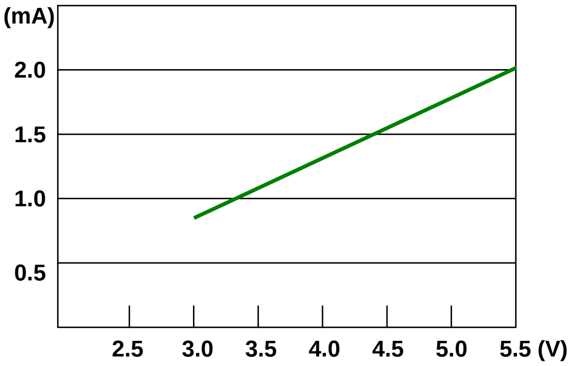


## 4-9. Typical operating current vs. VDD @ system clock = IHRC/4

**Measurement conditions:**
- IHRC=16MHz, system CLK = IHRC/4 = 4MHz
- eoscr.0 = 0, IHRC and LVD enabled
- 2 FPPA running
- One IO toggle, toggle frequency = 500KHz

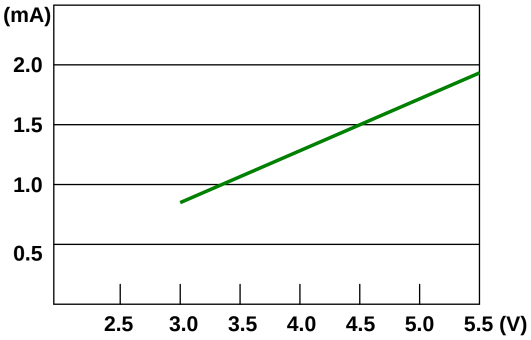


**Measurement conditions:**
- IHRC=16MHz, system CLK = IHRC/4 = 4MHz
- eoscr.0 = 0, IHRC and LVD enabled
- 1 FPPA running
- No IO toggle

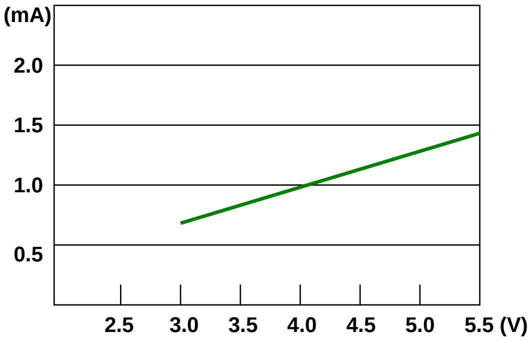


## 4-10. Typical operating current vs. VDD @ system clock = IHRC/6

**Measurement conditions:**
- IHRC=16MHz, system CLK = IHRC/6 = 2.6MHz
- eoscr.0 = 0, IHRC and LVD enabled
- 2 FPPA running
- One IO toggle, toggle frequency = 333KHz



**Measurement conditions:**
- IHRC=16MHz, system CLK = IHRC/6 = 2.6MHz
- eoscr.0 = 0, IHRC and LVD enabled
- 1 FPPA running
- No IO toggle

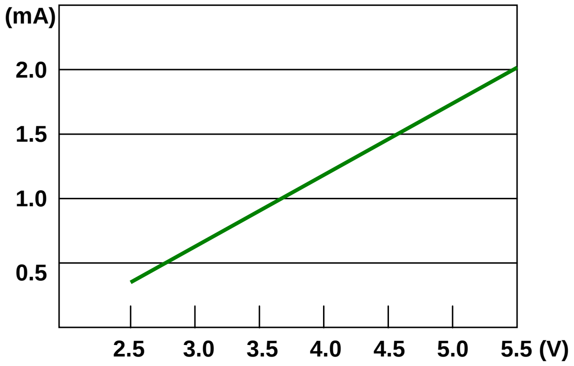


Preliminary Information, subject to change without notice

## 4-11. Typical operating current vs. VDD @ system clock = IHRC/8

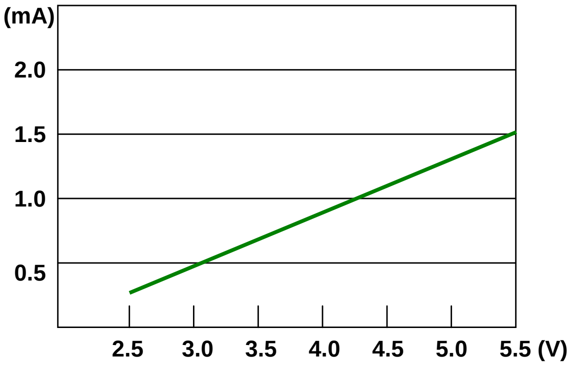
**Measurement conditions:**

- IHRC=16MHz, system CLK = IHRC/8 = 2MHz
- eoscr.0 = 0, IHRC and LVD enabled
- 2 FPPA running
- One IO toggle, toggle frequency = 250KHz



**Measurement conditions:**

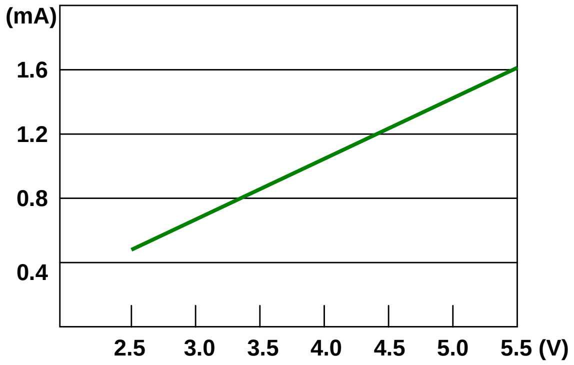
- IHRC=16MHz, system CLK = IHRC/8 = 2MHz
- eoscr.0 = 0, IHRC and LVD enabled
- 1 FPPA running
- No IO toggle



## 4-12. Typical operating current vs. VDD @ system clock = IHRC/16

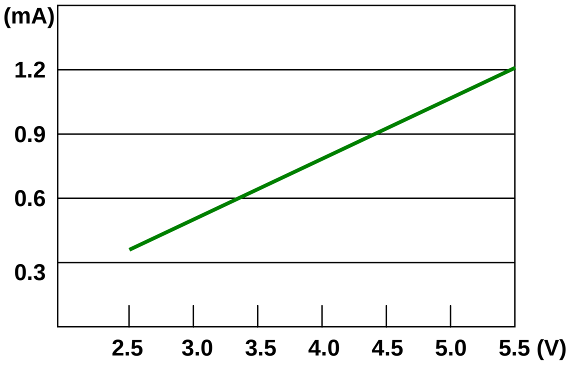
**Measurement conditions:**

- IHRC=16MHz, system CLK = IHRC/16 = 1MHz
- eoscr.0 = 0, IHRC and LVD enabled
- 2 FPPA running
- One IO toggle, toggle frequency = 125KHz



**Measurement conditions:**

- IHRC=16MHz, system CLK = IHRC/16 = 1MHz
- eoscr.0 = 0, IHRC and LVD enabled
- 1 FPPA running
- No IO toggle

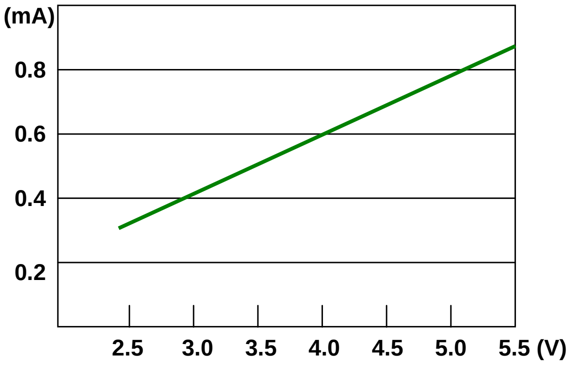


Preliminary Information, subject to change without notice

## 4-13. Typical operating current vs. VDD @ system clock = IHRC/32

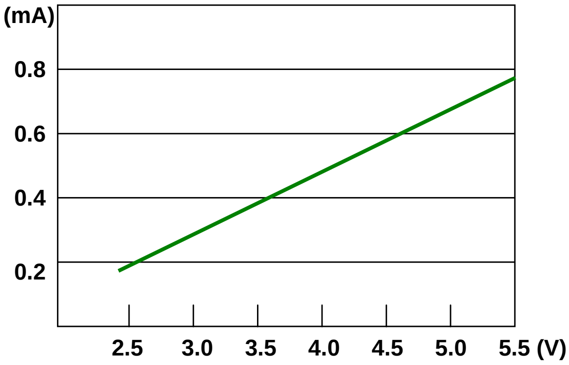
**Measurement conditions:**

- IHRC=16MHz, system CLK = IHRC/32 = 500KHz
- eoscr.0 = 0, IHRC and LVD enabled
- 2 FPPA running
- One IO toggle, toggle frequency = 62.5KHz



**Measurement conditions:**

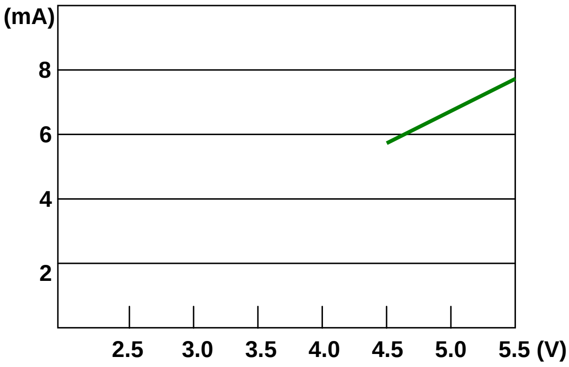
- IHRC=16MHz, system CLK = IHRC/32 = 500KHz
- eoscr.0 = 0, IHRC and LVD enabled
- 1 FPPA running
- No IO toggle



## 4-14. Typical operating current vs. VDD @ system clock = 16MHz XTAL (EOSC) /1

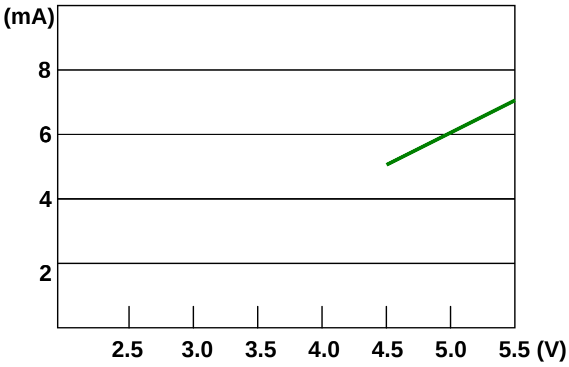
**Measurement Conditions:**

- XTAL=16MHz, system CLK = EOSC/1 = 16MHz
- eoscr.0 = 0, IHRC enabled
- 2 FPPA running
- misc.6 = 1, disable XTAL high drive mode
- One IO toggle, toggle frequency = 2MHz



**Measurement Conditions:**

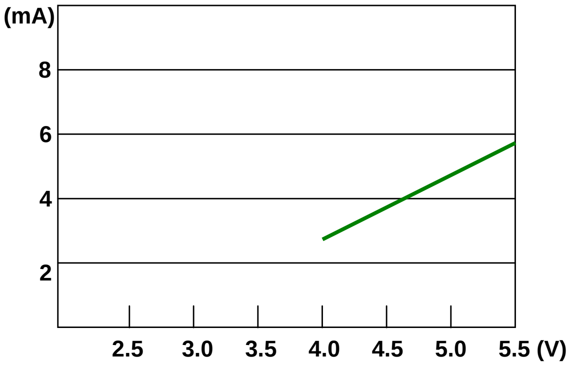
- XTAL=16MHz, system CLK = EOSC/1 = 16MHz
- eoscr.0 = 0, IHRC enabled
- 1 FPPA running
- misc.6 = 1, disable XTAL high drive mode
- No IO toggle



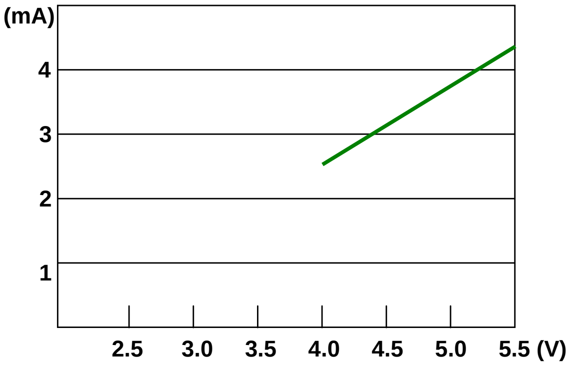
Preliminary Information, subject to change without notice

## 4-15. Typical operating current vs. VDD @ system clock = 16MHz XTAL (EOSC) /2

**Measurement Conditions:**
- XTAL=16MHz, system CLK = EOSC/2 = 8MHz
- eoscr.0 = 0, IHRC enabled
- 2 FPPA running
- misc.6 = 1, disable XTAL high drive mode
- One IO toggle, toggle frequency = 1MHz

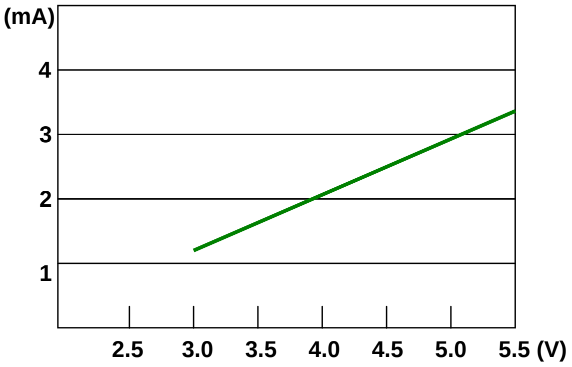


**Measurement Conditions:**
- XTAL=16MHz, system CLK = EOSC/2 = 8MHz
- eoscr.0 = 0, IHRC enabled
- 1 FPPA running
- misc.6 = 1, disable XTAL high drive mode
- No IO toggle

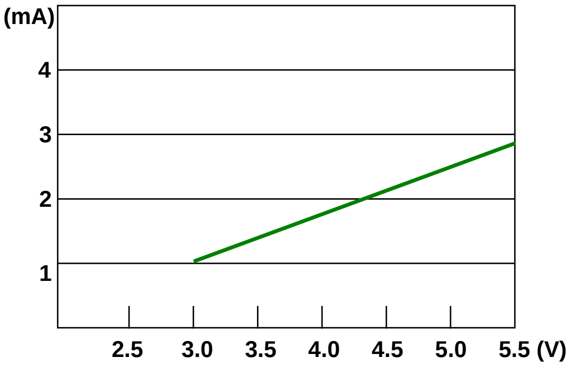


## 4-16. Typical operating current vs. VDD @ system clock = 16MHz XTAL (EOSC) /4

**Measurement Conditions:**
- XTAL=16MHz, system CLK = EOSC/4 = 4MHz
- eoscr.0 = 0, IHRC enabled
- 2 FPPA running
- misc.6 = 1, disable XTAL high drive mode
- One IO toggle, toggle frequency = 500kHz



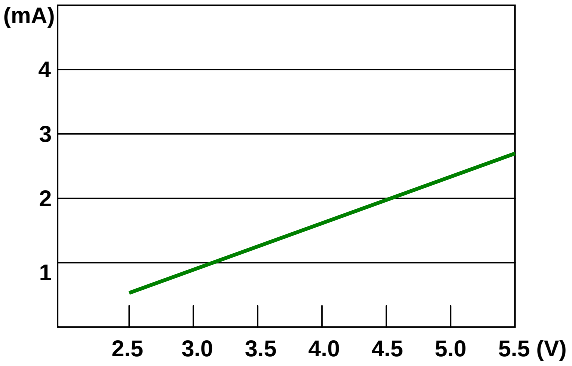
**Measurement Conditions:**
- XTAL=16MHz, system CLK = EOSC/4 = 4MHz
- eoscr.0 = 0, IHRC enabled
- 1 FPPA running
- misc.6 = 1, disable XTAL high drive mode
- No IO toggle



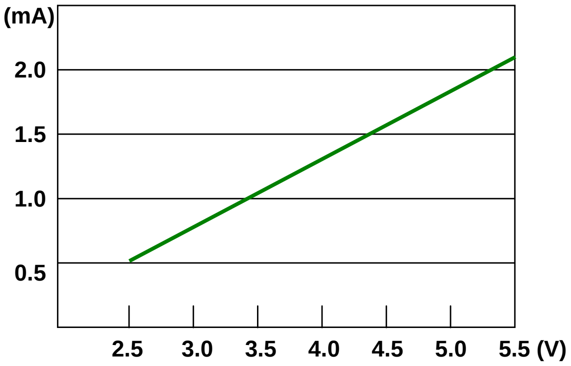
Preliminary Information, subject to change without notice

## 4-17. Typical operating current vs. VDD @ system clock = 16MHz XTAL (EOSC) /8

**Measurement Conditions:**
- XTAL=16MHz, system CLK = EOSC/8 = 2MHz
- eoscr.0 = 0, IHRC enabled
- 2 FPPA running
- misc.6 = 1, disable XTAL high drive mode
- One IO toggle, toggle frequency = 250kHz

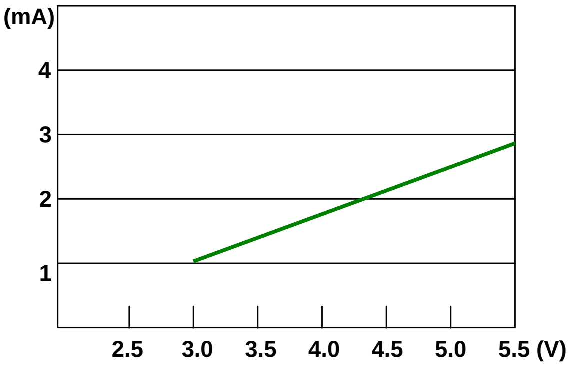


**Measurement Conditions:**
- XTAL=16MHz, system CLK = EOSC/8 = 2MHz
- eoscr.0 = 0, IHRC enabled
- 1 FPPA running
- misc.6 = 1, disable XTAL high drive mode
- No IO toggle

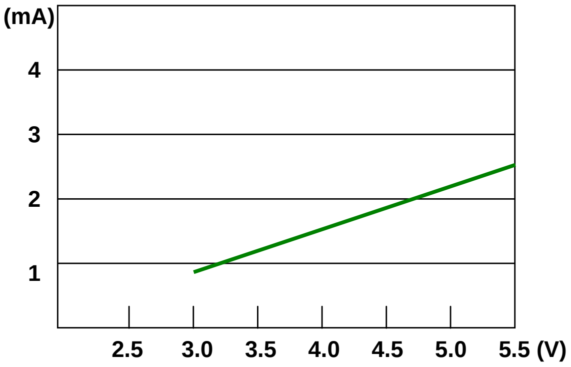


## 4-18. Typical operating current vs. VDD @ system clock = 4MHz XTAL (EOSC) /1

**Measurement Conditions:**
- XTAL=4MHz, system CLK = EOSC/1 = 4MHz
- eoscr.0 = 0, IHRC enabled
- 2 FPPA running
- misc.6 = 1, disable XTAL high drive mode
- One IO toggle, toggle frequency = 500kHz



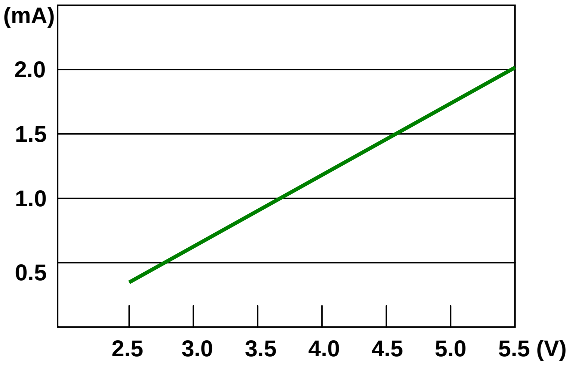
**Measurement Conditions:**
- XTAL=4MHz, system CLK = EOSC/1 = 4MHz
- eoscr.0 = 0, IHRC enabled
- 1 FPPA running
- misc.6 = 1, disable XTAL high drive mode
- No IO toggle



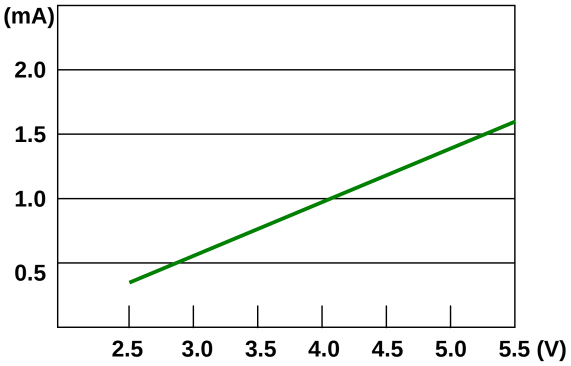
Preliminary Information, subject to change without notice

## 4-19. Typical operating current vs. VDD @ system clock = 4MHz XTAL (EOSC) /2

**Measurement Conditions:**
- XTAL=4MHz, system CLK = EOSC/2 = 2MHz
- eoscr.0 = 0, IHRC enabled
- 2 FPPA running
- misc.6 = 1, disable XTAL high drive mode
- One IO toggle, toggle frequency = 250kHz

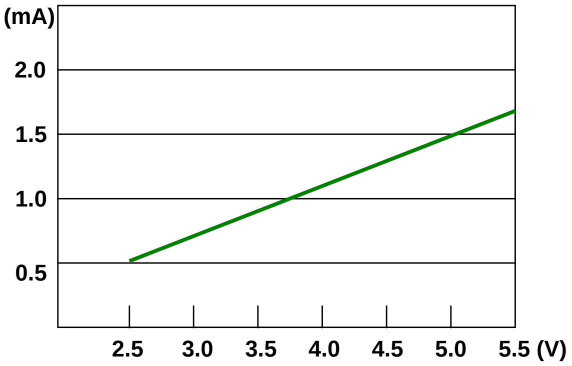


**Measurement Conditions:**
- XTAL=4MHz, system CLK = EOSC/2 = 2MHz
- eoscr.0 = 0, IHRC enabled
- 1 FPPA running
- misc.6 = 1, disable XTAL high drive mode
- No IO toggle

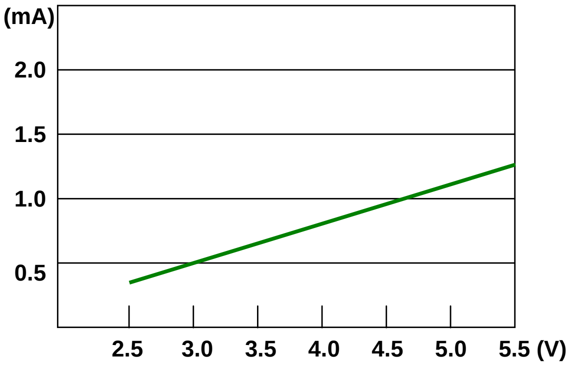


## 4-20. Typical operating current vs. VDD @ system clock = 4MHz XTAL (EOSC) /4

**Measurement Conditions:**
- XTAL=4MHz, system CLK = EOSC/4 = 1MHz
- eoscr.0 = 0, IHRC enabled
- 2 FPPA running
- misc.6 = 1, disable XTAL high drive mode
- One IO toggle, toggle frequency = 125kHz



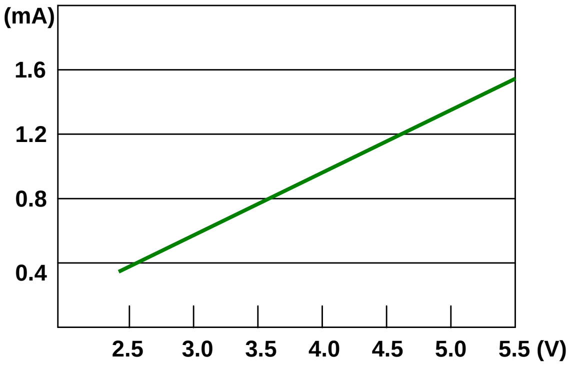
**Measurement Conditions:**
- XTAL=4MHz, system CLK = EOSC/4 = 1MHz
- eoscr.0 = 0, IHRC enabled
- 1 FPPA running
- misc.6 = 1, disable XTAL high drive mode
- No IO toggle



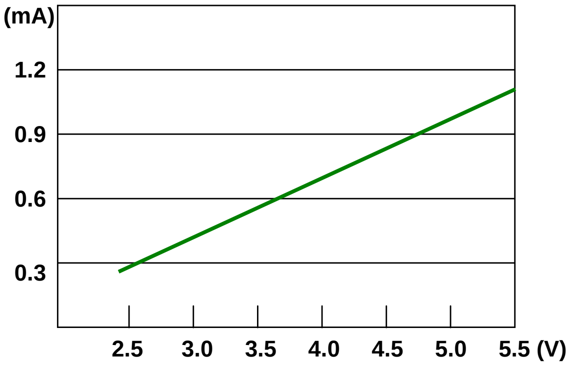
Preliminary Information, subject to change without notice

## 4-21. Typical operating current vs. VDD @ system clock = 4MHz XTAL (EOSC) /8

**Measurement Conditions:**
- XTAL=4MHz, system CLK = EOSC/8 = 500KHz
- eoscr.0 = 0, IHRC enabled
- 2 FPPA running
- misc.6 = 1, disable XTAL high drive mode
- One IO toggle, toggle frequency = 62.5kHz

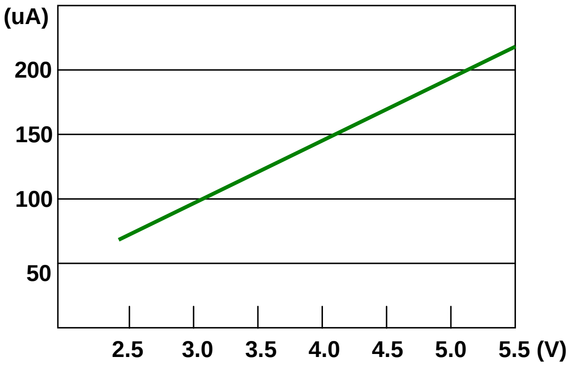


**Measurement Conditions:**
- XTAL=4MHz, system CLK = EOSC/8 = 500KHz
- eoscr.0 = 0, IHRC enabled
- 1 FPPA running
- misc.6 = 1, disable XTAL high drive mode
- No IO toggle

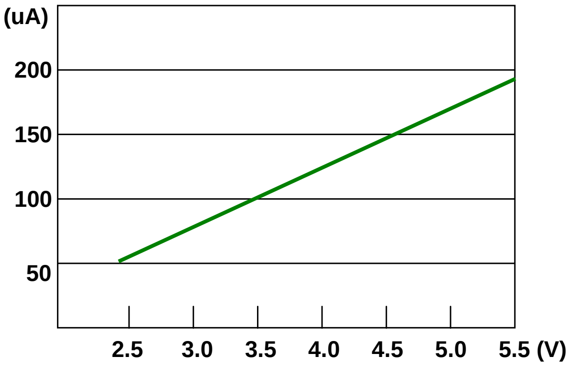


## 4-22. Typical operating current vs. VDD @ system clock = 32KHz XTAL (EOSC) /1

**Measurement Conditions:**
- XTAL=32KHz, system CLK = EOSC/1 = 32KHz
- eoscr.0 = 0, IHRC enabled
- 2 FPPA running
- misc.6 = 1, disable XTAL high drive mode
- One IO toggle, toggle frequency = 4KHz



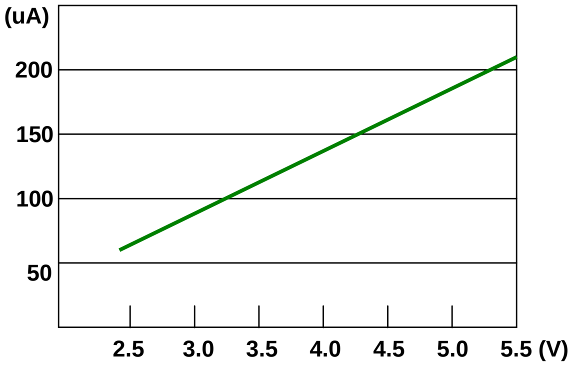
**Measurement Conditions:**
- XTAL=32KHz, system CLK = EOSC/1 = 32KHz
- eoscr.0 = 0, IHRC enabled
- 1 FPPA running
- misc.6 = 1, disable XTAL high drive mode
- No IO toggle



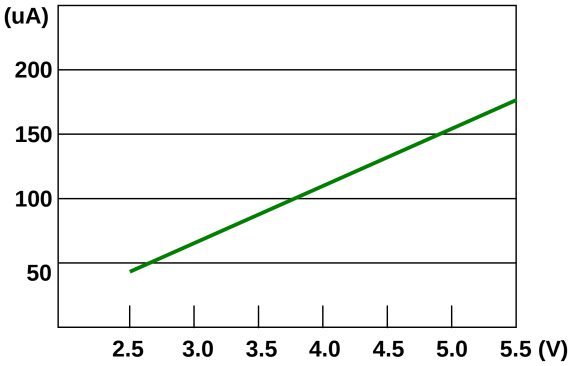
Preliminary Information, subject to change without notice

## 4-23. Typical operating current vs. VDD @ system clock = 32KHz XTAL (EOSC) /2

**Measurement Conditions:**
- XTAL=32KHz, system CLK = EOSC/2 = 16KHz
- eoscr.0 = 0, IHRC enabled
- 2 FPPA running
- misc.6 = 1, disable XTAL high drive mode
- One IO toggle, toggle frequency = 2KHz

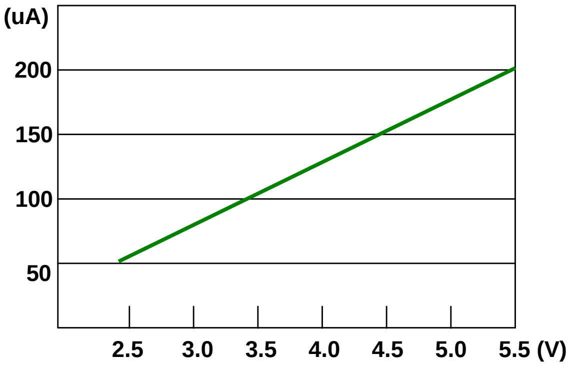


**Measurement Conditions:**
- XTAL=32KHz, system CLK = EOSC/2 = 16KHz
- eoscr.0 = 0, IHRC enabled
- 1 FPPA running
- misc.6 = 1, disable XTAL high drive mode
- No IO toggle

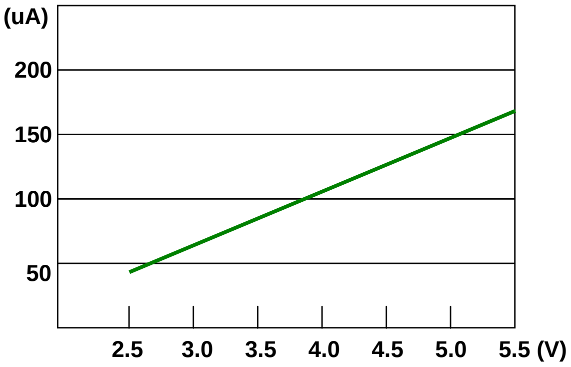


## 4-24. Typical operating current vs. VDD @ system clock = 32KHz XTAL (EOSC) /4

**Measurement Conditions:**
- XTAL=32KHz, system CLK = EOSC/4 = 8KHz
- eoscr.0 = 0, IHRC enabled
- 2 FPPA running
- misc.6 = 1, disable XTAL high drive mode
- One IO toggle, toggle frequency = 1KHz

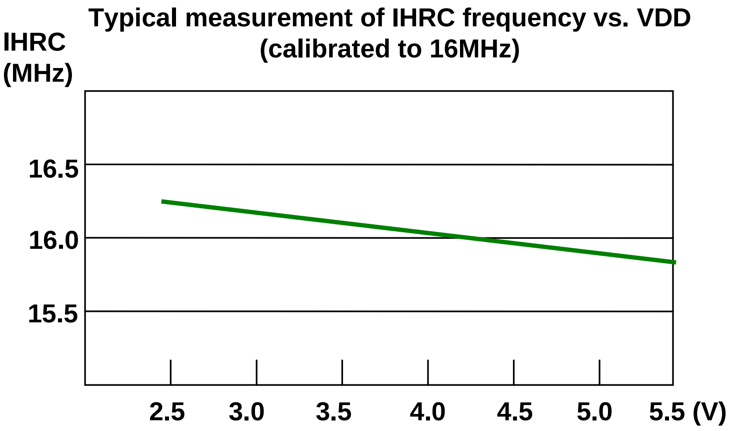


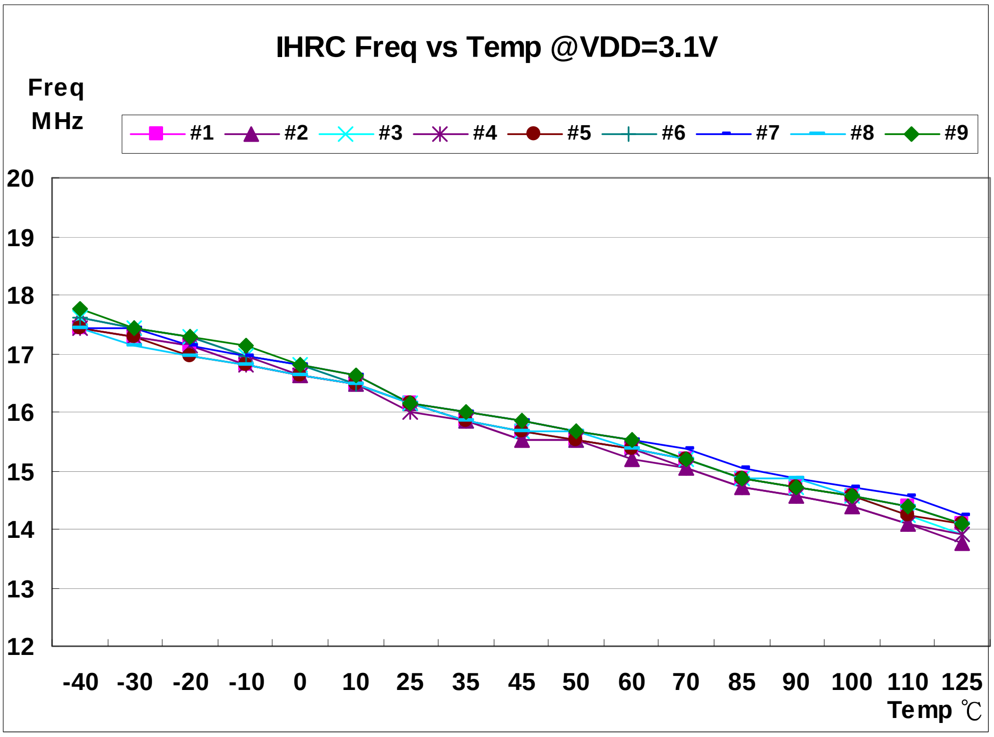
**Measurement Conditions:**
- XTAL=32KHz, system CLK = EOSC/4 = 8KHz
- eoscr.0 = 0, IHRC enabled
- 1 FPPA running
- misc.6 = 1, disable XTAL high drive mode
- No IO toggle



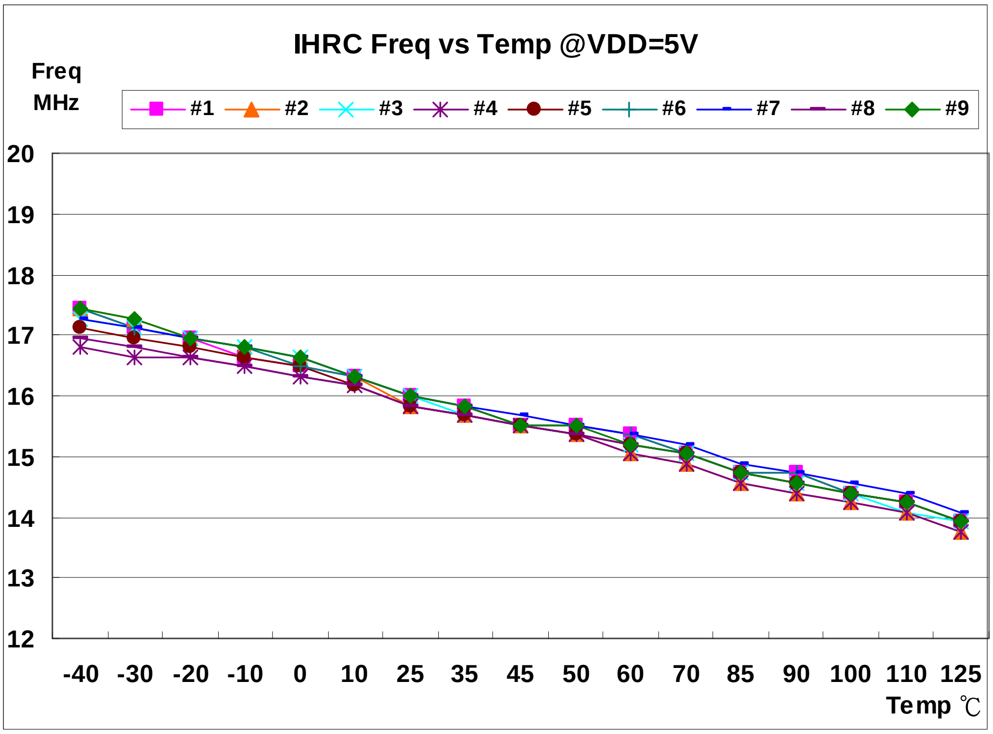
Preliminary Information, subject to change without notice

## 4-25. Typical measurement of IHRC frequency vs. VDD and temperature

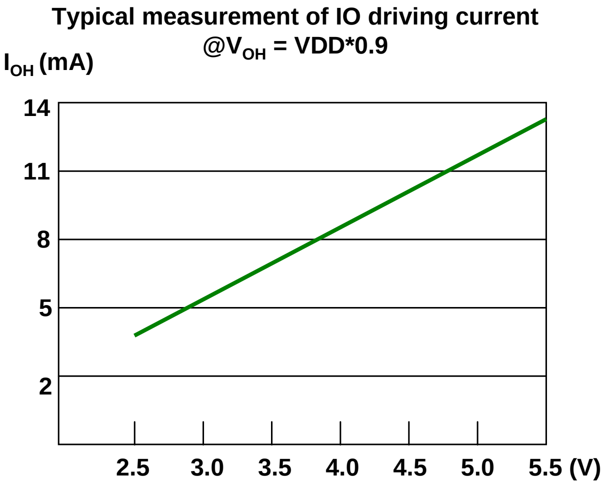


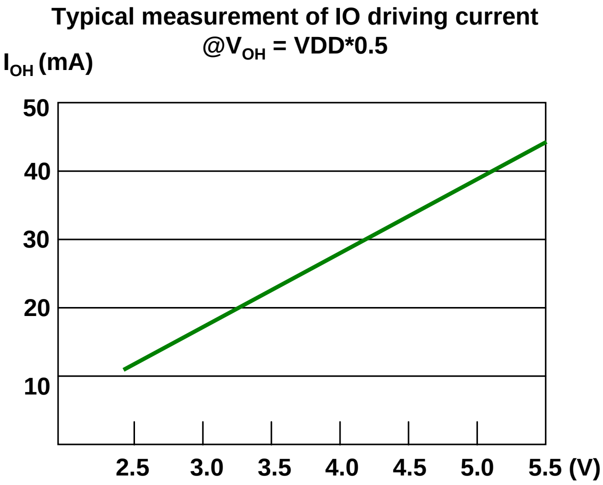


Preliminary Information, subject to change without notice

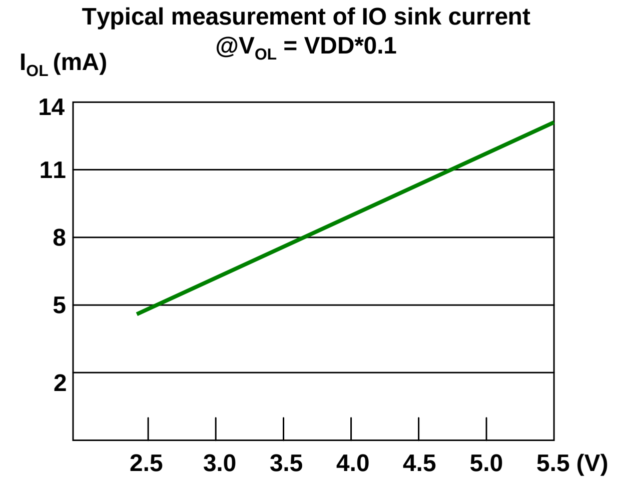


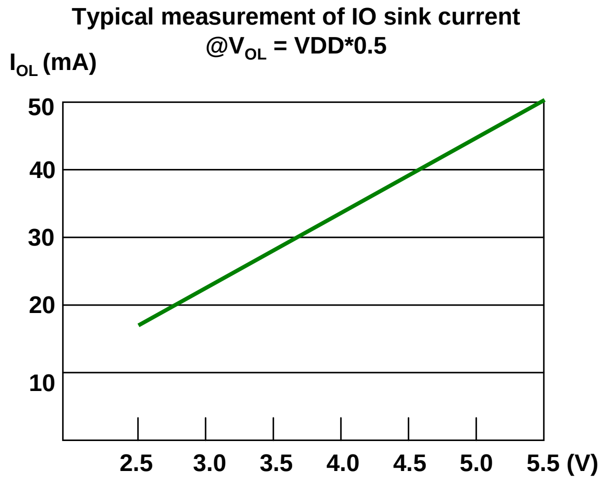
## 4-26. Typical measurement of IO driving/sink current



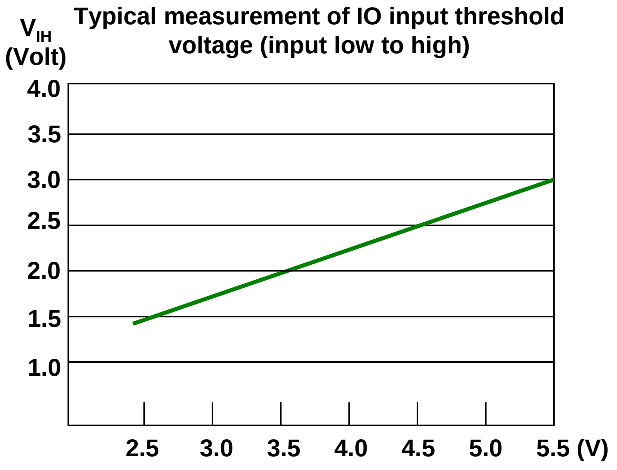


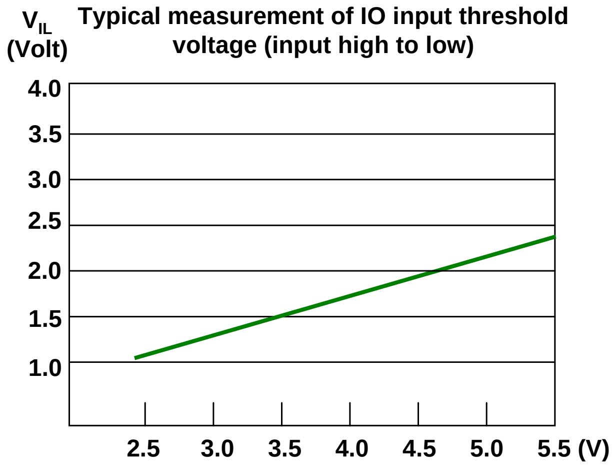
Preliminary Information, subject to change without notice





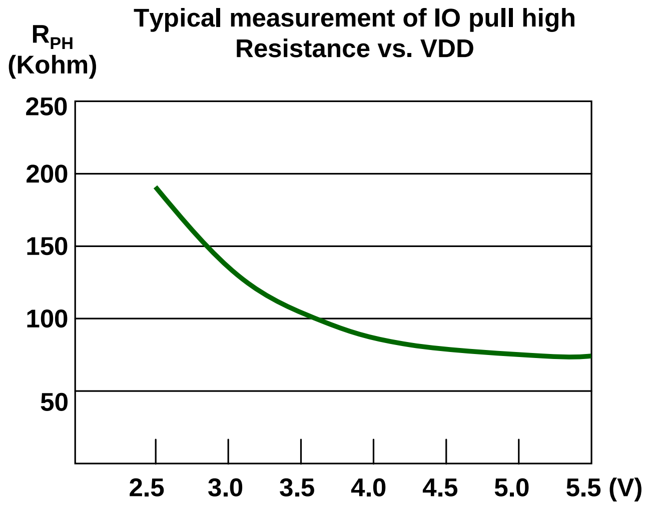
## 4-27. Typical measurement of IO input threshold voltage



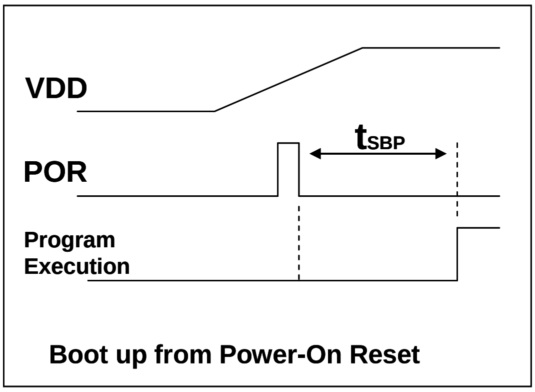


Preliminary Information, subject to change without notice

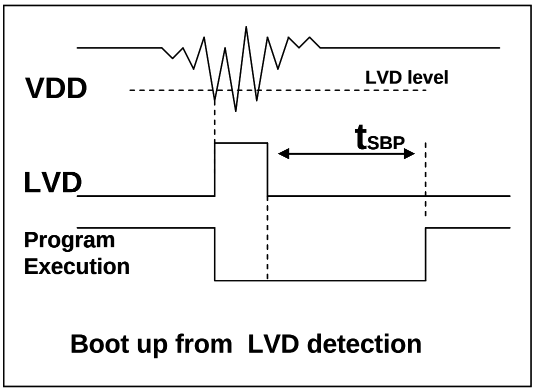
## 4-28. Typical measurement of IO pull high resistance



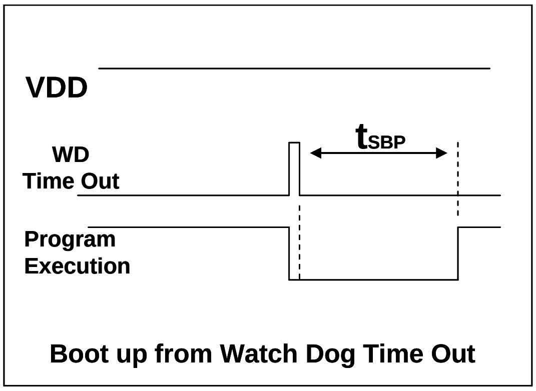
## 4-29. Timing charts for boot up conditions



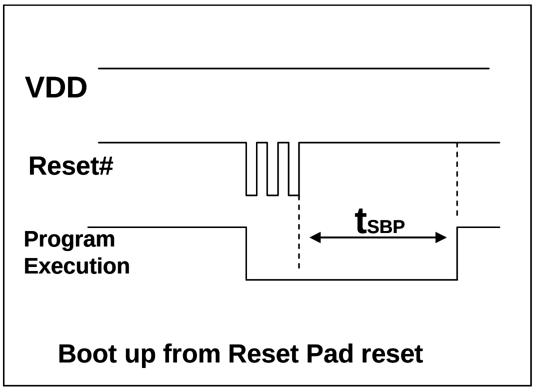
Boot up from Power-On Reset



Boot up from LVD detection



Boot up from Watch Dog Time Out

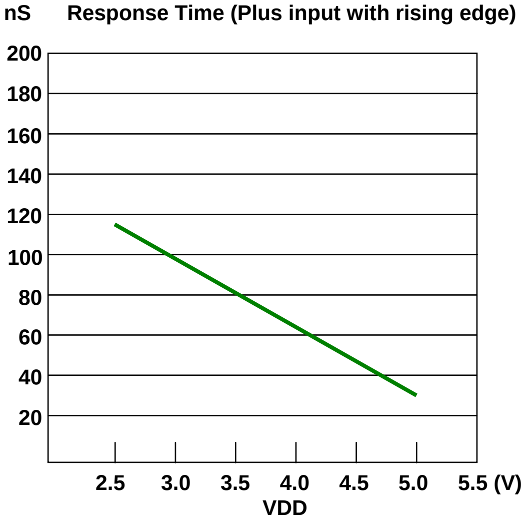


Boot up from Reset Pad reset

## 4-30. Typical measurement of Comparator Responsive Time (Use $V_{internal\ R}$)

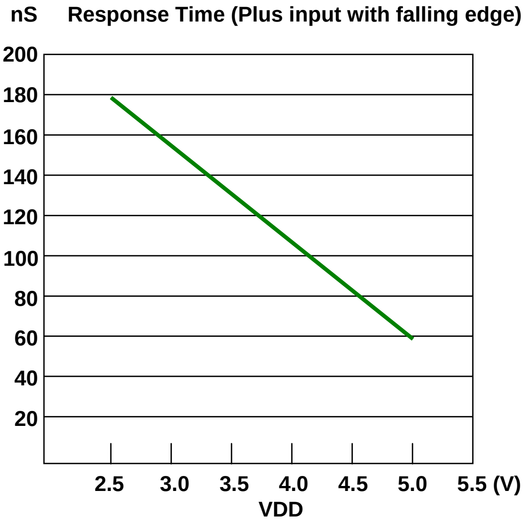
**Measurement Conditions:**

- Plus input: PC5
- Minus input: $V_{internal\ R}$ = (5/16)*VDD
- $V_{internal\ R}$: use Case I, R1
- Fix $V_{internal\ R}$, use PC5 rising edge to measure response time



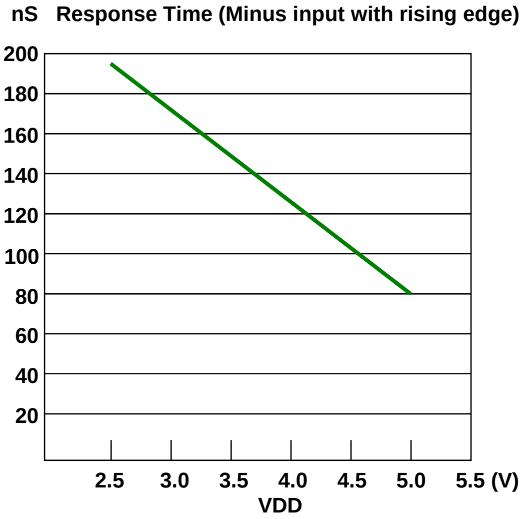
**Measurement Conditions:**

- Plus input: PC5
- Minus input: $V_{internal\ R}$ = (5/16)*VDD
- $V_{internal\ R}$: use Case I, R1
- Fix $V_{internal\ R}$, use PC5 falling edge to measure response time



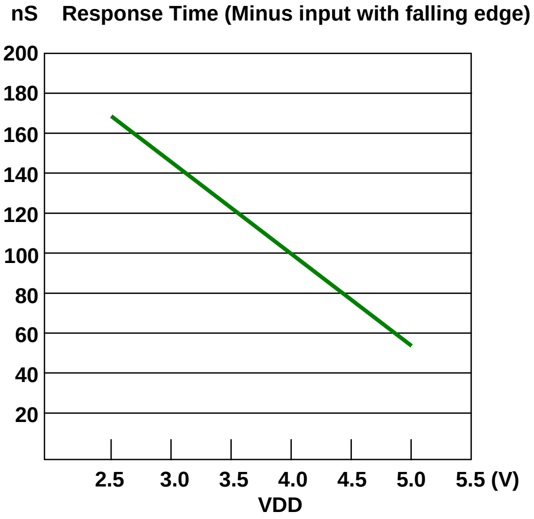
**Measurement Conditions:**

- Plus input: $V_{internal\ R}$ = (5/16)*VDD
- Minus input: PC5
- $V_{internal\ R}$: use Case I, R1
- Fix $V_{internal\ R}$, use PC5 rising edge to measure response time



**Measurement Conditions:**

- Plus input: $V_{internal\ R}$ = (5/16)*VDD
- Minus input: PC5
- $V_{internal\ R}$: use Case I, R1
- Fix $V_{internal\ R}$, use PC5 falling edge to measure response time

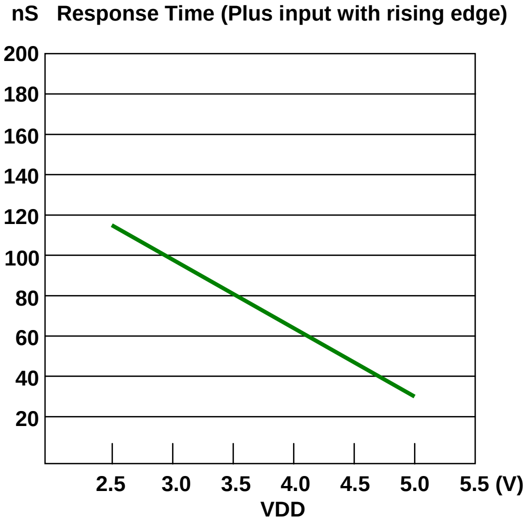


Preliminary Information, subject to change without notice

## 4-31. Typical measurement of Comparator Responsive Time

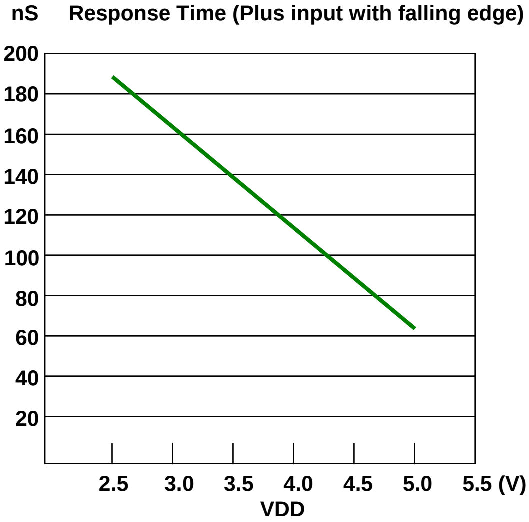
Measurement Conditions:

- Plus input: PC5
- Minus input: PC0 = (5/16)VDD
- Fix PC0, use PC5 rising edge to measure response time



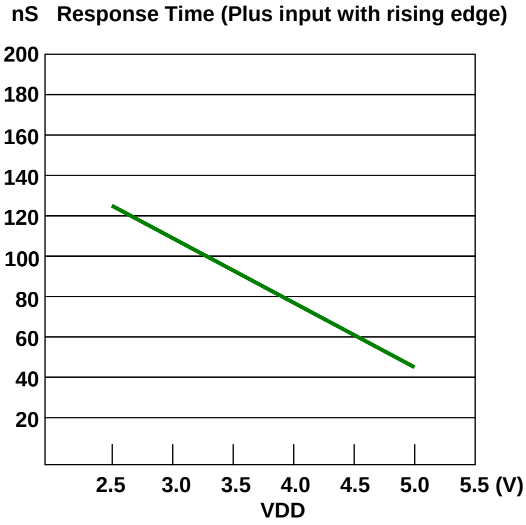
Measurement Conditions:

- Plus input: PC5
- Minus input: PC0 = (5/16)VDD
- Fix PC0, use PC5 falling edge to measure response time



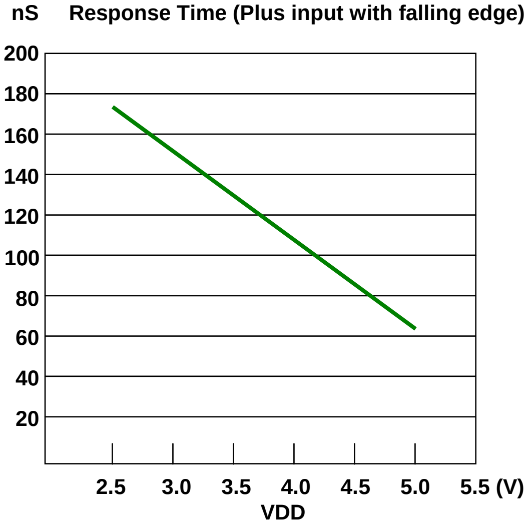
Measurement Conditions:

- Plus input: PC5 = (5/16)VDD
- Minus input: PC0
- Fix PC5, use PC0 rising edge to measure response time



Measurement Conditions:

- Plus input: PC5 = (5/16)VDD
- Minus input: PC0
- Fix PC5, use PC0 falling edge to measure response time



# 5. Functional Description

## 5-1. Processing Units

There are two processing units (FPP unit) inside the chip of P232. In each processing unit, it includes (i) its own Program Counter to control the program execution sequence (ii) its own Stack Pointer to store or restore the program counter for program execution (iii) its own accumulator (iv) Status Flag to record the status of program execution. Each FPP unit has its own program counter and accumulator for program execution, flag register to record the status, and stack pointer for jump operation. Based on such architecture, FPP unit can execute its own program independently, thus parallel processing can be expected.

These two FPP units share the same 2Kx16 bits OTP program memory, 200 bytes data SRAM and all the IO ports, these two FPP units are operated at mutual exclusive clock cycles to avoid interference. One task switch is built inside the chip to decide which FPP unit should be active for the corresponding cycle. The hardware diagram and basic timing diagram of FPP units are illustrated in Fig. 1. For FPP0 unit, its program will be executed in sequence every other system clock, shown as $(M-1)_{th}$, $M_{th}$ and $(M+1)_{th}$ instructions. For FPP1 unit, its program will be also executed in sequence every other system clock, shown as $(N-1)_{th}$, $N_{th}$ and $(N+1)_{th}$ instructions.

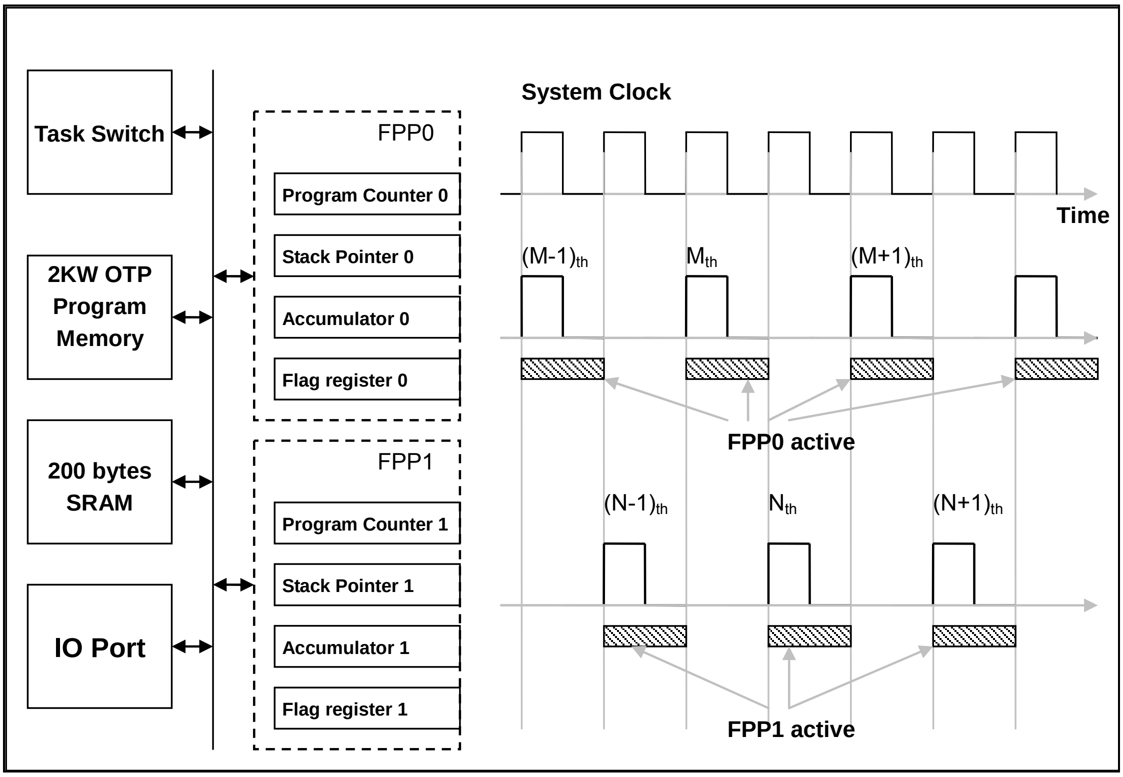


Fig. 1 Hardware and Timing Diagram of FPP unit

Each FPP unit has half computing power of whole system; for example, FPP0 and FPP1will be operated at 8MHz if system clock is 16MHz. The FPP unit can be enabled or disabled by programming the FPP unit Enable Register, only FPP0 is enabled after power-on reset. The system initialization will be started from FPP0 and FPP1 unit can be enabled by user's program if necessary. Both FPP0 and FPP1 can be enabled or disabled by using any one FPP unit.

### 5-1-1. Program Counter

Program Counter (PC) is the unit that contains the address of an instruction to be executed next. The program counter is automatically incremented at each instruction cycle so that instructions are retrieved sequentially from the program memory. Certain instructions, such as branches and subroutine calls, interrupt the sequence by placing a new value in the program counter. The bit length of the program counter is 11 for P232. The program counter of FPP0 is 0 after hardware reset and 1 for FPP1. Whenever an interrupt happens in FPP0, the program counter will jump to 'h10 for interrupt service routine. Each FPP unit has its own program counter to control the program execution sequence.

### 5-1-2. Stack Pointer

The stack pointer in each processing unit is used to point the top of the stack area where the local variables and parameters to subroutines are stored; the stack pointer register (sp) is located in IO address 0x02h. The bit number of stack pointer is 8 bit; the stack memory cannot be accessed over 200 bytes and should be defined within 200 bytes from 0x00h address. The stack memory of P232 for each FPP unit can be assigned by user via stack pointer register, means that the depth of stack pointer for each FPP unit is adjustable in order to optimize system performance. The following example shows how to define the stack in the ASM (assembly language) project:

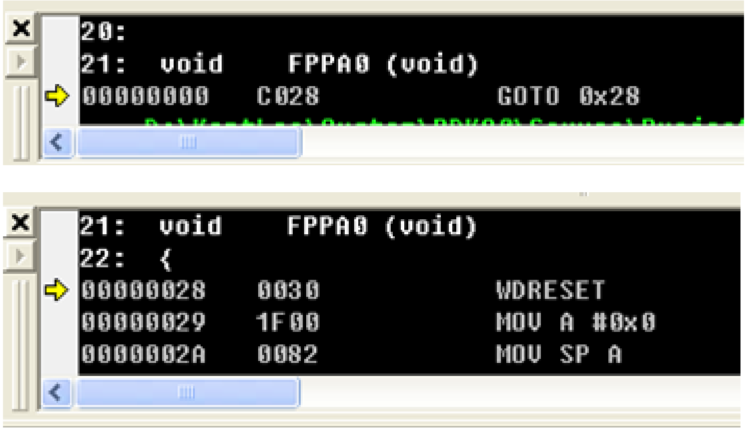
```
     . ROMADR 0
     GOTO          FPPA0
     GOTO          FPPA1
     ...
     . RAMADR 0                               // Address must be less than 0x100
     WORD          Stack0  [1]                // one WORD
     WORD          Stack1  [2]                // two WORD
     ...
FPPA0:
     SP   =        Stack0;                    // assign Stack0 for FPPA0,
                                              // one level call because of Stack0[1]
     ...
     call          function1
     ...
FPPA1:
     SP   =        Stack1;                    // assign Stack1 for FPPA1,
                                              // two level call because of Stack1[2]
     ...
     call          function2
     ...
```

Preliminary Information, subject to change without notice

In Mini-C project, the stack calculation is done by system software, user will not have effort on it, and the example is shown as below:

```
void        FPPA0  (void)
{
...
}
```

User can check the stack assignment in the window of program disassembling, Fig. 2 shows that the status of stack before FPP0 execution, system has calculated the required stack space and has reserved for the program.



```
20:
21:  void     FPPA0 (void)
00000000    C028             GOTO  0x28
```

```
21:  void     FPPA0 (void)
22:  {
00000028    0030             WDRESET
00000029    1F00             MOV  A  #0x0
0000002A    0082             MOV  SP  A
```

Fig. 2 Stack Assignment in Mini-C project

## 5-2. Program Memory – OTP

The OTP (One Time Programmable) program memory is used to store the program instructions to be executed. All program codes for each FPP unit are stored in this OTP memory for both FPP0 and FPP1. The OTP program memory may contains the data, tables and interrupt entry. After reset, the initial address for FPP0 is ‘h000 and ‘h001 for FPP1, the interrupt entry is ‘h010 and interrupt is for FPP0 only, the last eight addresses are reserved for system using, like checksum, serial number, etc. The OTP program memory for P232 is 2Kx16 bits that is partitioned as below.

| Address | Function |
|---:|---|
| 000 | FPP0 reset – goto instruction |
| 001 | FPP1 reset – goto instruction |
| 002 | Reserved |
| • | • |
| 00F | Reserved |
| 010 | Interrupt entry address |
| 011 | User program memory |
| • | • |
| • | • |
| 7F7 | User program memory |
| 7F8 | System Using |
| • | • |
| 7FF | System Using |

Table 1: Program Memory Organization

In order to have maximum flexibility for user program using, the user program memory is shared with both FPP0 and FPP1, and the program space allocation is done by program compiler automatically, user does not need to specify the address if not necessary. Table 2 shows one example of program memory using.

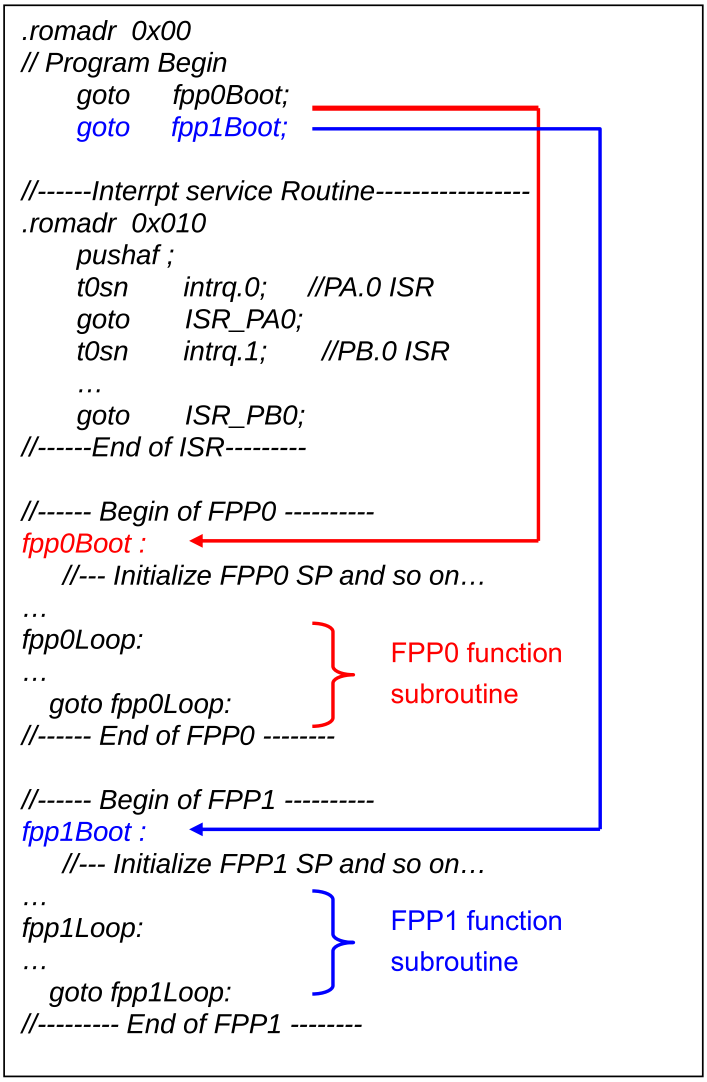
| Address | Function |
|---:|---|
| 000 | FPP0 reset – goto instruction (goto ‘h011) |
| 001 | FPP1 reset – goto instruction (goto ‘h3A1) |
| 002 | Reserved |
| • | • |
| 00F | Reserved |
| 010 | Interrupt entry address |
| 011 | Begin of FPP0 user program |
| • | • |
| • | • |
| 3A0 | End of FPP0 user program |
| 3A1 | Begin of FPP1 program |
| • | • |
| • | • |
| 537 | End of FPP1 program |
| 538 | Not used |
| • | • |
| • | • |
| 7F7 | Not used |
| 7F8 | System Using |
| • | • |
| 7FF | System Using |

Table 2: Example of Program Memory Using

Preliminary Information, subject to change without notice

### 5-3. Program Structure

After power-up, the program starting address of FPP0 is 0x000 and 0x001 for FPP1. The 0x010 is the entry address of interrupt service routine, which belongs to FPP0 only. The basic firmware structure for P232 is shown as Fig. 3, the program codes of two FPP units are placed in one whole program space. Except for the initial addresses of processing units and entry address of interrupt, the memory location is not specially specified; the program codes of processing unit can be resided at any location no matter what the processing unit is. After power-up, the fpp0Boot will be executed first, which will include the system initialization and other FPP units enabled.



```
.romadr  0x00
// Program Begin
       goto     fpp0Boot;
       goto     fpp1Boot;

//------Interrpt service Routine-----------------
.romadr  0x010
       pushaf ;
       t0sn      intrq.0;     //PA.0 ISR
       goto      ISR_PA0;
       t0sn      intrq.1;      //PB.0 ISR
       …
       goto      ISR_PB0;
//------End of ISR---------

//------ Begin of FPP0 ----------
fpp0Boot :
     //--- Initialize FPP0 SP and so on…
…
fpp0Loop:
…
   goto fpp0Loop:
//------ End of FPP0 --------

//------ Begin of FPP1 ----------
fpp1Boot :
     //--- Initialize FPP1 SP and so on…
…
fpp1Loop:
…
   goto fpp1Loop:
//--------- End of FPP1 --------
```

Fig. 3 Program Structure

## 5-4. Boot Procedure – Fast Boot up and Normal Boot up

By code option, user can choose fast boot up or normal boot up when compiling the program. For 16MHz IHRC and 32KHz ILRC, it will take about 20us for fast boot up (128 IHRC clock cycles plus stable time of hardware circuit) and 32ms for normal boot up (1024 ILRC clock cycles plus stable time of hardware circuit), that is $t_{SBP}$ in the Fig. 4. This selected code option will be automatically written into bit 3 of clkmd register and to select the default system clock after boot up. After boot up procedure, the default system clock is IHRC/6 for fast boot up and ILRC for normal boot up. No matter whether fast boot up is chosen or not, user should ensure that the power should be stable after boot up time.

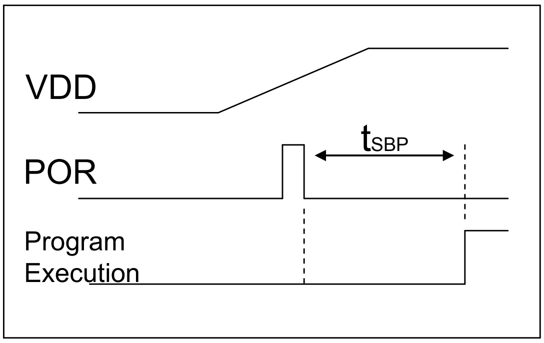


**Fig 4: Power-On Sequence**

Fig. 5 shows the typical program flow after boot up. Please notice that the FPP1 is disabled after reset and recommend NOT enabling FPP1 before system and FPP0 initialization.

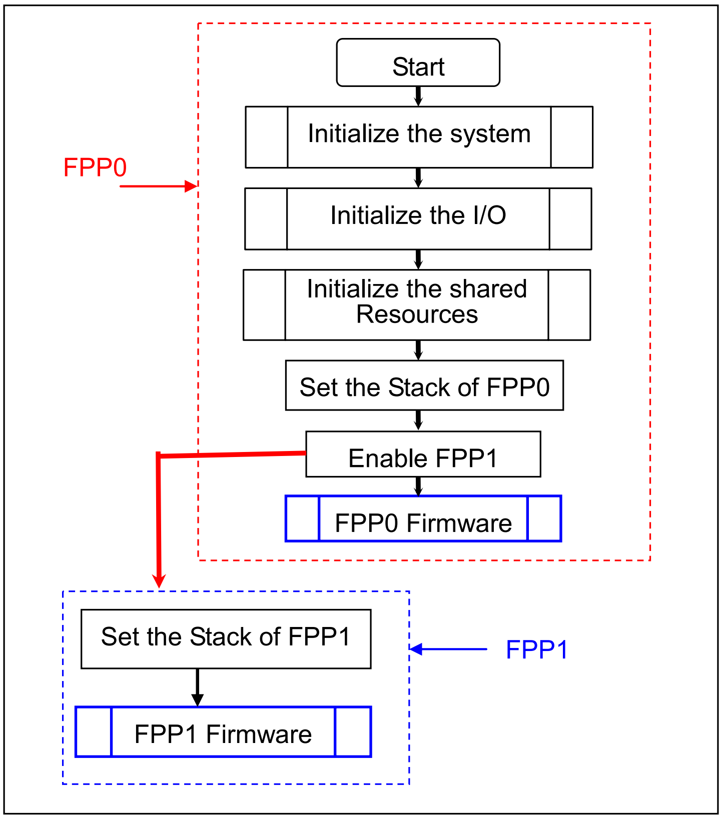


Fig. 5 Boot Procedure

## 5-5. Data Memory -- SRAM

Fig. 6 shows the SRAM data memory organization of P232, all the SRAM data memory could be accessed by FPP0 and FPP1 directly with 1T clock cycle, the data access 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 for all FPP units.

The stack memory for each processing unit should be independent from each other, and defined in the data memory. The stack pointer is defined in the stack pointer register of each processing unit; 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, the memory size of indirect memory access is only 256 bytes only, all the 200 bytes data memory of P232 can be accessed by indirect access mechanism.

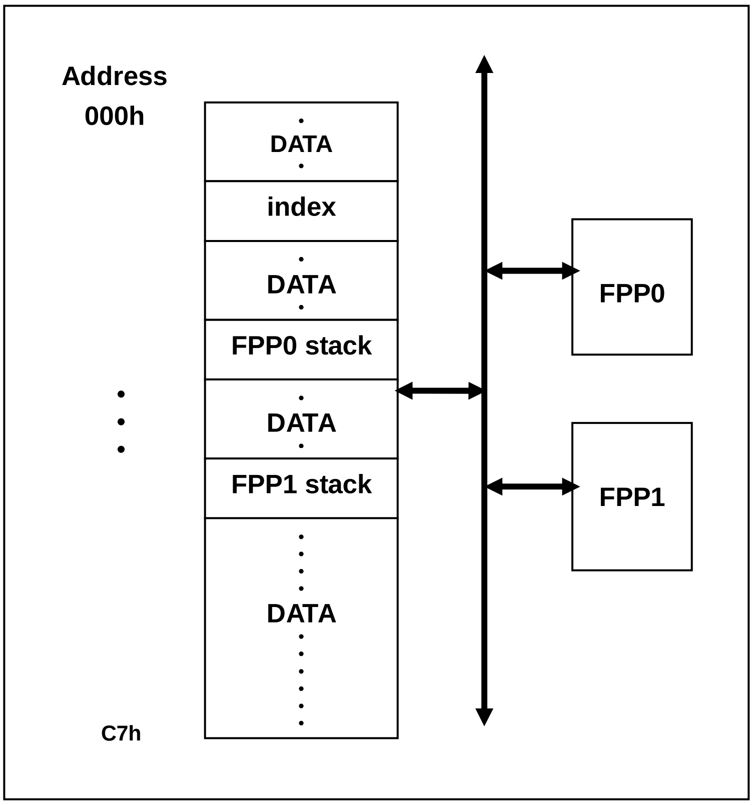


Fig. 6 Data Memory Organization

## 5-6. Arithmetic and Logic Unit

Arithmetic and Logic Unit (ALU) is the computation element to operate integer arithmetic, logic, shift and other specialized operations. The operation data can be from instruction, accumulator or SRAM data memory. Computation result could be written into accumulator or SRAM. FPP0 and FPP1 will share ALU for its corresponding operation.

## 5-7. Oscillator and clock

There are three hardware oscillator circuits provided by P232: external crystal oscillator, internal high RC oscillator (IHRC) and internal low RC oscillator (ILRC). Those above oscillators can be enabled and chosen by setting the corresponding bits in ***clkmd*** register to generate different system clock to meet different application.

### 5-7-1. Internal High RC oscillator and Internal Low RC oscillator

The IHRC is enabled after power-up and can be disabled via bit 4 of ***clkmd*** register, the frequency of IHRC can be calibrated by ***ihrcr*** register; usually it is calibrated to 16MHz to eliminate the variation in fabrication process; the frequency deviation after calibration can be within 1% normally. The IHRC calibration can be enabled when compiling the user's program and chip is calibrated one by one during programming the OTP codes into the chip.

The frequency of IHRC will drift with supply voltage and operating temperature, the major factor is the temperature variation. Table 3 shows the frequency deviation of IHRC under different conditions.

| | Deviation | Conditions |
|---|---|---|
| IHRC is calibrated to 16MHz | ±1% | VDD=5V, 25$^{o}$C |
| | ±3% | 2.5V<VDD<5.5V, 25$^{o}$C |
| | ±8% | 2.5V<VDD<5.5V, 0$^{o}$C<Ta<70$^{o}$C |
| | ±12% | 2.5V<VDD<5.5V, -40$^{o}$C<Ta<85$^{o}$C |

Table 3: Frequency Deviation of IHRC

The ILRC is also enabled after power-up and can be disabled via bit 2 of ***clkmd*** register, the frequency of ILRC is fixed at 32 KHz, however, it is also varied by process, supply voltage and temperature, please refer to DC specification. User should pay attention to frequency variation when using ILRC.

### 5-7-2. IHRC Frequency Adjustment (.ADJUST_OTP_IHRCR)

In IDE utility, the options for IHRC calibration will be selected by user during compilation and the corresponding script command will be inserted into the user's program automatically, the options for selection is shown as Table 4:

| | CLKMD | IHRCR | Comments |
|---|---|---|---|
| ○ Set 8MHz : | = 34h (IHRC / 2) | Calibrate | IHRC is calibrated to 16MHz, CLK=8MHz (IHRC/2) |
| ○ Set 4MHz : | = 14h (IHRC / 4) | Calibrate | IHRC is calibrated to 16MHz, CLK=4MHz (IHRC/4) |
| ○ Set 2MHz : | = 3Ch (IHRC / 8) | Calibrate | IHRC is calibrated to 16MHz, CLK=2MHz (IHRC/8) |
| ○ Set 1MHz : | = 1Ch (IHRC / 16) | Calibrate | IHRC is calibrated to 16MHz, CLK=1MHz (IHRC/16) |
| ○ Set ILRC : | = E4h (ILRC / 1) | Calibrate | IHRC is calibrated to 16MHz, CLK=ILRC |
| ○ Disable | No change | No Change | IHRC is NOT calibrated, CLK is not changed |

Table 4 Options for IHRC calibration

Usually, .ADJUST_OTP_IHRCR script command is put at the beginning of program to set the desired operating frequency after boot up. The IHRC calibration is done once when programming the OTP code and will NOT be executed again after then. If user chooses different option, the state of P234 will be different after power up; it shows the difference after executing this command:

(1) .ADJUST_OTP_IHRCR 8MIPS

CLKMD = 0x34 after executing command for both fast boot up and normal boot up:

- IHRC is calibrated to 16MHz and enabled
- System Clock = IHRC/2 = 8MHz
- Watch Dog is disabled, ILRC is enabled, and PA5 acts as input pin.

(2) .ADJUST_OTP_IHRCR 4MIPS

CLKMD = 0x14 after executing command for both fast boot up and normal boot up:

- IHRC is calibrated to 16MHz and enabled
- System Clock = IHRC/4 = 4MHz
- Watch Dog is disabled, ILRC is enabled, and PA5 acts as input pin.

(3) .ADJUST_OTP_IHRCR 2MIPS

CLKMD = 0x3C after executing command:

- IHRC is calibrated to 16MHz and enabled
- System Clock = IHRC/8 = 2MHz
- Watch Dog is disabled, ILRC is enabled, and PA5 acts as input pin

(4) .ADJUST_OTP_IHRCR 1MIPS

CLKMD = 0x1C after executing command:

- IHRC is calibrated to 16MHz and enabled
- System Clock = IHRC/16 = 1MHz
- Watch Dog is disabled, ILRC is enabled, and PA5 acts as input pin

**(5)** .ADJUST_OTP_IHRCR ILRC

CLKMD = 0XE4 after executing command for both fast boot up and normal boot up:

- IHRC is calibrated to 16MHz and disabled
- System Clock = ILRC
- Watch Dog is disabled, ILRC is enabled, PA5 acts as input pin

**(6)** .ADJUST_OTP_IHRCR DISABLE

CLKMD = 0XF6 after executing command for normal boot up:

- IHRC is NOT calibrated to 16MHz and enabled
- System Clock = ILRC
- Watch Dog is enabled, ILRC is enabled, and PA5 acts as input pin

CLKMD = 0x5E after executing command for fast boot up:

- IHRC is NOT calibrated to 16MHz and enabled
- System Clock = IHRC/6
- Watch Dog is enabled, ILRC is enabled, and PA5 acts as input pin

Although the IHRC frequency is usually calibrated to 16MHz, however, IDE utility provides the script command to calibrate IHRC to other frequency, the adjustable frequency of IHRC is around 15MHz to 17MHz. The script format for IHRC adjustment is:

***.ADJUST_OTP_IHRCR AAA***

Where AAA is the target frequency (in Hz) of system clock and the frequency of system clock will be kept within 937500~1062500Hz, 1875000~2125000Hz, 3750000~4250000Hz or 7500000~8500000Hz. The following cases show how to use the script command to have system clock for UART and IR applications.

**Case 1:** The Baud rate for UART transmission is 115200, the possible methods are:

```
.ADJUST_OTP_IHRCR   7833600
    // IHRC=15667200Hz, CLK=7833600Hz =68 * 115200, each FPP run at 3916800Hz
.ADJUST_OTP_IHRCR   3916800
    // IHRC=15667200Hz, CLK=3916800Hz =34 * 115200, each FPP run at 1958400Hz
```

**Case 2:** Using IHRC and Timer2 to generate 455KHz clock for IR application:

```
.ADJUST_OTP_IHRCR   4095000      //  IHRC=16.38MHz, CLK=4.095MHz
tm2c = 0b0010_00_0_0;            //  Use IHRC, disable output, period mode, no inverse
tm2s = 0b0_00_10001;             //  Timer2 clock: IHRC/18 = 16.38MHz / 18 = 910KHz
tm2ct = 0;                       //  counter register is reset to zero
tm2b = 0xff;                     //  upper bound of Timer2
clkmd = 0b110_1_1_0_0_0;         //  switch system clock = TM2_CLK &
                                 //  each FPP unit run at 910KHz/2 = 455KHz
```

Preliminary Information, subject to change without notice

More technical information about IHRC adjustment, please refer to IDE utility “Help → Application Note → IC Introduction → Instruction Introduction → ADJUST_OTP_IHRCR”. Please notice that only PDK3S-P-001 1 to 1 OTP Writer can support the function of calibrating IHRC into non-standard frequency (≠16MHz), it can NOT be done in silicon factory.

### 5-7-3. Crystal Oscillator

If crystal oscillator is used, a crystal or resonator is required between X1 and X2. Fig. 7 shows the hardware connection for using crystal oscillator; EOSC is the clock from crystal oscillator and its operating frequency can be ranged from 32 KHz to 16MHz, depending on the crystal placed on. Besides crystal, external capacitors C1 and C2 should be added, its values should depend on the specification of selected crystal or resonator; the bit [7-5] in ***eoscr*** register (0x0b) should be set to have good sinusoidal waveform for different frequency.

The pull high resistors for PA6 and PA7 must be disabled via bit [7-6] of ***paph*** register when crystal oscillator is used. The crystal oscillator is NOT used for the clock source of Timer2 because of possible clock glitch; however, it CAN be used for the clock source of Timer16.

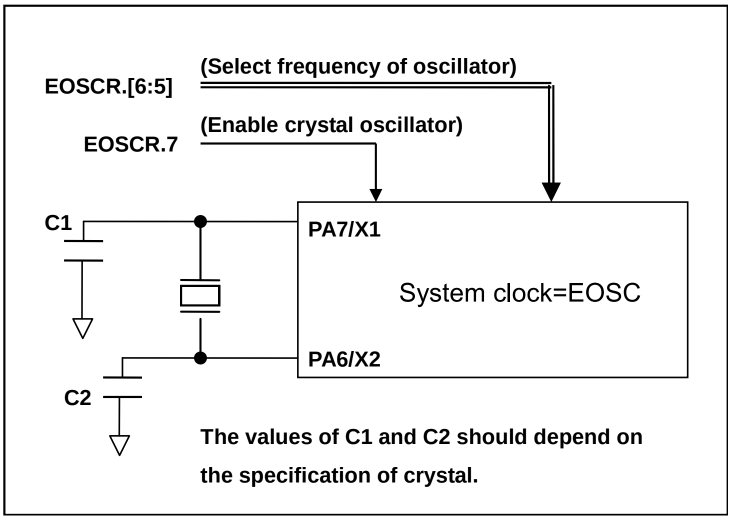

EOSCR.[6:5]
(Select frequency of oscillator)
EOSCR.7
(Enable crystal oscillator)
C1
PA7/X1
System clock=EOSC
PA6/X2
C2
The values of C1 and C2 should depend on the specification of crystal.

Fig. 7 Connection of crystal oscillator

To optimize power consumption and start-up time of crystal oscillator, the bit 6 of ***misc*** register provides option to increase driving capability for the crystal oscillator. In the beginning of crystal start-up, the bit 6 of ***misc*** register can be enabled to provide higher driving capability to speed up the oscillation; After the oscillation is stable, this bit can be disabled for the purpose of power saving.

When using the crystal oscillator, user should make sure the oscillator is already stable before switching to it. After power-up or enabling, the stable time of oscillator is different for different frequency. The example program for using the crystal oscillator is shown as below:

```
void FPPA0 (void)
{
    .ADJUST_OTP_IHRCR   DISABLE           //    No adjust IHRCR, WatchDog Enable
    . . .
    $     MISC      EC_High_Drive;        //    Enable high drive for crystal osc.
    PAC       =     0B_00_xxxxxx;         //    PA6, PA7 input mode
    PAPH      =     0B_00_xxxxxx;         //    Disable PA6, PA7 pull high resistor
    PADIDR    =     0B_11_xxxxxx;         //    Set PA6, PA7 as analog input
    $     EOSCR     Enable, 4Mhz;         //    EOSCR = 0b110_00000;
    $     T16M      EOSC, /1, BIT13;      //    When bit13 of Timer16:   0 -> 1
                                          //    Intrq.T16 =>1 to make sure crystal EOSC is stable
    WORD      count     =     0;
    stt16     count;
    Intrq . T16 =   0;
    wait1     Intrq.T16;                  //    count fm 0x0000 to 0x2000, set INTRQ.T16
    clkmd     =     0xA4;                 //    switch to EOSC;
    . . .
}
```

### 5-7-4. System Clock and LVD level

The system clock of P232 can come from EOSC, IHRC, ILRC or TM2_CLK (please refer to the hardware diagram of Timer2), the hardware diagram for system clock in the P232 is shown as Fig. 8. TM2_CLK is the clock output after pre-scalar and scalar hardware circuit of Timer2; P232 can provide the wide range system clock via ***clkmd*** register selection.

After power up, the running system clock will be IHRC/6 if fast boot up is selected and ILRC/1 for normal boot up. User can change the system clock any time by setting ***clkmd*** register and the new system clock will be changed into the new one immediately after writing the ***clkmd*** register.

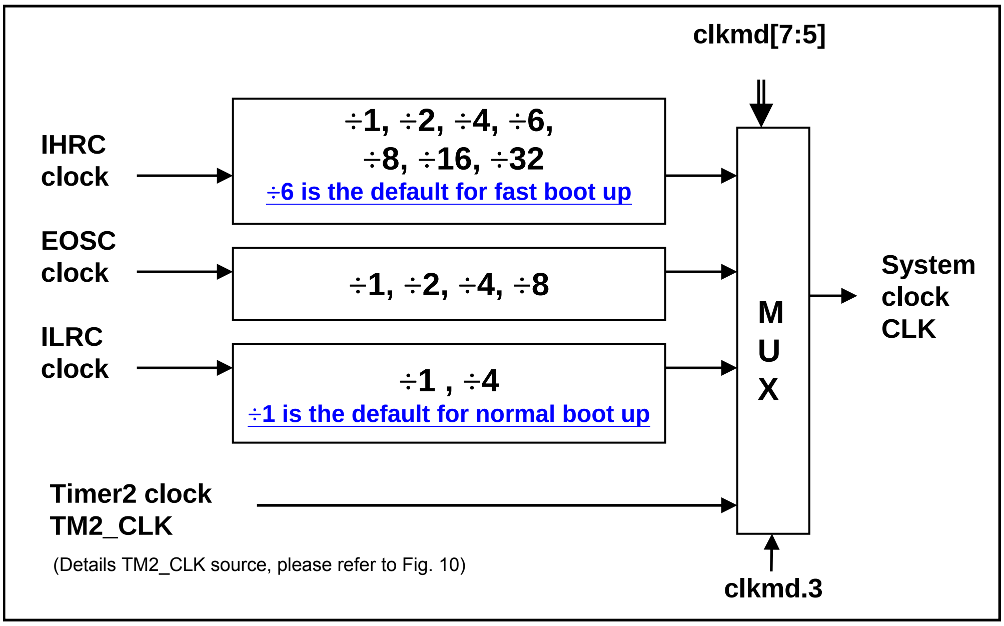


Fig. 8 Options of System Clock

User can choose different operating system clock depends on its requirement; the selected operating system clock should be in conjunction with supply voltage and LVD level to ensure system stable. The LVD level can be selected during compilation, the following operating frequency and LVD level is recommended:

- system clock = 8MHz with LVD=4.0V
- system clock = 4MHz with LVD=3.0V
- system clock = 2MHz with LVD=2.5V
- system clock = 1MHz with LVD=2.2V

No matter IHRC or external crystal oscillator, user had better choose system clock as half of oscillator clock (divided-by-2) to have system clock with 50% duty cycle. Although P232 can run 16MHz at 5.0 volt, however, it does not recommend using 16MHz clock as system clock because of the possible variation of high/low duty.

### 5-7-5. System Clock Switching

After IHRC calibration, user may want to switch system clock to a new frequency or may switch system clock at any time to optimize the system performance and power consumption. Basically, the system clock of P232 can be switched among IHRC, ILRC and EOSC by setting the ***clkmd*** register at any time; system clock will be the new one after writing to ***clkmd*** register immediately. Please notice that the original clock module can NOT be turned off at the same time as writing command to ***clkmd*** register. The examples are shown as below and more information about clock switching, please refer to the “Help → Application Note → IC Introduction → Register Introduction → CLKMD”.

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

```
...                                 //  system clock is ILRC
CLKMD       =  0x34 ；              //  switch to IHRC/2 ，ILRC CAN NOT be disabled here
CLKMD .2    =  0 ；                 //  ILRC CAN be disabled at this time
...
```

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

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

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

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

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

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

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

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

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

```
...                                 //  system clock is ILRC
CLKMD       =  0x30 ；              //  CAN NOT switch clock from ILRC to IHRC/2 and

                                    //  turn off ILRC oscillator at the same time\
```

## 5-8. 16-bit Timer (Timer16)

A 16-bit hardware timer (Timer16) is implemented in the P232, 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) 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 5 of ***integs*** register (IO address 0x0C).

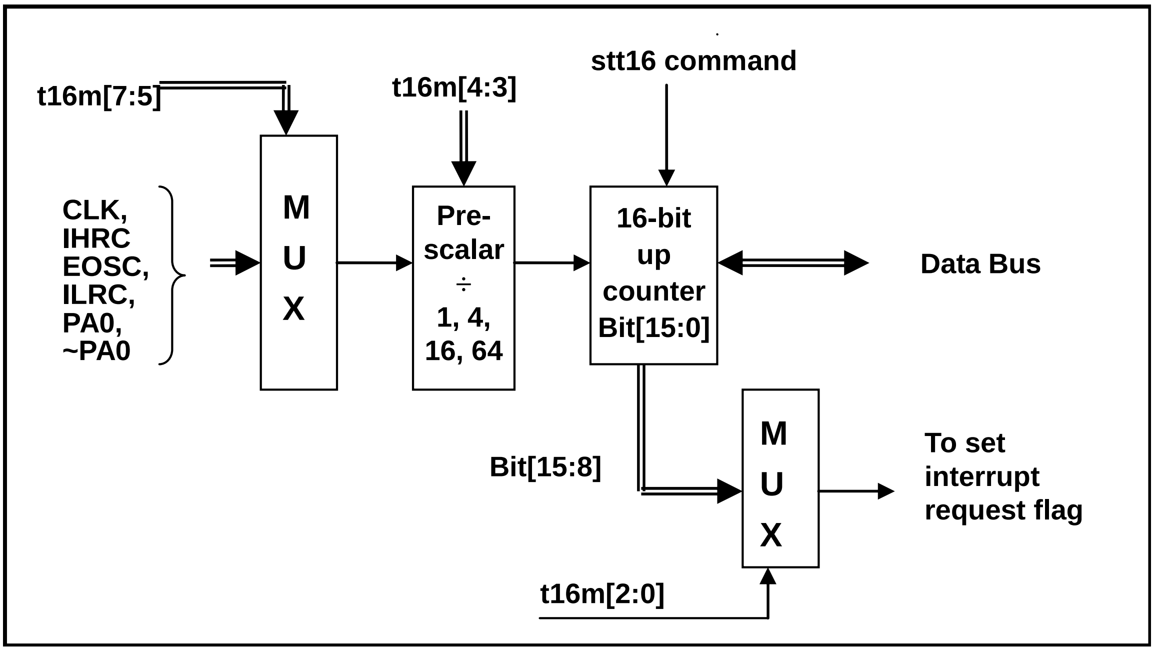


Fig. 9 Hardware diagram of Timer16

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, PA0_R, 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 can refer to “Help → 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 .ADJUST_OTP_IHRCR 8MIPS, 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 nth bit selected to request interrupt service, for example: n=10 if bit 10 is selected.

### 5-9. 8-bit Timer (Timer2) with PWM generation

An 8-bit hardware timer (Timer2) with PWM generation is implemented in the P232, please refer to Fig. 10 shown its hardware diagram, the clock sources of Timer2 may come from system clock, internal high RC oscillator (IHRC), internal low RC oscillator (ILRC), PA0, PA3, PA4 and comparator output, bit [7:4] of register tm2c are used to select the clock of Timer2. Please notice that external crystal oscillator is NOT to be the clock of Timer2 because of possible clock glitch. If IHRC is selected for Timer2 clock source, the clock sent to Timer2 will keep running when using ICE in halt state. The output of Timer2 can be sent to pin PA2 or PA3, depending on bit [3-2] of tm2c register. 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. TM2_CLK can be selected as system clock in order to provide special frequency of system clock, please refer to ***clkmd*** register.

The Timer2 counter performs 8-bit up-counting operation only; the counter values can be set or read back by tm2ct register. The 8-bit counter will be clear to zero automatically when its values reach for upper bound register, the upper bound register is used to define the period of timer or duty of PWM. There are two operating modes for Timer2: period mode and PWM mode; period mode is used to generate periodical output waveform or interrupt event; PWM mode is used to generate PWM output waveform with optional 6-bit or 8-bit PWM resolution, Fig. 11 shows the timing diagram of Timer2 for both period mode and PWM mode.

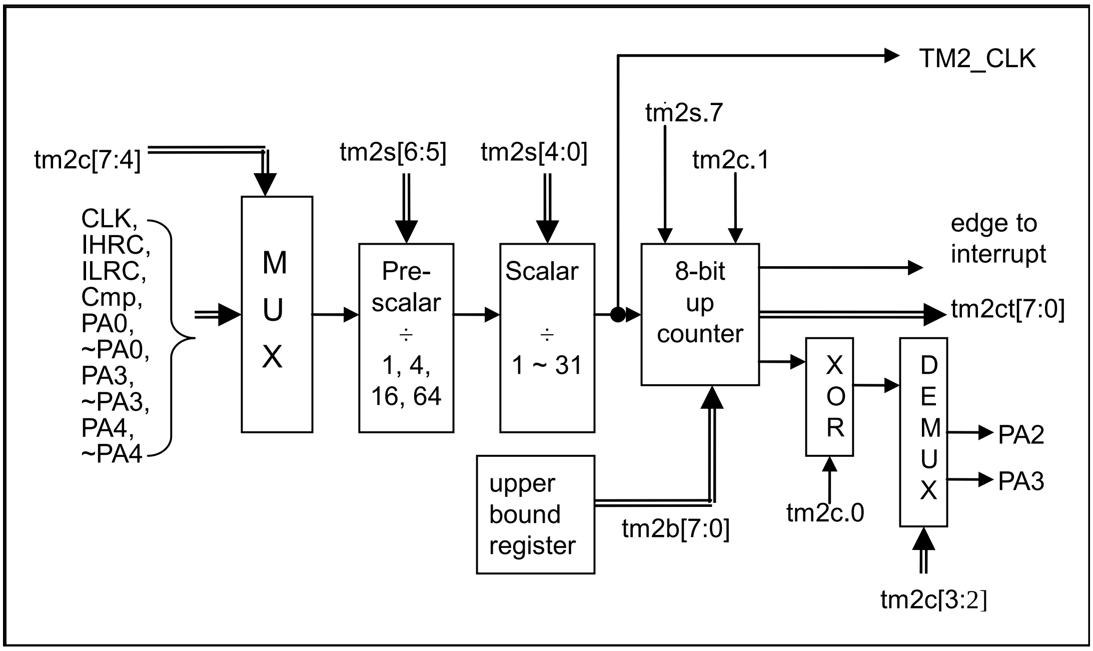
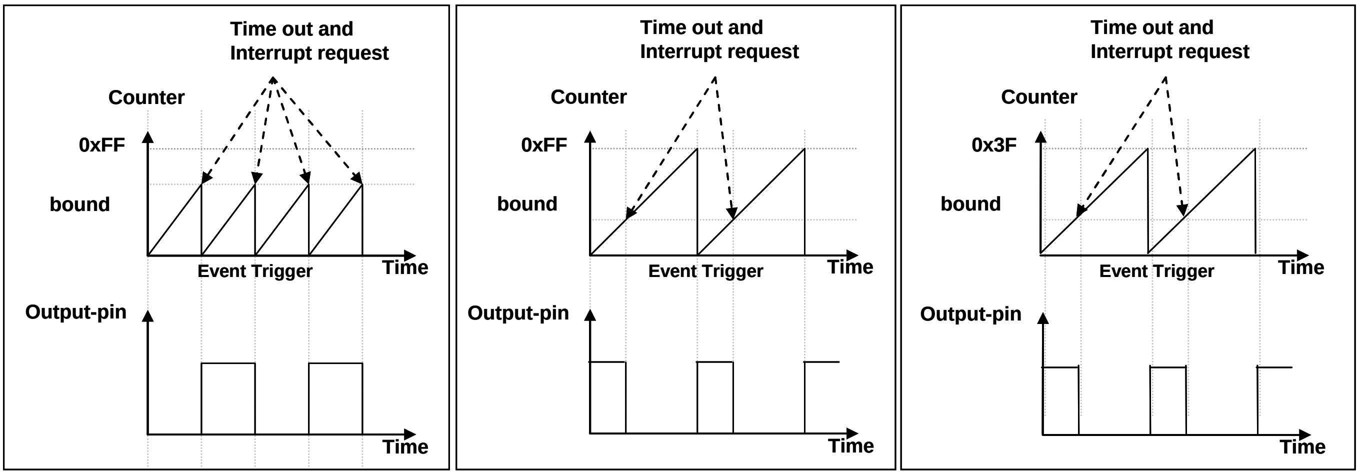


Fig. 10 Timer2 hardware diagram

Preliminary Information, subject to change without notice



**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)

### 5-9-1. 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 (1, 4, 16, 64)

S2 = tm2s[4:0] : scalar register in decimal (1 ~ 31)

Example 1:

tm2c = 0b0001_1100, 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_1100, 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

Preliminary Information, subject to change without notice

Example 3:

tm2c = 0b0001_1100, 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_1100, 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 PA2 is shown as below:

```
void    FPPA0  (void)
{
    .ADJUST_OTP_IHRCR    8MIPS
    …
    tm2ct  =  0x0;
    tm2b  =  0x7f;
    tm2s  =  0b0_00_00001;          //   8-bit PWM, pre-scalar = 1, scalar = 2
    tm2c  =  0b0001_01_0_0;         //   system clock, output=PA2, period mode
    while(1)
    {
        nop;
    }
}
```

### 5-9-2. 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**

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

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

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

S2 = tm2s[4:0] : scalar register in decimal (1 ~ 31)

Preliminary Information, subject to change without notice

Example 1:

tm2c = 0b0001_1110, 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_1110, 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_1110, Y=8MHz

tm2b = 0b1111_1111, K=255

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

➔ frequency of output = 8MHz ÷ ( 256 × 1 × (0+1) ) = 31.25KHz

➔ duty of output = [(255+1) ÷ 256] × 100% = 100%

Example 4:

tm2c = 0b0001_1110, Y=8MHz

tm2b = 0b0000_1001, K = 9

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

➔ frequency of output = 8MHz ÷ ( 256 × 1 × (0+1) ) = 31.25KHz

➔ duty of output = [(9+1) ÷ 256] × 100% = 3.9%

The sample program for using the Timer2 to generate PWM waveform from PA2 is shown as below:

```
void    FPPA0  (void)
{
     .ADJUST_OTP_IHRCR    8MIPS
     wdreset;
     tm2ct = 0x0;
     tm2b = 0x7f;
     tm2s = 0b0_00_00001;          //   8-bit PWM, pre-scalar = 1, scalar = 2
     tm2c = 0b0001_01_1_0;         //   system clock, output=PA2, PWM mode
     while(1)
     {
          nop;
     }
}
```

### 5-9-3. 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:

$$\textbf{Frequency of Output} = Y \div [64 \times S1 \times (S2+1)]$$

$$\textbf{Duty of Output} = [(K+1) \div 64] \times 100\%$$

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

<u>Example 1:</u>

tm2c = 0b0001_1110, 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_1110, 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%

Example 3:

tm2c = 0b0001_1110, Y=8MHz

tm2b = 0b0011_1111, K=63

tm2s = 0b1_00_00000, S1=1, S2=0

➔ frequency of output = 8MHz ÷ ( 64 × 1 × (0+1) ) = 125KHz

➔ duty of output = [(63+1) ÷ 64] × 100% = 100%

Example 4:

tm2c = 0b0001_1110, 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%

## 5-10. WatchDog Timer

The watchdog timer (WDT) is an 11-bit counter with clock coming from ILRC, The frequency of ILRC is around 32 KHz and the timeout period of watchdog timer is 512 ILRC clock cycle, therefore, the timeout period of WDT is around 16ms at 5V and 32ms at 3.3V. However, 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. WDT can be cleared by power-on-reset or by command *wdreset* at any time. When WDT is timeout, P232 will be reset to restart the program execution. The relative timing diagram of watchdog timer is shown as Fig. 12.

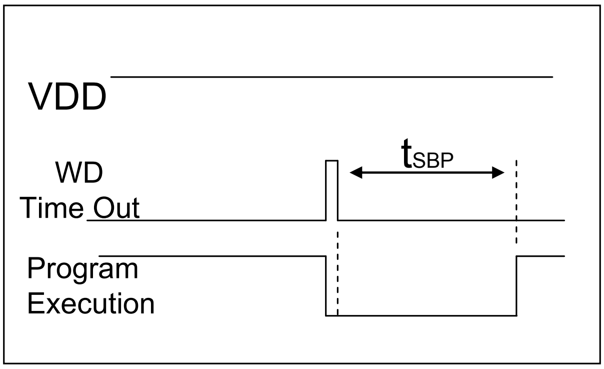


**Fig 12 : Watch Dog Time Out Sequence**

## 5-11. Interrupt

There are six interrupt lines for P232: two external interrupt lines (PA0, PB0, the edge type is specified by ***integsr*** register), Timer16 interrupt, Timer2 interrupt, ADC interrupt and comparator 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. 13. All the interrupt request flags are set by hardware and cleared by software. All the interrupt request lines are also controlled by ***engint*** command (enable global interrupt) to enable interrupt operation and ***disgint*** command (disable global interrupt) to disable it.

Whenever P232 detects the interrupt request and jumps to the interrupt address, global interrupt will be disabled automatically without user's command and global interrupt will be enabled automatically again whenever *reti* instruction is executed. By adjusting the memory location of stack point, the depth of stack pointer for every FPP unit could be fully specified by user to achieve maximum flexibility of system. Interrupt is for FPP0 only; the entry address of interrupt service routine is 0x010.

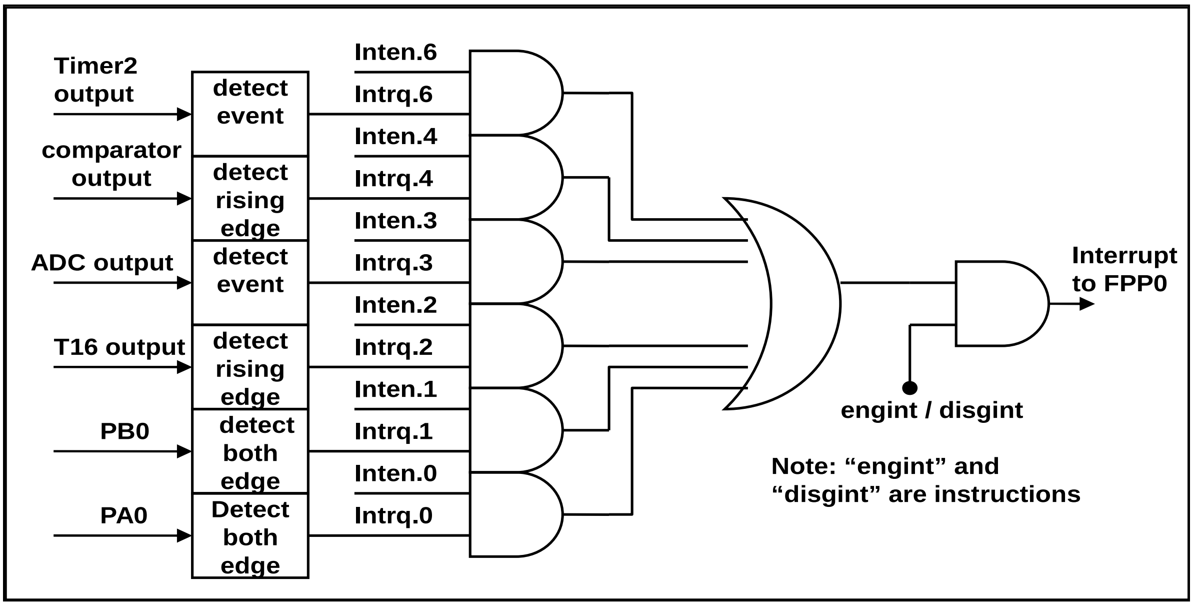


Fig. 13 Hardware diagram of interrupt controller

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.

```
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  (INTRQ.0)
    {                           //   Here for PA0 interrupt service routine
        INTRQ.0 =   0;
        ...
    }
     ...
    POPAF                       //   restore ALU and FLAG register
}
```

If user wants to accept other interrupt event during executing the interrupt service routine, using the ***engint*** to enable it again. Although it allows nesting interrupt by using ***engint***, please take care of stack needed, and please notice that Mini-C can not calculate the required stack for nesting interrupt. Example for nesting interrupt is shown as below:

```
void    Interrupt   (void)
{
     PUSHAF ;
         …
     ENGINT ;
         …
     //  allow other interrupt event
         …
     POPAF ;
}
```

## 5-12. Power-Down

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. The internal low frequency RC oscillator must be enabled before entering the Power-Down mode, means that bit 2 of register *clkmd* (0x03) must be set to high before issuing "*stopsys*" command in order to resume the system when wakeup. The following shows the internal status of P232 in detail when "*stopsys*" command is issued:

- All the oscillator modules are turned off
- Enable internal low RC oscillator (set bit 2 of register clkmd)
- OTP memory is turned off
- The contents of SRAM and registers remain unchanged
- Wake-up sources: ONLY IO toggle.

All the input pins can be used to wake-up the system whenever toggling its state during power-down. The wake-up function of IO pin can be also disabled if the corresponding bit in registers ***padidr***, ***pbdidr*** and ***pcdidr*** is set to high. 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
//      PADIDR   =    xxxx;    //    if the I/O pin will not be used to wakeup system,
//      PBDIDR   =    xxxx;    //    please set the corresponding pins to analog pins.
//      PCDIDR   =    xxxx;    //

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

## 5-13. IO Pins

Most of IO pins in P232 can have digital input/output function as well as analog input. For the digital function, all the bi-directional input/output lines in the P232 can be configured as different function independently by data registers (*pa, pb, pc*), control registers (*pac, pbc, pcc*) and pull-high registers (*paph, pbph, pcph*), open-drain register (*paod*) is for port A only. All these pins have Schmitt-trigger input buffer and output driver with CMOS level. When it is set to output low or open-drain low states, the pull-up resistor is turned off automatically. If user wants to read the pin state, please notice that it should be set to input mode before reading the data port; if user reads the data port when it is set to output or open-drain mode, the reading data comes from data register, NOT from IO pad. All the IO pins can be used to wakeup system when P232 was put in power-down mode. The hardware diagram of IO buffer is shown as Fig. 14, the logic in red color is for port A only.

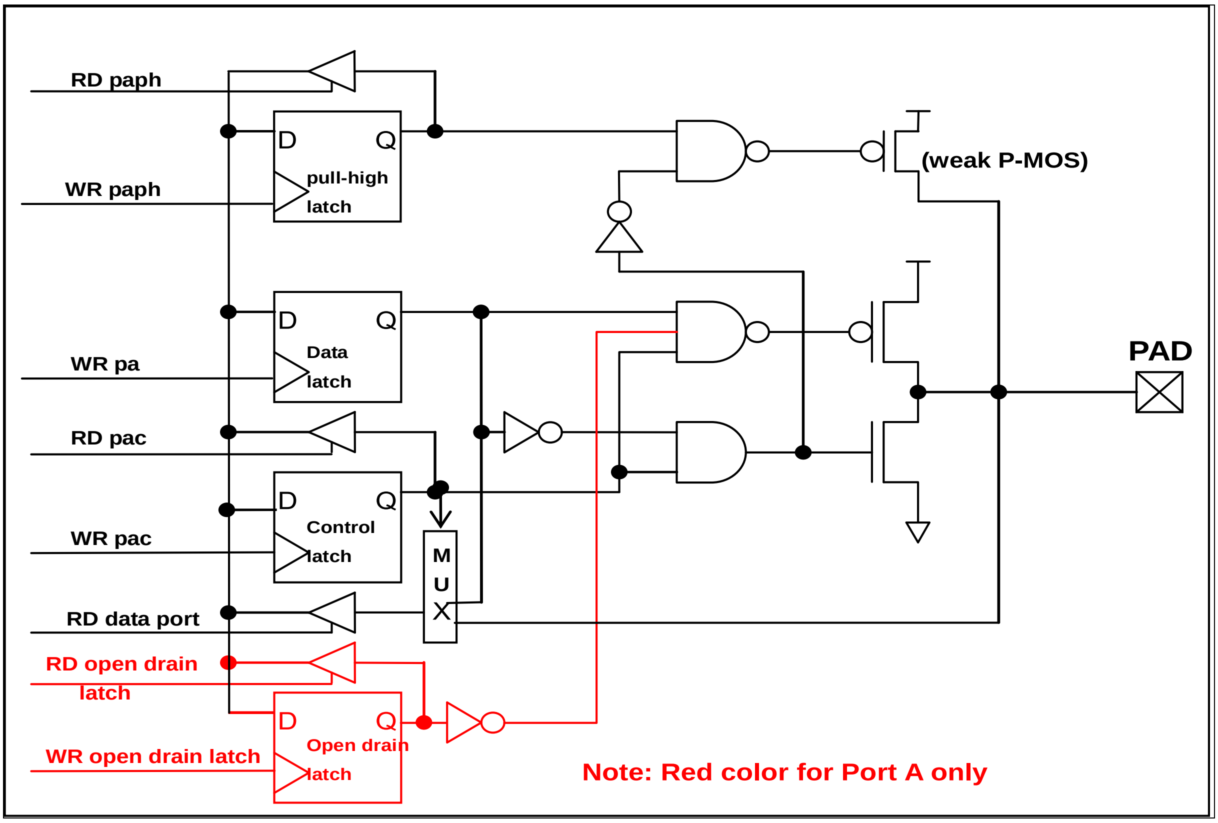


Fig. 14 Hardware diagram of IO buffer

Using the PA0 as the example, the truth table of IO buffer is also shown in the Table 5.

| *pa.0* | *pac.0* | *paph.0* | *paod.0* | Description |
|---|---|---|---|---|
| X | 0 | 0 | X | Input without pull-up resistor |
| X | 0 | 1 | X | Input with pull-up resistor |
| 0 | 1 | X | 0 | Output low without pull-up resistor |
| 1 | 1 | 0 | 0 | Output high without pull-up resistor |
| 1 | 1 | 1 | 0 | Output high with pull-up resistor |
| 0 | 1 | X | 1 | Open drain output low without pull-up resistor |
| 1 | 1 | 0 | 1 | Open drain output tri-state without pull-up resistor |
| 1 | 1 | 1 | 1 | Open drain output tri-state with pull-up resistor |

Table 5 Truth table of PA0

## 5-14. Reset

There are many conditions to reset the P232, including:

(1) Power-On Reset (POR)

(2) PRST# pin active in normal operation

(3) WDT timeout in normal operation

(4) VDD glitch is detected by LVD

POR (Power-On-Reset) is active to put P232 in initial state when power-up, watchdog timeout is the abnormal case of software execution, and LVD is used to detect VDD glitch for the abnormal case of power supply. Once reset is asserted, most of all the registers in P232 will be set to default values, only GDIO (IO address 0x07) keeps the same value after watch dog time out; System should be restarted once abnormal cases happen, or by jumping program counter to address ’h000. The data memory is in uncertain state when reset comes from power-up and LVD; however, the content will be kept when reset comes from PRST# pin or WDT timeout.

## 5-15. Comparator

### 5-15-1. Comparator Hardware Diagram

One comparator is built inside the P232; Fig. 15 shows its hardware diagram. It can compare signals between two pins or with either internal reference voltage $V_{\text{internal R}}$ or internal band-gap 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 PC0/CIN3-, PC5/CIN+, Internal band-gap 1.20 volt, PA3/CIN1-, PA4/CIN2- or $V_{\text{internal R}}$ selected by bit [3:1] of gpcc register, and the plus input of comparator can be PC5/CIN+ or $V_{\text{internal R}}$ selected by bit 0 of gpcc register. The output result can be enabled to output to PB0 directly, or sampled by Time2 clock (TM2_CLK) which comes Timer2 module. The output can be also inversed the polarity by bit 4 of ***gpcc*** register, the comparator output can be used to request interrupt service.

The comparator is disabled after power-on reset and can be enabled by setting gpcc.7=1, the comparator module can be put into power-down mode only when issuing stopsys command which will put P232 into power-down mode.

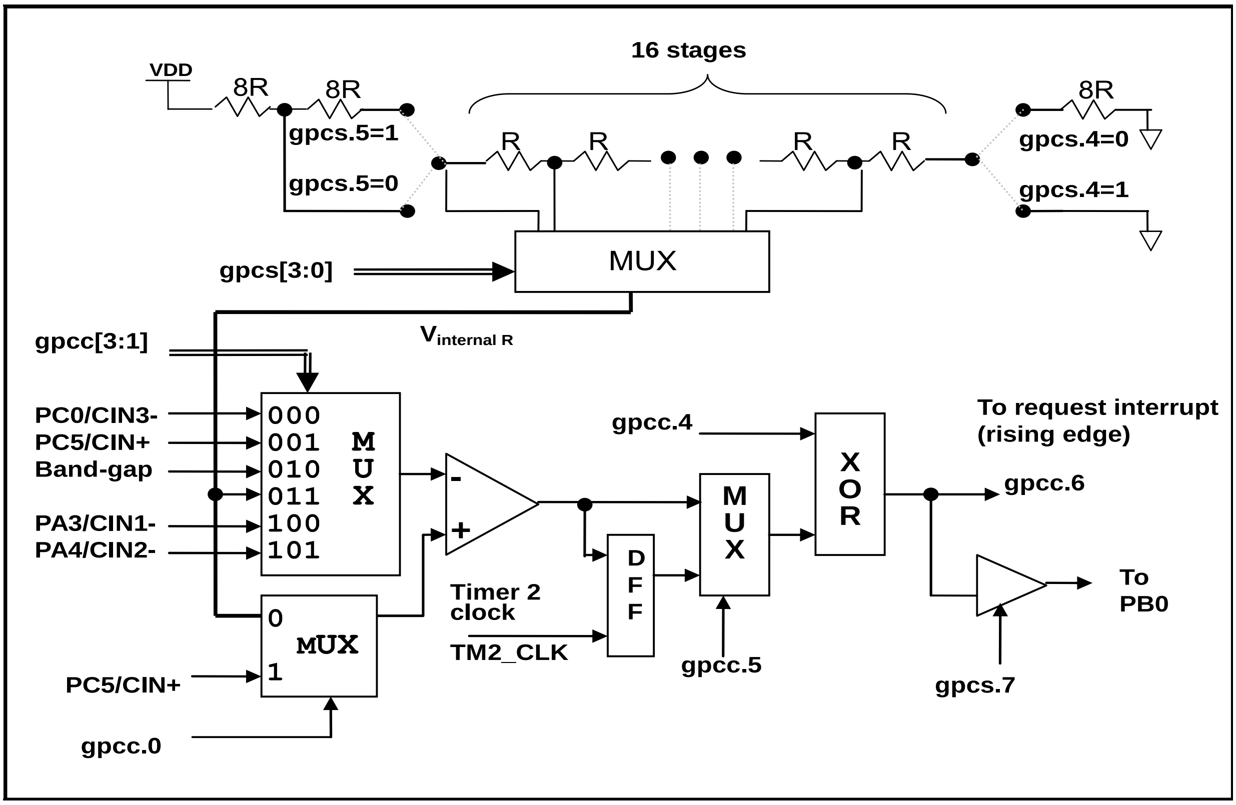


Fig. 15 Hardware diagram of comparator

### 5-15-2. Analog Inputs

A simplified circuit for the analog inputs is shown in the Fig. 15-1. All the analog input pins for comparator are shared function with a digital input which had reverse biased ESD protection diodes to VDD and GND, therefore, the analog input signal must be between VDD and GND.

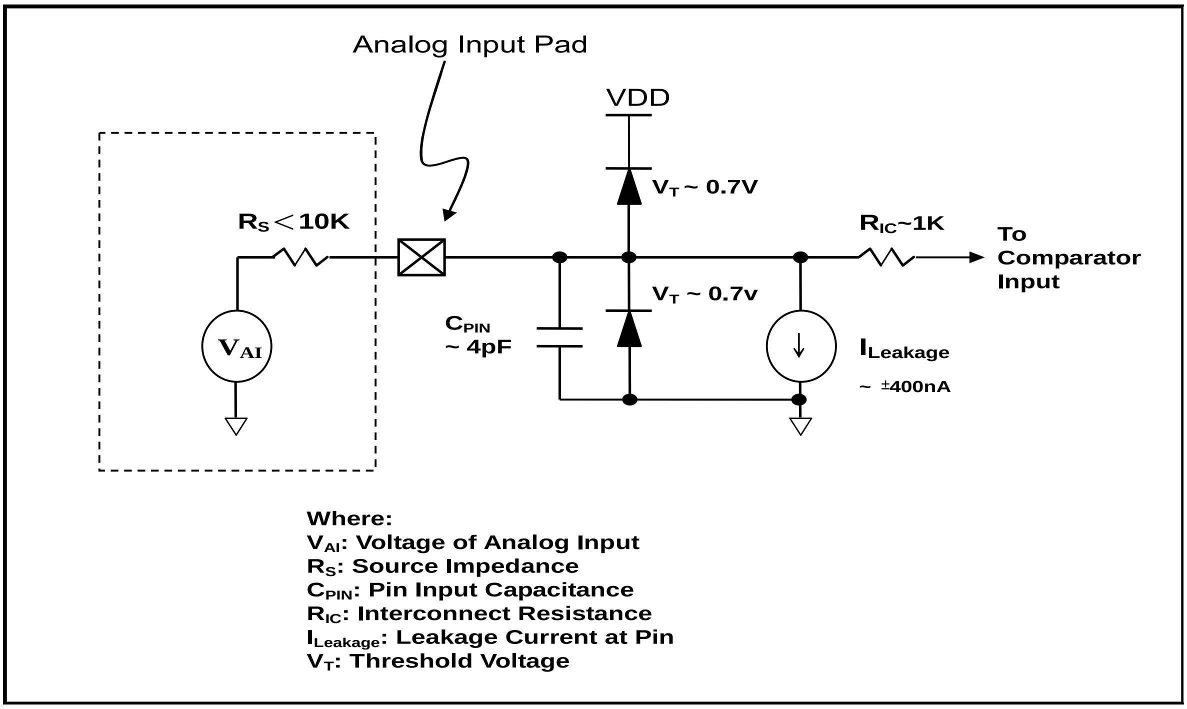
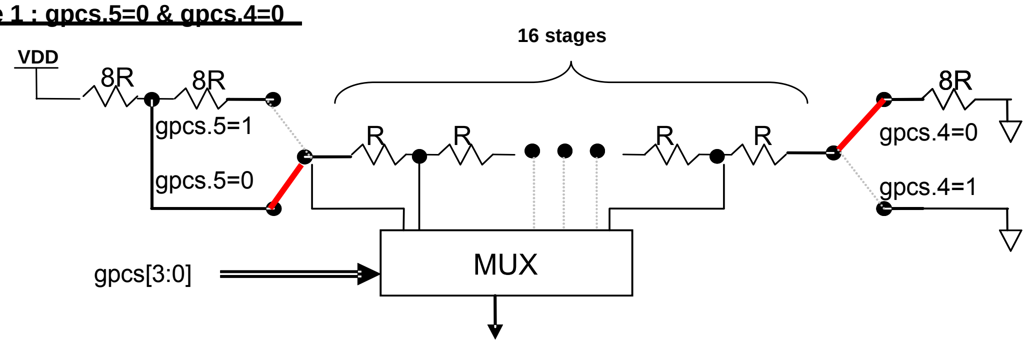


Fig. 15-1 Analog Input Model of Comparator

### 5-15-3. 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. 16 to Fig. 19 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)*VDD to (3/4)*VDD.

Case 1 : gpcs.5=0 & gpcs.4=0



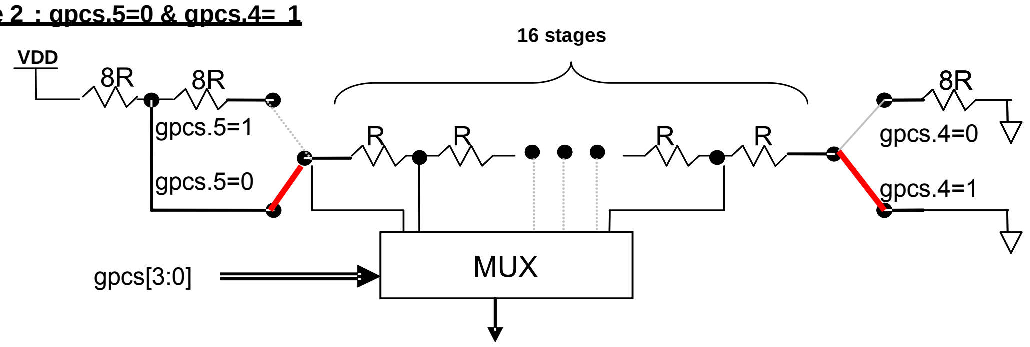
$V_{\text{internal R}}$ = (3/4) VDD ~ (1/4) VDD + (1/32) VDD

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

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

Fig. 16 $V_{\text{internal R}}$ hardware connection if gpcs.5=0 and gpcs.4=0

Case 2 : gpcs.5=0 & gpcs.4= 1

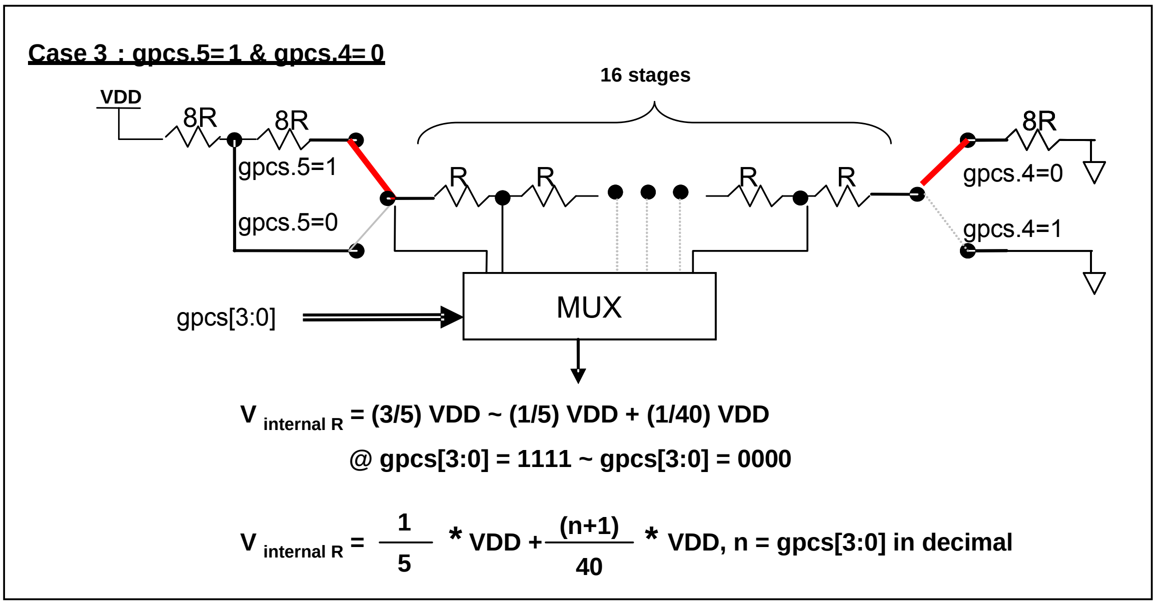


$V_{\text{internal R}}$ = (2/3) VDD ~ (1/24) VDD

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

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

Fig. 17 $V_{\text{internal R}}$ hardware connection if gpcs.5=0 and gpcs.4=1

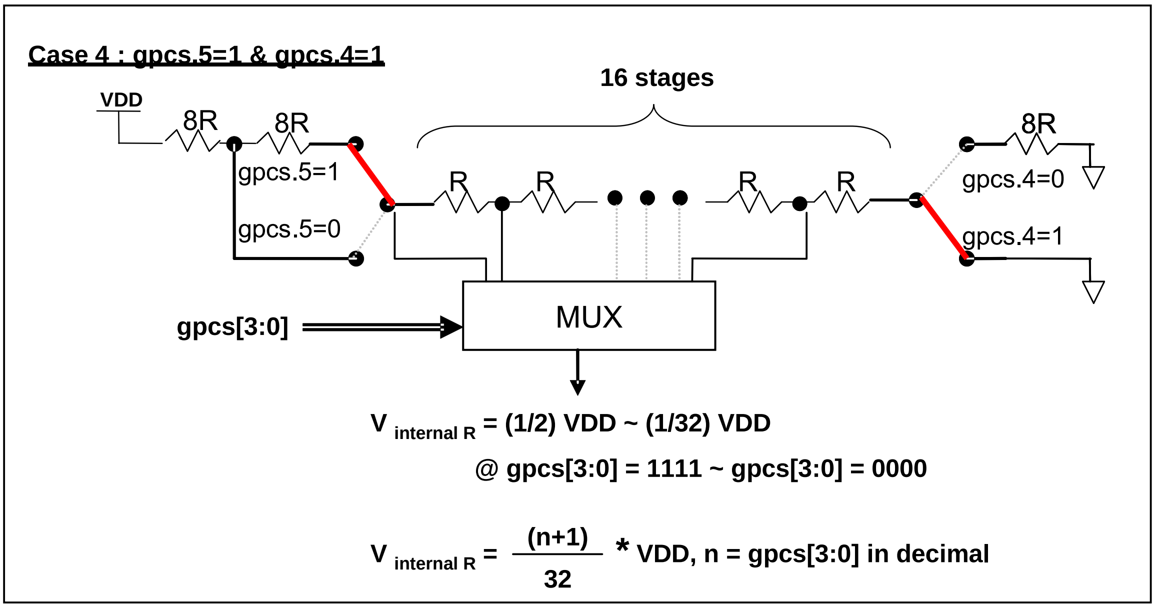


$$V_{\text{internal R}} = (3/5)\ \text{VDD} \sim (1/5)\ \text{VDD} + (1/40)\ \text{VDD}$$

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

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

Fig. 18 $V_{\text{internal R}}$ hardware connection if gpcs.5=1 and gpcs.4=0



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

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

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

Fig. 19 $V_{\text{internal R}}$ hardware connection if gpcs.5=1 and gpcs.4=1

### 5-15-4. Comparator Interrupt Operation

Interrupt request flag intrq.4 will be set to request interrupt service whenever the rising edge of comparator output gpcc.6 is detected, this interrupt request flag will be latched until cleared by software via ***intrq*** register. The interrupt service routine handling is similar to other interrupt request lines. The relative hardware timing diagram is shown as Fig. 19-1.

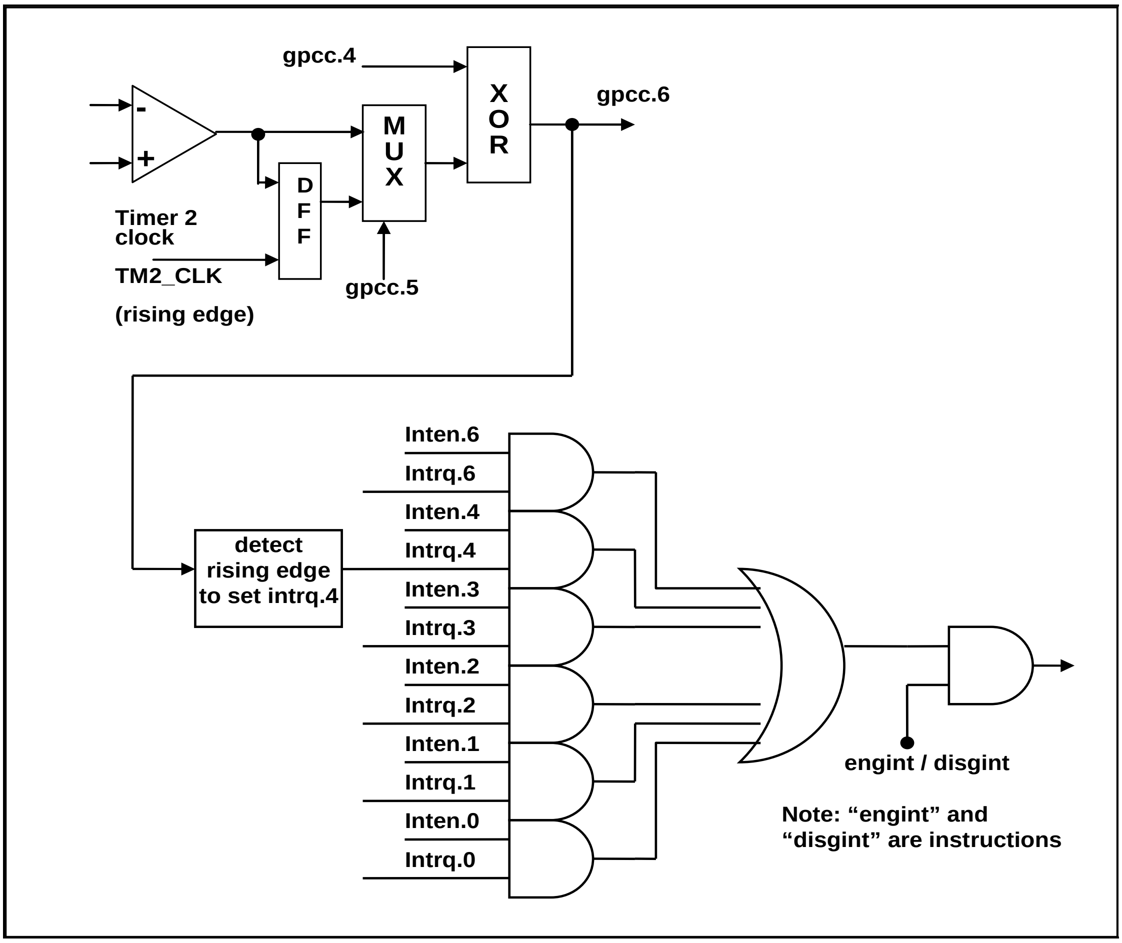


Fig. 19-1 Hardware Diagram of Comparator Interrupt Operation

### 5-15-5. Synchronizing Comparator Output to Timer2

The comparator output can be synchronized with Timer2 by setting gpcc.5=1. When enabled, the comparator output is sampled by the rising edge of Timer2 clock source (TM2_CLK). If the pre-scalar function is used with Timer2, the comparator output is sampled after the pre-scaling and scaling functions, Please refer to Fig. 10 Timer2 hardware diagram and Fig. 15 Comparator hardware diagram, TM2_CLK is the clock source after pre-scaling and scaling functions, and will be sent to Timer2 counter for counting and comparator for sampling clock.

### 5-15-6. Comparator Response Time

The comparator output is undetermined for a period of time after changing an input source or a new reference voltage; this period is referred as the response time. Please refer to Fig. 4-30 and Fig. 4-31 the measurement of comparator responsive time. Typically, the measurement will be based on the compared voltage level near (VDD-1.5)/2. If the compared voltage level is not within the range, the responsive time may be longer.

### 5-15-7. Using the comparator

**Case I:**

Choosing PC0/CIN3- as minus input and $V_{internal\ R}$ with (18/32)*VDD voltage level as plus input, the comparator result will be output to PB0, the comparator result will be output to PB0. $V_{internal\ R}$ is configured as Fig. 16 and gpcs [3:0] = 4b'1001 (n=9) to have $V_{internal\ R}$ = (1/4)*VDD + [(9+1)/32]*VDD = (18/32)*VDD.

```
gpcs      = 0b1_0_00_1001;       // output to PB0, V internal R = VDD*(18/32)
gpcc      = 0b1_0_0_0_000_0;     // enable comp, - input: PC0/CIN3-, + input: V internal R
pcdidr    = 0b000000_0_1;        // disable PC0/CIN3- digital input to prevent leakage current
```

**Case 2:**

Choosing $V_{internal\ R}$ as minus input with (14/32)*VDD voltage level and PC5/CIN+ as plus input, the comparator result will be inversed and then output to PB0. $V_{internal\ R}$ is configured as Fig. 19 and gpcs [3:0] = 4b'1101 (n=13) to have $V_{internal\ R}$ = [(13+1)/32]*VDD = (14/32)*VDD.

```
gpcs      = 0b1_1_1_1_1101;      // V internal R = VDD*(14/32)
gpcc      = 0b1_0_0_1_011_1;     // Inverse output, - input: V internal R, + input: PC5/CIN+
pcdidr    = 0b000000_1_0;        // disable PC5/CIN+ digital input to prevent leakage current
```

### 5-15-8. Using the comparator and band-gap 1.20V

The internal band-gap module can provide 1.20 volt, it can measure the external supply voltage level. The band-gap 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 VDD, the VDD voltage level can be detected by adjusting the voltage level of $V_{internal\ R}$ to compare with band-gap. If N (gpcs[3:0] in decimal) is the number to let $V_{internal\ R}$ closest to band-gap 1.20 volt, the supply voltage VDD can be calculated by using the following equations:

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

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

Preliminary Information, subject to change without notice

## 5-16. Analog-to-Digital Conversion (ADC) module

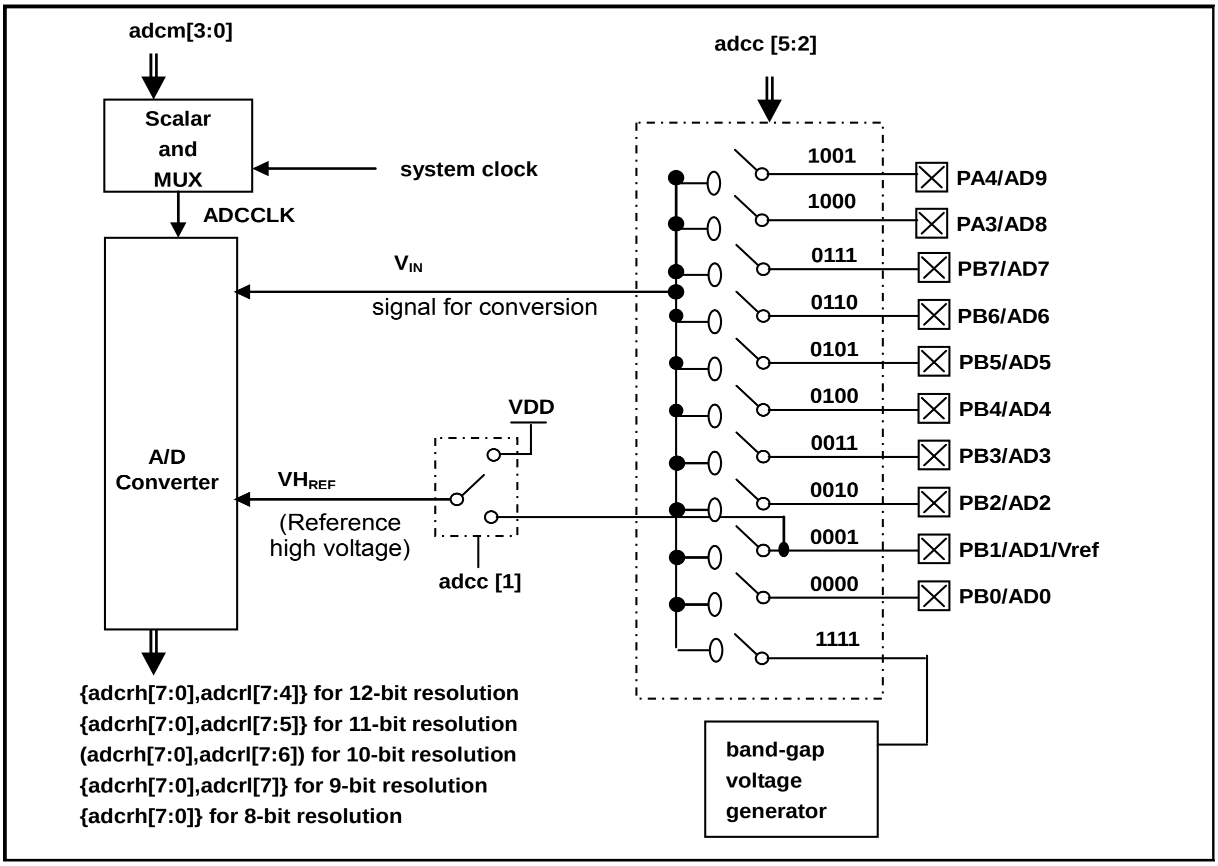


Fig. 20 ADC Block Diagram

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

- ADC Control Register (***adcc***)
- ADC Mode Register (***adcm***)
- ADC Result High/Low Register (***adcrh***, ***adcrl***)
- Port A/B Digital Input Disable Register (***padidr***, ***pbdidr***)

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

**(1)** Configure the ADC module:

- ◆ Configure the voltage reference high by ***adcc*** register
- ◆ Select the ADC input channel by ***adcc*** register
- ◆ Select the bit resolution of ADC by ***adcm*** register
- ◆ Configure the AD conversion clock by ***adcm*** register
- ◆ Configure the pin as analog input by ***padidr**, **pbdidr*** register
- ◆ Enable the ADC module by ***adcc*** register

**(2)** Configure interrupt for ADC: (if desired)

- ◆ Clear the ADC interrupt request flag in bit 3 of ***intrq*** register
- ◆ Enable the ADC interrupt request in bit 3 of ***inten*** register
- ◆ Enable global interrupt by issuing ***engint*** command

**(3)** Start AD conversion:

- ◆ Set ADC process control bit in the ***adcc*** register to start the conversion (set1 ***adcc***.6).

**(4)** Wait for the completion flag of AD conversion, by either:

- ◆ Waiting for the completion flag by using command "*wait1* ***addc***.6"; or
- ◆ Waiting for the ADC interrupt.

**(5)** Read the ADC result registers:

- ◆ Read ***adcrh*** and ***adcrl*** the result registers

**(6)** For next conversion, goto step 1 or step 2 as required.

### 5-16-1. 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. 21, 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 and 10-bit resolution requirements, and 10MΩ under 500Hz input frequency and 10-bit resolution.

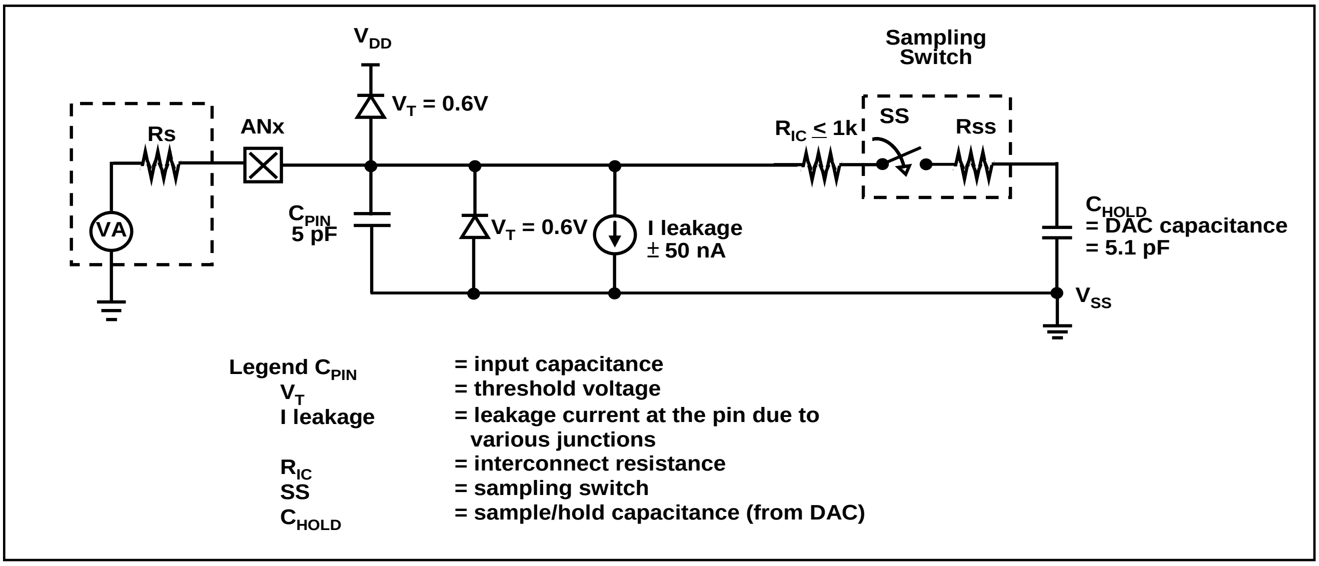

VDD
Sampling Switch
Rs
ANx
VT = 0.6V
RIC ≤ 1k
SS
Rss
VA
CPIN 5 pF
VT = 0.6V
I leakage ± 50 nA
CHOLD = DAC capacitance = 5.1 pF
VSS
Legend CPIN = input capacitance
VT = threshold voltage
I leakage = leakage current at the pin due to various junctions
RIC = interconnect resistance
SS = sampling switch
CHOLD = sample/hold capacitance (from DAC)

**Fig. 21 Analog Input Model**

Before starting the AD conversion, the minimum signal acquisition time should be met for the selected analog input signal. The signal acquisition time ($T_{ACQ}$) of ADC in P232 series is fixed to one clock period of ADCLK, the selection of ADCLK must be met the minimum signal acquisition time.

### 5-16-2. Select the ADC bit resolution

The ADC bit resolution is also selectable from 8-bit to 12-bit, depending on the requirement of customers' application. Higher resolution can detect small signal variation; however, it will take more time to convert the analog signal to digital signal. The selection can be done via ***adcm*** register. The ADC bit resolution should be configured before starting the AD conversion.

### 5-16-3. ADC clock selection

The clock of ADC module (ADCLK) can be selected by ***adcm*** register; there are 8 possible options for ADCLK from sysclk/1 to sysclk/128. 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.

### 5-16-4. AD conversion

The process of AD conversion starts from setting START/DONE bit (bit 6 of ***adcc***) to high, the START/DONE flag for read will be cleared automatically, then converting analog signal bit by bit and finally setting START/DONE high to indicate the completion of AD conversion. If ADCLK is selected, $T_{ADCLK}$ is the period of ADCLK and the AD conversion time can be calculated as follows:

- 8-bit resolution: AD conversion time = 13 $T_{ADCLK}$
- 9-bit resolution: AD conversion time = 14 $T_{ADCLK}$
- 10-bit resolution: AD conversion time = 15 $T_{ADCLK}$
- 11-bit resolution: AD conversion time = 16 $T_{ADCLK}$
- 12-bit resolution: AD conversion time = 17 $T_{ADCLK}$

### 5-16-5. Configure the analog pins

The 11 analog input signals for ADC shared with Port A[3], Port A[4], and Port B[7:0]. To avoid leakage current at the digital circuit, those pins defined for analog input should disable the digital input function (set the corresponding bit of ***padidr*** or ***pbdidr*** register to be 1).

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

### 5-16-6. 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
PBDIDR  =  0B_XXXX_1111;          //  PB0 ~ PB3digital input is disabled
```

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

```
$  ADCC     Enable, PB3;          //  set PB3 as ADC input
$  ADCC     Enable, PB2, VRefH;   //  set PB2 as ADC input, PB1 as VRefH
$  ADCC     Enable, PB0;          //  set PB0 as ADC input
```

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

```
$  ADCM     12BIT, /32;           //  recommend /32 @System Clock=16MHz
$  ADCM     12BIT, /16;           //  recommend /16 @System Clock=8MHz
$  ADCM     12BIT, /8;            //  recommend /8 @System Clock=4MHz
$  ADCM     8BIT, /8;
```

Then, start the ADC conversion:

```
AD_START    =  1;                 //  start ADC conversion
WAIT1       AD_DONE;              //  wait ADC conversion result
```

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

```
WORD        Data;                 //  two bytes result: ADCRH and ADCRL
Data        =  (ADCRH << 8) | ADCRL;
```

The ADC can be disabled by using the following method:

```
$  ADCC     Disable;
```

or

```
ADCC        =  0;
```

Preliminary Information, subject to change without notice

## 5-17. Software Utility (Code Generate)

For FPPA™ architecture, it is easy to partition the program codes module by module based on different functions. In IDE utility, user can use "Code Generate" to generate the program codes, this function can be selected from "Execute → Code Generate" and several functions are provided as below:

- UART IO (without buffer)
- UART with buffer
- SPI IO (without buffer)
- SPI In with buffer
- SPI Out with buffer
- I$^2$C Master / 24Cxx
- I$^2$C Slave
- ADC Module
- Mul Div module (multiplication and division)
- PWM module
- PS2 Mouse – Host
- PS2 Mouse – Slave

# 6. IO Registers

## 6-1. Address mapping of IO registers

| Name | Address | Function | POR / LVD / PRST# reset | WDT reset |
|---|---|---|---|---|
| *flag* | 0x00 | Arithmetic status flag | vvvv_0000 | vvvv_0000 |
| *fppen* | 0x01 | FPP unit enable register | vvvv_vv01 | vvvv_vv01 |
| *sp* | 0x02 | Stack pointer | xxxx_xxxx | uuuu_uuuu |
| *clkmd* | 0x03 | Clock mode register | 1111_0110 (normal)<br>0101_1110(fast) | 1111_0110 (normal)<br>0101_1110(fast) |
| *inten* | 0x04 | Interrupt enable register | v0v0_0000 | v0v0_0000 |
| *intrq* | 0x05 | Interrupt request register | xxxx_xxxx | xxxx_xxxx |
| *t16m* | 0x06 | Timer16 mode register | 0000_0000 | 0000_0000 |
| *gdio* | 0x07 | General data register for IO | 0000_0000 | uuuu_uuuu |
|  | 0x08<br>~<br>0x09 | Reserved |  |  |
| *eoscr* | 0x0a | External oscillator setting register (write only) | 0000_0000 | 0000_0000 |
| *ihrcr* | 0x0b | Internal high RC oscillator control register | 0000_0000 | 0000_0000 |
| *integrs* | 0x0c | Interrupt edge select register (write only) | 0000_0000 | 0000_0000 |
| *padidr* | 0x0d | Port A digital input disable (write only) | 0000_0000 | 0000_0000 |
| *pbdidr* | 0x0e | Port B digital input disable (write only) | 0000_0000 | 0000_0000 |
| *pcdidr* | 0x0f | Port C digital input disable (write only) | vvvv_vv00 | vvvv_vv00 |
| *pa* | 0x10 | Port A data register | xxxx_xxxx | uuuu_uuuu |
| *pac* | 0x11 | Port A control register | 0000_0000 | 0000_0000 |
| *paph* | 0x12 | Port A pull high register | 0000_0000 | 0000_0000 |
| *paod* | 0x13 | Port A open drain register (write only) | 0000_0000 | 0000_0000 |
| *pb* | 0x14 | Port B data register | xxxx_xxxx | uuuu_uuuu |
| *pbc* | 0x15 | Port B control register | 0000_0000 | 0000_0000 |
| *pbph* | 0x16 | Port B pull high register | 0000_0000 | 0000_0000 |
| *pc* | 0x17 | Port C data register | 0000_0000 | uuuu_uuuu |
| *pcc* | 0x18 | Port C control register | 0000_0000 | 0000_0000 |
| *pcph* | 0x19 | Port C pull high register | 0000_0000 | 0000_0000 |
| *adcc* | 0x20 | AD control register | 0000_0000 | 0000_0000 |
| *adcm* | 0x21 | AD mode register | 0000_0000 | 0000_0000 |
| *adcrh* | 0x22 | AD result high register (read only) | xxxx_xxxx | xxxx_xxxx |
| *adcrl* | 0x23 | AD result low register (read only) | xxxx_xxxx | xxxx_xxxx |
| *tm2c* | 0x3c | Timer2 control register | 0000_0000 | 0000_0000 |
| *tm2ct* | 0x3d | Timer2 counter register. | 0000_0000 | 0000_0000 |
| *tm2s* | 0x37 | Timer2 scalar register (write only) | 0000_0000 | 0000_0000 |
| *tm2b* | 0x09 | Timer2 bound register (write only) | 0000_0000 | 0000_0000 |
| *gpcc* | 0x3e | Comparator control register | 0x00_0000 | 0x00_0000 |
| *gpcs* | 0x22 | Comparator selection register (write only) | 0000_0000 | 0000_0000 |
| *misc* | 0x3b | Miscellaneous register (write only) | 0000_0000 | 0000_0000 |
| u: unchanged, v: reserved, x: unknown |  |  |  |  |

## 6-2. Registers Functional Description

### 6-2-1. ACC Status Flag Register (*flag*), IO address = 0x00

| Bit | Reset | R/W | Description |
|---|---|---|---|
| 7-4 | - | - | Reserved. These four bits are “1” when reading. |
| 3 | 0 | R/W | OV (Overflow Flag). This bit is set 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. |

### 6-2-2. FPP unit Enable Register (*fppen*), IO address = 0x01

| Bit | Reset | R/W | Description |
|---|---|---|---|
| 7–2 | - | - | Reserved. Please keep 0 for future compatibility. |
| 1 | 0 | R/W | FPP1 enable. This bit is used to enable FPP1. 0 / 1: disable / enable |
| 0 | 1 | R/W | FPP0 enable. This bit is used to enable FPP0. 0 / 1: disable / enable |

### 6-2-3. 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. |

### 6-2-4. 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<br>(normal boot up)<br>or<br>010<br>(fast boot up)</td><td rowspan="4">R/W</td><td colspan="2">System clock selection:</td></tr>
<tr><td>Type 0, clkmd[3]=0</td><td>Type 1, clkmd[3]=1</td></tr>
<tr><td>000: internal high RC/4<br>001: internal high RC/2<br>010: internal high RC<br>011: external OSC/4<br>100: external OSC/2<br>101: external OSC<br>110: internal low RC/4<br>111: internal low RC (default)</td><td>000: internal high RC/16<br>001: internal high RC/8<br>010: internal high RC/6 (default)<br>011: internal high RC/32<br>100: reserved<br>101: external OSC/8<br>110: Timer2 Clock (TM2_CLK)<br>111: reserved</td></tr>
<tr><td colspan="2">Notes:<br>1. These three bits are configured automatically after boot procedure, the default values are 3‘b111 for normal boot up and 3’b010 for fast boot up.<br>2. Timer2 clock (TM2_CLK) is the clock after pre-scaling and scaling module.</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 or 1</td><td>RW</td><td colspan="2">Clock Type Select. This bit is used to select the clock type in bit [7:5] and configured automatically after boot procedure. 0 / 1: Type 0 / Type 1.</td></tr>
</table>

| | | | |
|---|---|---|---|
| 2 | 1 | R/W | Internal Low RC Enable. 0 / 1: disable / enable |
| 1 | 1 | R/W | Watch Dog Enable. 0 / 1: disable / enable |
| 0 | 0 | R/W | Pin PA5/RESET# function. 0 / 1: PA5 / RESET#. |

### 6-2-5. Interrupt Enable Register (*inten*), IO address = 0x04

| Bit | Reset | R/W | Description |
|---|---|---|---|
| 7 | - | - | Reserved. |
| 6 | 0 | R/W | Enable interrupt from Timer2. 0 / 1: disable / enable. |
| 5 | - | - | 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. 0 / 1: disable / enable. |
| 0 | 0 | R/W | Enable interrupt from PA0.0 / 1: disable / enable. |

### 6-2-6. Interrupt Request Register (*intrq*), IO address = 0x05

| Bit | Reset | R/W | Description |
|---|---|---|---|
| 7 | - | - | Reserved. |
| 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 output, this bit is set by hardware (rising edge) and cleared by software. 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, this bit is set by hardware and cleared by software.<br>0 / 1: No request / Request |
| 0 | - | R/W | Interrupt Request from pin PA0, this bit is set by hardware and cleared by software.<br>0 / 1: No Request / request |

### 6-2-7. Timer16 mode Register (*t16m*), IO address = 0x06

| Bit | Reset | R/W | Description |
|---|---|---|---|
| 7 - 5 | 000 | R/W | Timer Clock source selection.<br>000: disable<br>001: system clock (CLK)<br>010: reserved<br>011: PA0 rising edge (PA0)<br>100: internal high RC oscillator (IHRC)<br>101: external OSC (EOSC)<br>110: internal low RC oscillator (ILRC)<br>111: PA0 falling edge (~PA0) |

Preliminary Information, subject to change without notice

| | | | |
|---|---|---|---|
| 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 selected bit goes high.<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 |

### 6-2-8. General Data register for IO (*gdio*), IO address = 0x07

| Bit | Reset | R/W | Description |
|---|---|---|---|
| 7 – 0 | 00 | R/W | General data for IO. This port is the general data buffer in IO space and cleared when POR or LVD, and it will KEEP the old values when reset from watch dog time out. It can perform the IO operation, like *wait0* gdio.x, *wait1* gdio.x and *tog* gdio.x to replace of operations which instructions are supported in memory space (ex: *wait1* mem; *wait0* mem; *tog* mem). |

### 6-2-9. 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 oscillator selection.<br>00 : reserved<br>01 : 32KHz crystal oscillator<br>10 : 4MHz crystal oscillator<br>11 : 16MHz crystal oscillator |
| 4 – 1 | - | - | Reserved. Please keep 0 for future compatibility. |
| 0 | 0 | WO | Power down IHRC and LVD. 0 / 1 : Normal / Power down |

### 6-2-10. Internal High RC oscillator control Register (*ihrcr*), IO address = 0x0b

| Bit | Reset | R/W | Description |
|---|---|---|---|
| 7 – 0 | 00 | WO | Bit [7:0] for frequency calibration of IHRC. 0x00 for lowest frequency and 0xff for highest frequency. |

### 6-2-11. 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 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 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. |

### 6-2-12. Port A Digital Input Disable Register (p*adidr*), IO address = 0x0d

| Bit | Reset | R/W | Description |
|---|---|---|---|
| 7 | 0 | WO | Disable PA7 digital input and wake up event. 0 / 1 : enable / disable.<br>This bit should be set to high to prevent leakage current when external crystal oscillator is used. If this bit is set to high, PA7 can NOT be used to wake up the system during power-down mode. |
| 6 | 0 | WO | Disable PA6 digital input and wake up event. 0 / 1 : enable / disable.<br>This bit should be set to high to prevent leakage current when external crystal oscillator is used. If this bit is set to high, PA6 can NOT be used to wake up the system during power-down mode. |
| 5 | 0 | WO | Disable wake up event. 0 / 1 : enable / disable.<br>This bit can be set to high to disable wake up from PA5 toggling during power-down mode. |
| 4 | 0 | WO | Disable PA4 digital input and wake up event. 0 / 1 : enable / disable.<br>This bit should be set to high when PA4 is assigned as AD input to prevent leakage current. If this bit is set to high, PA4 can NOT be used to wake up the system during power-down mode. |
| 3 | 0 | WO | Disable PA3 digital input. 0 / 1 : enable / disable.<br>This bit should be set to “1” when PA3 is assigned as AD input to prevent leakage current. If this bit is set to high, PA3 can NOT be used to wake up the system during power-down mode. |
| 2 | 0 | WO | Disable wake up event. 0 / 1 : enable / disable.<br>This bit can be set to high to disable wake up from PA2 toggling during power-down mode. |
| 1 | - | - | Reserved. |
| 0 | 0 | WO | Disable wake up event. 0 / 1 : enable / disable.<br>This bit can be set to high to disable wake up from PA0 toggling during power-down mode. |

### 6-2-13. Port B Digital Input Disable Register (*pbdidr*), IO address = 0x0e

| Bit | Reset | R/W | Description |
|---|---|---|---|
| 7 – 0 | 0x00 | WO | Disable PB7~PB0 digital input to prevent leakage when the pin is assigned for AD input. When disable is selected, the wakeup function from this pin is also disabled.<br>0 / 1 : enable / disable. |

### 6-2-14. Port C Digital Input Disable Register (*pcdidr*), IO address = 0x0f

| Bit | Reset | R/W | Description |
|---|---|---|---|
| 7 – 2 | - | - | Reserved. |
| 1 | 0 | WO | Disable PC5~PC4 digital input to prevent leakage when the pin is assigned for analog input. Therefore, this bit should be set to 1 when comparator is selected and PC5 acts as input pin. When disable is selected, the wakeup function from these two pins is also disabled. 0 / 1 : enable / disable. |
| 0 | 0 | WO | Disable PC3~PC0 digital input to prevent leakage when the pin is assigned for analog input. Therefore, this bit should be set to 1 when comparator is selected and PC0 acts as input pin. When disable is selected, the wakeup function from this pin is also disabled. 0 / 1 : enable / disable. |

### 6-2-15. Port A Data Register (*pa*), IO address = 0x10

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

### 6-2-16. Port A Control Register (*pac*), IO address = 0x11

| Bit | Reset | R/W | Description |
|---|---|---|---|
| 7 - 0 | 8'h00 | 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<br>Please note that PA5 can be INPUT or OUTPUT LOW ONLY, the output state will be tri-state when PA5 is programmed into output mode with data 1. |

### 6-2-17. Port A Pull-High Register (*paph*), IO address = 0x12

| Bit | Reset | R/W | Description |
|---|---|---|---|
| 7 - 0 | 8'h00 | 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<br>Please note that PA5 does NOT have pull-up resistor. |

Preliminary Information, subject to change without notice

### 6-2-18. Port A Open-Drain Register (*paod*), IO address = 0x13

| Bit | Reset | R/W | Description |
|---|---|---|---|
| 7 – 0 | 8'h00 | WO | Port A open-drain register. This register is used to set the output buffer to be open-drain output if the corresponding pin is set to be output mode.<br>0 / 1 : floating / Open-drain output low<br>Notes: The control bit is only active when port is set to output. |

### 6-2-19. Port B Data Register (*pb*), IO address = 0x14

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

### 6-2-20. Port B Control Register (*pbc*), IO address = 0x15

| Bit | Reset | R/W | Description |
|---|---|---|---|
| 7 - 0 | 8'h00 | 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 |

### 6-2-21. Port B Pull-High Register (*pbph*), IO address = 0x16

| Bit | Reset | R/W | Description |
|---|---|---|---|
| 7 - 0 | 8'h00 | 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. 0 / 1 : disable / enable |

### 6-2-22. Port C Data Register (*pc*), IO address = 0x17

| Bit | Reset | R/W | Description |
|---|---|---|---|
| 7 – 6 | - | - | Reserved. |
| 5 – 0 | - | R/W | Bit [5:0] of data register for Port C. |

### 6-2-23. Port C Control Register (*pcc*), IO address = 0x18

| Bit | Reset | R/W | Description |
|---|---|---|---|
| 7 – 6 | - | - | Reserved. |
| 5 - 0 | 6'h00 | R/W | Bit [5:0] of port C control register. This register is used to define input mode or output mode for each corresponding pin. 0 / 1: input / output |

### 6-2-24. Port C Pull-High Register (*pcph*), IO address = 0x19

| Bit | Reset | R/W | Description |
|---|---|---|---|
| 7 – 6 | - | - | Reserved. |
| 5 - 0 | 6'h00 | R/W | Bit [5:0] of Port C pull-high register. This register is used to enable the internal pull-high device on each corresponding pin. 0 / 1 : disable / enable |

### 6-2-25. ADC Control Register (*adcc*), IO address = 0x20

| Bit | Reset | R/W | Description |
|---|---|---|---|
| 7 | 0 | R/W | Enable ADC function. 0/1: Disable/Enable. |
| 6 | - | R/W | ADC process control bit.<br>Write "1" to start AD conversion, and the flag is cleared automatically when starting the AD conversion ;<br>Read "1" to indicate the completion of AD conversion and "0" is in progressing. |
| 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/Vref,<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>1111: bandgap 1.20 volt reference voltage<br>Others: reserved |
| 1 | 0 | R/W | Vref high selector. This bit is used to select the source as Vref high.<br>0 / 1 : VDD / PB1 |
| 0 | - | - | Reserved. (keep 0 for future compatibility) |

### 6-2-26. ADC Mode Register (*adcm*), IO address = 0x21

| Bit | Reset | R/W | Description |
|---|---|---|---|
| 7 – 5 | 000 | WO | Bit Resolution.<br>000:8-bit, AD 8-bit result [7:0] = ***adcrh***[7:0].<br>001:9-bit, AD 9-bit result [8:0] = { ***adcrh***[7:0], ***adcrl***[7] }.<br>010:10-bit, AD 10-bit result [9:0] = { ***adcrh***[7:0], ***adcrl***[7:6] }.<br>011:11-bit, AD 11-bit result [10:0] = { ***adcrh***[7:0], ***adcrl***[7:5] }.<br>100:12-bit, AD 12-bit result [11:0] = { ***adcrh***[7:0], ***adcrl***[7:4] }.<br>others: reserved, |
| 4 | - | - | Reserved (keep 0 for future compatibility) |
| 3 – 1 | 000 | WO | ADC clock source selection.<br>000: sysclk/1,<br>001: sysclk/2,<br>010: sysclk/4,<br>011: sysclk/8,<br>100: sysclk/16,<br>101: sysclk/32,<br>110: sysclk/64,<br>111: sysclk/128, |
| 0 | - | - | Reserved |

### 6-2-27. 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. |

### 6-2-28. 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 |

### 6-2-29. Miscellaneous Register (misc), IO address = 0x3b

| Bit | Reset | R/W | Description |
|---|---|---|---|
| 7 | - | - | Reserved. Please keep 0 for future compatibility. |
| 6 | 0 | WO | Enable the high driving current of external crystal oscillator. 0 / 1 : enable / disable. When this bit is enabled, the settling time of oscillator can be reduced in order to speed up the boot up time; however, the power consumption becomes higher when enabled. This bit can be disabled after the oscillator is stable in order to reduce power consumption. |
| 5 | 0 | WO | Fast wake up.<br>0 : disable. It requires 1024 ILRC (internal low RC) clocks to wake up.<br>1 : enable. It requires only 20 us to wake up normally. |
| 4 – 0 | - | - | Reserved. Please keep 0 for future compatibility. |

### 6-2-30. Timer2 Control Register (tm2c), IO address = 0x3c

| Bit | Reset | R/W | Description |
|---|---|---|---|
| 7 – 4 | 0000 | R/W | Timer2 clock selection.<br>0000 : disable<br>0001 : system clock<br>0010 : internal high RC oscillator (IHRC)<br>0011 : reserved<br>0100 : ILRC<br>0101 : comparator output<br>011x : reserved<br>1000 : PA0 (rising edge)<br>1001 : ~PA0 (falling edge)<br>1010 : PA3 (rising edge)<br>1011 : ~PA3 (falling edge)<br>1100 : PA4 (rising edge)<br>1101 : ~PA4 (falling edge)<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. |

Preliminary Information, subject to change without notice

| | | | |
|---|---|---|---|
| 3 – 2 | 00 | R/W | Timer2 output selection.<br>00 : disable<br>01 : PA2<br>10 : PA3<br>11 : reserved |
| 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. |

### 6-2-31. Timer2 Counter Register (tm2ct), IO address = 0x3d

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

### 6-2-32. Timer2 Scalar Register (tm2s), IO address = 0x37

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

### 6-2-33. Timer2 Bound Register (tm2b), IO address = 0x09

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

### 6-2-34. Comparator Control Register (gpcc), IO address = 0x3e

| Bit | Reset | R/W | Description |
|---|---|---|---|
| 7 | 0 | R/W | Enable comparator.<br>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 : PC0/CIN3-<br>001 : PC5/CIN+<br>010 : Internal 1.20 volt band-gap reference voltage<br>011 : $V_{\text{internal R}}$<br>100 : PA3/CIN1-<br>101 : PA4/CIN2- |
| 0 | 0 | R/W | Selection the plus input (+) of comparator.<br>0 : $V_{\text{internal R}}$<br>1 : PC5/CIN+ |

### 6-2-35. Comparator Selection Register (gpcs), IO address = 0x22

| Bit | Reset | R/W | Description |
|---|---|---|---|
| 7 | 0 | WO | Comparator output enable (to PB0).<br>0 / 1 : disable / enable |
| 6 | - | - | Reserved. |
| 5 | 0 | WO | Selection of high range of comparator. |
| 4 | 0 | WO | Selection of low range of comparator. |
| 3 – 0 | 0000 | WO | Selection the voltage level of comparator.<br>0000 (lowest) ~ 1111 (highest) |

## 6-3. Code Options

There are some options to be chosen when compiling the program, its descriptions is shown as below:

| | |
|---|---|
| Boot up time | Two options for boot up time:<br>◆ Fast boot up: It takes about 20us for boot up procedure<br>◆ Normal boot up: It takes about 32ms for boot up procedure<br>Please notice that fast boot up is only suitable for stable power system environment. |
| LVD voltage level | Four options for LVD voltage level in order to meet different operating frequency.<br>◆ LVD=4.0V for 8MIPS<br>◆ LVD=3.0V for 4MIPS<br>◆ LVD=2.5V for 2MIPS<br>◆ LVD=2.2V for 1MIPS |
| Code Security | Two options to protect the program code inside the chip.<br>No protect or 7/8 protect. |

## 7. Instructions

| Symbol | Description |
|---|---|
| ACC | Accumulator |
| a | Accumulator |
| 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) |
| pc0 | Program counter for FPP0 |
| pc1 | Program counter for FPP1 |

### 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
Example: ***mov pb, a;***
Result: **pb ← a**
Affected flags: 『N』 Z 『N』 C 『N』 AC 『N』 OV

*nmov* M, a

Take the negative logic (2's complement) of ACC to put on memory
Example: ***mov MEM, a;***
Result: MEM ← 〒a
Affected flags: 『N』 Z 『N』 C 『N』 AC 『N』 OV
Application Example:

```
-----------------------------------------------------------------------------------------------------------
    mov     a, 0xf5 ;          // ACC is 0xf5
    nmov    ram9, a;           // ram9 is 0x0b, ACC is 0xf5
-----------------------------------------------------------------------------------------------------------
```

*nmov* a, M

Take the negative logic (2's complement) of memory to put on ACC
Example: ***mov a, MEM ;***
Result: a ← 〒MEM; Flag Z is set when 〒MEM is zero.
Affected flags: 『Y』 Z 『N』 C 『N』 AC 『N』 OV
Application Example:

```
-----------------------------------------------------------------------------------------------------------
    mov     a, 0xf5 ;
    mov     ram9, a ;            // ram9 is 0xf5
    nmov    a, ram9 ;            // ram9 is 0xf5, ACC is 0x0b
-----------------------------------------------------------------------------------------------------------
```

*ldtabh* index

Load high byte data in OTP program memory to ACC by using index as OTP address. It needs 2T to execute this instruction.
Example: ***ldtabh index;***
Result: a ← {bit 15~8 of OTP [index]};
Affected flags: 『N』 Z 『N』 C 『N』 AC 『N』 OV
Application Example:

```
-----------------------------------------------------------------------------------------------------------
    word     ROMptr ;          // declare a pointer of ROM in RAM
    ...
    mov      a, la@TableA ;   // assign pointer to ROM TableA (LSB)
    mov      lb@ROMptr, a ;   // save pointer to RAM (LSB)
    mov      a, ha@TableA ;   // assign pointer to ROM TableA (MSB)
    mov      hb@ROMptr, a ;   // save pointer to RAM (MSB)
    ...
    ldtabh   ROMptr ;         // load TableA MSB to ACC (ACC=0X02)
    ...
TableA :     dc     0x0234, 0x0042, 0x0024, 0x0018 ;
-----------------------------------------------------------------------------------------------------------
```

*ldtabl* index

Load low byte data in OTP to ACC by using index as OTP address. It needs 2T to execute this instruction.
Example: ***ldtabl index;***
Result: a ← {bit7~0 of OTP [index]};
Affected flags: 『N』 Z 『N』 C 『N』 AC 『N』 OV
Application Example:

```
----------------------------------------------------------------------------------------
    word     ROMptr ;       // declare a pointer of ROM in RAM
    ...
    mov      a, la@TableA ; // assign pointer to ROM TableA (LSB)
    mov      lb@ROMptr, a ; // save pointer to RAM (LSB)
    mov      a, ha@TableA ; // assign pointer to ROM TableA (MSB)
    mov      hb@ROMptr, a ; // save pointer to RAM (MSB)
    ...
    ldtabl   ROMptr ;       // load TableA LSB to ACC (ACC=0x34)
    ...
TableA :     dc     0x0234, 0x0042, 0x0024, 0x0018 ;
----------------------------------------------------------------------------------------
```

*ldt16* word

Move 16-bit counting values in Timer16 to memory in word.

Example: ***ldt16 word;***

Result: word ← 16-bit timer

Affected flags: 『N』Z 『N』C 『N』AC 『N』OV

Application Example:

```
----------------------------------------------------------------------------------------
    word     T16val ;       // declare a RAM word
    ...
    clear     lb@ T16val ;  // clear T16val (LSB)
    clear     hb@ T16val ;  // clear T16val (MSB)
    stt16     T16val ;      // initial T16 with 0
    ...
    set1      t16m.5 ;      // enable Timer16
    ...
    set0      t16m.5 ;      // disable Timer16
    ldt16     T16val ;      // save the T16 counting value to T16val
    ...
----------------------------------------------------------------------------------------
```

*stt16* word

Store 16-bit data from memory in word to Timer16.

Example: ***stt16 word;***

Result: 16-bit timer ←word

Affected flags: 『N』Z 『N』C 『N』AC 『N』OV

Application Example:

```
----------------------------------------------------------------------------------------
    word     T16val ;          // declare a RAM word
    ...
    mov      a, 0x34 ;
    mov      lb@ T16val , a ;  // move 0x34 to T16val (LSB)
    mov      a, 0x12 ;
    mov      hb@ T16val , a ;  // move 0x12 to T16val (MSB)
    stt16    T16val ;          // initial T16 with 0x1234
    ...
----------------------------------------------------------------------------------------
```

*idxm* a, index

Move data from specified memory to ACC by indirect method. It needs 2T to execute this instruction.

Example: ***idxm a, index;***

Result: a ← [index], where index is declared by word.

Affected flags: 『N』Z 『N』C 『N』AC 『N』OV

Application Example:

```
---------------------------------------------------------------------------------------------
    word     RAMIndex ;         // declare a RAM pointer
    ...
    mov      a, 0x5B ;          // assign pointer to an address (LSB)
    mov      lb@RAMIndex, a ;   // save pointer to RAM (LSB)
    mov      a, 0x00 ;          // assign 0x00 to an addr.(MSB), be 0
    mov      hb@RAMIndex, a ;   // save pointer to RAM (MSB)
    ...
    idxm     a, RAMIndex ;      // move data in address 0x5B to ACC
---------------------------------------------------------------------------------------------
```

*Idxm* index, a

Move data from ACC to specified memory by indirect method. It needs 2T to execute this instruction.

Example: ***idxm index, a;***

Result: [index] ← a; where index is declared by word.

Affected flags: 『N』Z 『N』C 『N』AC 『N』OV

Application Example:

```
---------------------------------------------------------------------------------------------
    word     RAMIndex ;         // declare a RAM pointer
    ...
    mov      a, 0x5B ;         // assign pointer to an address (LSB)
    mov      lb@RAMIndex, a ;  // save pointer to RAM (LSB)
    mov      a, 0x00 ;         // assign 0x00 to an addr.(MSB), be 0
    mov      hb@RAMIndex, a ;  // save pointer to RAM (MSB)
    ...
    mov      a, 0xA5 ;
    idxm     RAMIndex, a ;     // mov 0xA5 to memory in address 0x5B
---------------------------------------------------------------------------------------------
```

*xch* M

Exchange data between ACC and memory

Example: ***xch MEM ;***

Result: MEM ← a , a ← MEM

Affected flags: 『N』Z 『N』C 『N』AC 『N』OV

*pushaf*

Move the *ACC* and *flag* register to memory that address specified in the stack pointer.

Example: ***pushaf;***

Result: [sp] ← {flag, ACC};
sp ← sp + 2 ;

Affected flags: 『N』Z 『N』C 『N』AC 『N』OV

Preliminary Information, subject to change without notice

| | |
|---|---|
| | 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 |

Preliminary Information, subject to change without notice

| | |
|---|---|
| *clear* M | Clear the content of memory<br>Example: ***clear MEM ;***<br>Result: MEM ← 0<br>Affected flags: 『N』 Z 『N』 C 『N』 AC 『N』 OV |

## 7-3. Shift Operation Instructions

| | |
|---|---|
| *sr* a | Shift right of ACC<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<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<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<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<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<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<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<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 |

| | |
|---|---|
| *swap* M | Swap the high nibble and low nibble of memory<br>Example: ***swap MEM ;***<br>Result: MEM (b3,b2,b1,b0,b7,b6,b5,b4) ← MEM (b7,b6,b5,b4,b3,b2,b1,b0)<br>Affected flags: 『N』 Z 『N』 C 『N』 AC 『N』 OV |

## 7-4. Logic Operation Instructions

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

*xor* M, a

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

*not* a

Perform 1's complement (logical complement) of ACC
Example: ***not a ;***
Result: a ← ～a
Affected flags: 『Y』 Z 『N』 C 『N』 AC 『N』 OV
Application Example:

```
------------------------------------------------------------------------------------------------
    mov       a, 0x38 ;   // ACC=0X38
    not       a ;         // ACC=0XC7
------------------------------------------------------------------------------------------------
```

*not* M

Perform 1's complement (logical complement) of memory
Example: ***not MEM ;***
Result: MEM ← ～MEM
Affected flags: 『Y』 Z 『N』 C 『N』 AC 『N』 OV
Application Example:

```
------------------------------------------------------------------------------------------------
    mov       a, 0x38 ;
    mov       mem, a ;   // mem = 0x38
    not       mem ;       // mem = 0xC7
------------------------------------------------------------------------------------------------
```

*neg* a

Perform 2's complement of ACC
Example: ***neg a;***
Result: a ← 〒a
Affected flags: 『Y』 Z 『N』 C 『N』 AC 『N』 OV
Application Example:

```
------------------------------------------------------------------------------------------------
    mov       a, 0x38 ;   // ACC=0X38
    neg       a ;         // ACC=0XC8
------------------------------------------------------------------------------------------------
```

*neg* M

Perform 2's complement of memory
Example: ***neg MEM;***
Result: MEM ← 〒MEM
Affected flags: 『Y』 Z 『N』 C 『N』 AC 『N』 OV
Application Example:

```
------------------------------------------------------------------------------------------------
    mov       a, 0x38 ;
    mov       mem, a ;    // mem = 0x38
    not       mem ;        // mem = 0xC8
------------------------------------------------------------------------------------------------
```

*comp* a, I

Compare ACC with immediate data

Example: ***comp a, 0x55;***

Result: Flag will be changed by regarding as ( a - 0x55 )

Affected flags: 『Y』Z 『Y』C 『Y』AC 『Y』OV

Application Example:

```
------------------------------------------------------------------------------------------------------------------
    mov      a, 0x38 ;
    comp     a, 0x38 ;   // Z flag is set
    comp     a, 0x42 ;   // C flag is set
    comp     a, 0x24 ;   // C, Z flags are clear
    comp     a, 0x6a ;   // C, AC flags are set
------------------------------------------------------------------------------------------------------------------
```

*comp* a, M

Compare ACC with the content of memory

Example: ***comp a, MEM;***

Result: Flag will be changed by regarding as ( a - MEM )

Affected flags: 『Y』Z 『Y』C 『Y』AC 『Y』OV

Application Example:

```
------------------------------------------------------------------------------------------------------------------
    mov      a, 0x38 ;
    mov      mem, a ;
    comp     a, mem ;   // Z flag is set
    mov      a, 0x42 ;
    mov      mem, a ;
    mov      a, 0x38 ;
    comp     a, mem ;   // C flag is set
------------------------------------------------------------------------------------------------------------------
```

*comp* M, a

Compare ACC with the content of memory

Example: ***comp MEM, a;***

Result: Flag will be changed by regarding as ( MEM - a )

Affected flags: 『Y』Z 『Y』C 『Y』AC 『Y』OV

## 7-5. Bit Operation Instructions

*set0* IO.n

Set bit n of IO port to low

Example: ***set0 pa.5 ;***

Result: set bit 5 of port A to low

Affected flags: 『N』Z 『N』C 『N』AC 『N』OV

*set1* IO.n

Set bit n of IO port to high

Example: ***set1 pb.5 ;***

Result: set bit 5 of port B to high

Affected flags: 『N』Z 『N』C 『N』AC 『N』OV

*tog* IO.n

Toggle bit state of bit n of IO port

Example: ***tog pa.5 ;***

Result: toggle bit 5 of port A

Affected flags: 『N』Z 『N』C 『N』AC 『N』OV

Preliminary Information, subject to change without notice

*set0* M.n

Set bit n of memory to low

Example: ***set0 MEM.5*** ;

Result: set bit 5 of MEM to low

Affected flags: 『N』Z 『N』C 『N』AC 『N』OV

*set1* M.n

Set bit n of memory to high

Example: ***set1 MEM.5 ;***

Result: set bit 5 of MEM to high

Affected flags: 『N』Z 『N』C 『N』AC 『N』OV

*swapc* IO.n

Swap the nth bit of IO port with carry bit

Example: ***swapc IO.0;***

Result: C ← IO.0 , IO.0 ← C

When IO.0 is a port to output pin, carry C will be sent to IO.0;

When IO.0 is a port from input pin, IO.0 will be sent to carry C;

Affected flags: 『N』Z 『Y』C 『N』AC 『N』OV

Application Example1 (serial output) :

```
...
set1     pac.0 ;      // set PA.0 as output
...
set0     flag.1 ;     // C=0
swapc    pa.0 ;       // move C to PA.0 (bit operation), PA.0=0
set1     flag.1 ;     // C=1
swapc    pa.0 ;       // move C to PA.0 (bit operation), PA.0=1
...
```

Application Example2 (serial input) :

```
...
set0      pac.0 ;     // set PA.0 as input
...
swapc    pa.0 ;       // read PA.0 to C (bit operation)
src       a ;         // shift C to bit 7 of ACC
swapc    pa.0 ;       // read PA.0 to C (bit operation)
src       a ;         // shift new C to bit 7, old C
...
```

## 7-6. Conditional Operation Instructions

*ceqsn* a, I

Compare ACC with immediate data and skip next instruction if both are equal.

Flag will be changed like as (a ← a - I)

Example:
```
ceqsn   a, 0x55 ;
inc     MEM ;
goto    error ;
```

Result: If a=0x55, then “goto error”; otherwise, “inc MEM”.

Affected flags: 『Y』Z 『Y』C 『Y』AC 『Y』OV

| | |
|---|---|
| *ceqsn* a, M | Compare ACC with memory and skip next instruction if both are equal.<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 |
| *ceqsn* M, a | Compare ACC with memory and skip next instruction if both are equal.<br>Example: ***ceqsn MEM, a;***<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 |
| *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 |
| *wait0* IO.n | Go next instruction until bit n of IO port is low, otherwise, wait here.<br>Example: ***wait0 pa.5;***<br>Result: Wait bit 5 of port A low to execute next instruction;<br>Affected flags: 『N』 Z 『N』 C 『N』 AC 『N』 OV |
| *wait1* IO.n | Go next instruction until bit n of IO port is high, otherwise, wait here.<br>Example: ***wait1 pa.5;***<br>Result: Wait bit 5 of port A high to execute next instruction;<br>Affected flags: 『N』 Z 『N』 C 『N』 AC 『N』 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 |
| *delay* I | Delay the (N＋1) cycles which N is specified by the immediate data, the timing is based on the executing FPP unit. After the *delay* instruction is executed, the ACC will be zero.<br>Example: ***delay 0x05;***<br>Result: Delay 6 cycles here<br>Affected flags: 『N』 Z 『N』 C 『N』 AC 『N』 OV<br>Note: Because ACC is the temporarily buffer for counting, please make sure that it will not be interrupted when executing this instruction. Otherwise, it may be not the expected delay time. |
| *delay* a | Delay the (N＋1) cycles which N is specified by the content of ACC, the timing is based on the executing FPP unit. After the *delay* instruction is executed, the ACC will be zero.<br>Example: ***delay a;***<br>Result: Delay 16 cycles here if ACC=0fh<br>Affected flags: 『N』 Z 『N』 C 『N』 AC 『N』 OV<br>Note: Because ACC is the temporarily buffer for counting, please make sure that it will not be interrupted when executing this instruction. Otherwise, it may be not the expected delay time. |

| | |
|---|---|
| *delay* M | Delay the (N＋1) cycles which N is specified by the content of memory, the timing is based on the executing FPP unit. After the *delay* instruction is executed, the ACC will be zero.<br>Example: ***delay M;***<br>Result: Delay 256 cycles here if M=ffh<br>Affected flags: 『N』Z 『N』C 『N』AC 『N』OV<br>Note: Because ACC is the temporarily buffer for counting, please make sure that it will not be interrupted when executing this instruction. Otherwise, it may be not the expected delay time. |
| *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 |
| *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 FPP0<br>Affected flags: 『N』Z 『N』C 『N』AC 『N』OV |

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

## 7-8. Summary of Instructions Execution Cycle

| | |
|---|---|
| 2T | ***ldtabh, ldtabl, idxm*** |
| 1T | Others |

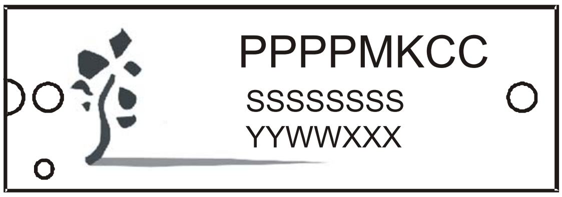
## 7-9. Summary of affected flags by Instructions

| 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 | - | - | - | - | *nmov* M, a | - | - | - | - |
| *nmov* a, M | Y | - | - | - | *ldtabh* index | - | - | - | - | *ldtabl* index | - | - | - | - |
| *ldt16* index | - | - | - | - | *stt16* index | | | | | *xch* M | - | - | - | - |
| *idxm* a, index | - | - | - | - | *idxm* index, a | - | - | - | - | *wdreset* | - | - | - | - |
| *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 | - | - | - | - |
| *swap* M | - | - | - | - | *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* a, M | Y | - | - | - |
| *xor* M, a | Y | - | - | - | *not* a | Y | - | - | - | *not* M | Y | - | - | - |
| *neg* a | Y | - | - | - | *neg* M | Y | - | - | - | *comp* a, I | Y | Y | Y | Y |
| *comp* a, M | Y | Y | Y | Y | *comp* M, a | Y | Y | Y | Y | *set0* IO.n | - | - | - | - |
| *set1* IO.n | - | - | - | - | *tog* 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 | *ceqsn* M, a | 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 | *wait0* IO.n | - | - | - | - | *wait1* IO.n | - | - | - | - |
| *call* label | - | - | - | - | *goto* label | - | - | - | - | *delay* a | - | - | - | - |
| *delay* I | - | - | - | - | *delay* M | - | - | - | - | *ret* I | - | - | - | - |
| *ret* | | | | | *reti* | - | - | - | - | *nop* | - | - | - | - |
| *pcadd* a | | | | | *engint* | - | - | - | - | *disgint* | - | - | - | - |
| *stopsys* | | | | | *reset* | - | - | - | - | *cneqsn a, I* | Y | Y | Y | Y |
| *xor a, IO* | Y | | | | *xor IO, a* | | | | | *cneqsn a, M* | Y | Y | Y | Y |

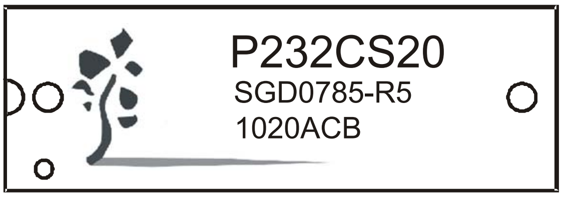
# 8. Package Information

## 8-1. Package Marking Information

Top Mark



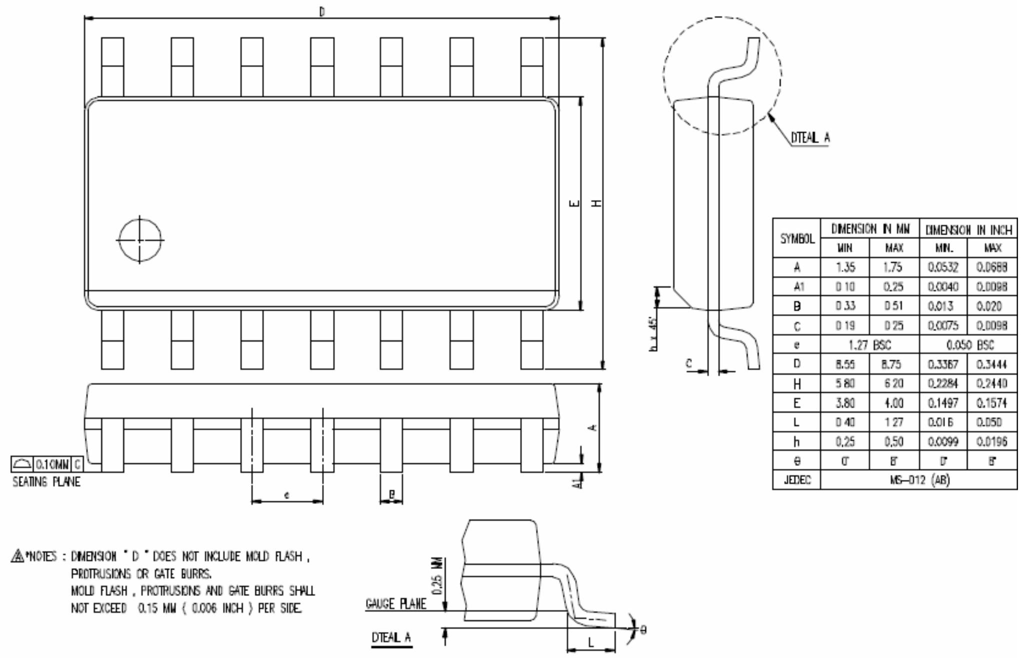
Example



**Legend:**

| | |
|---|---|
| **PPPP** | **PADAUK Technology part number information** |
| **M** | **Memory type of product** |
| | **C: OTP** |
| | **F: Flash** |
| **K** | **Package type of product** |
| | **S: SOP** |
| | **D: DIP** |
| | **SS: SSOP** |
| | **K: Skinny DIP** |
| **CC** | **Pin count of package** |
| **SS……S** | **Lot number information** |
| **YY** | **Year Code (last 2 digits of calendar year)** |
| **WW** | **Week Code** |
| **XXX** | **PADAUK Technology package information** |

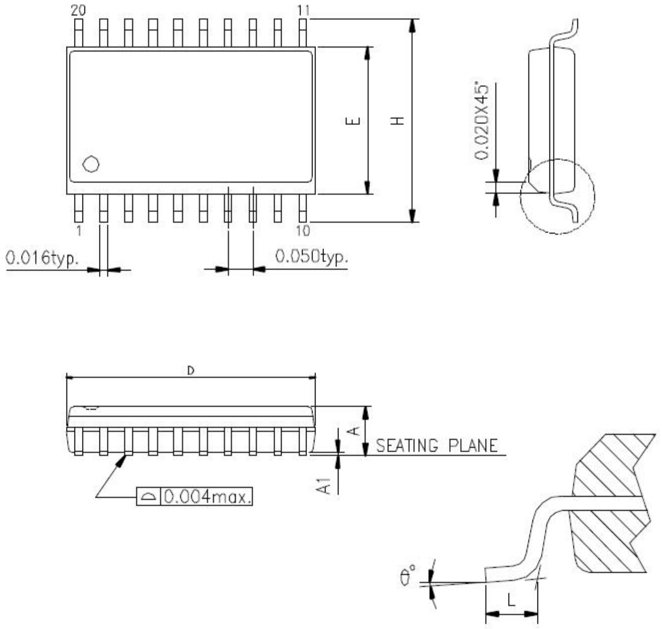
## 8-2. Lead of SOP14



| SYMBOL | DIMENSION IN MM |  | DIMENSION IN INCH |  |
|---|---|---|---|---|
|  | MIN | MAX | MIN. | MAX |
| A | 1.35 | 1.75 | 0.0532 | 0.0688 |
| A1 | 0 10 | 0.25 | 0.0040 | 0.0098 |
| B | 0 33 | 0 51 | 0.013 | 0.020 |
| C | 0 19 | 0 25 | 0.0075 | 0.0098 |
| e | 1.27 BSC |  | 0.050 BSC |  |
| D | 8.55 | 8.75 | 0.3367 | 0.3444 |
| H | 5 80 | 6 20 | 0.2284 | 0.2440 |
| E | 3.80 | 4.00 | 0.1497 | 0.1574 |
| L | 0 40 | 1 27 | 0.01 6 | 0.050 |
| h | 0.25 | 0.50 | 0.0099 | 0.0196 |
| θ | 0° | 8° | 0° | 8° |
| JEDEC | MS–012 (AB) |  |  |  |

⚠*NOTES : DIMENSION " D " DOES NOT INCLUDE MOLD FLASH , PROTRUSIONS OR GATE BURRS.
MOLD FLASH , PROTRUSIONS AND GATE BURRS SHALL NOT EXCEED 0.15 MM ( 0.006 INCH ) PER SIDE.

## 8-3. Lead of SOP20



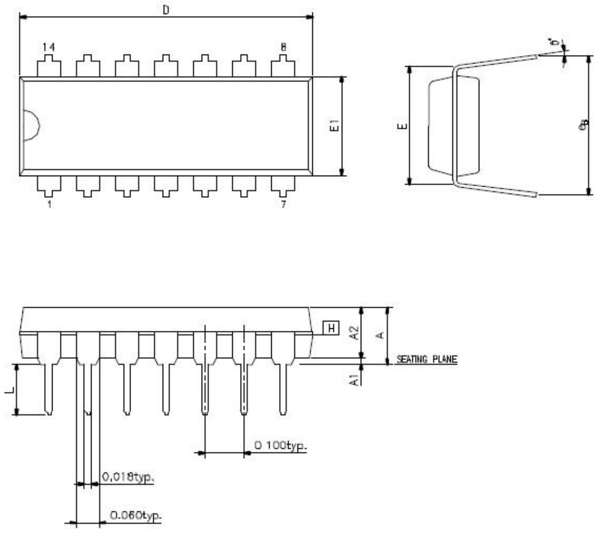
| SYMBOLS | MIN. | MAX. |
|---|---|---|
| A | 0.093 | 0.104 |
| A1 | 0.004 | 0.012 |
| D | 0.496 | 0.508 |
| E | 0.291 | 0.299 |
| H | 0.394 | 0.419 |
| L | 0.016 | 0.050 |
| θ° | 0 | 8 |

UNIT : INCH

NOTES:
1.JEDEC OUTLINE : MS–013 AC
2.DIMENSIONS "D" DOES NOT INCLUDE MOLD FLASH, PROTRUSIONS OR GATE BURRS.MOLD FLASH, PROTRUSIONS AND GATE BURRS SHALL NOT EXCEED .15mm (.006in) PER SIDE.
3.DIMENSIONS "E" DOES NOT INCLUDE INTER–LEAD FLASH, OR PROTRUSIONS. INTER–LEAD FLASH AND PROTRUSIONS SHALL NOT EXCEED .25mm (.010in) PER SIDE.

Preliminary Information, subject to change without notice

## 8-4. Lead of DIP14



| SYMBOLS | MIN. | NOR. | MAX. |
|---|---|---|---|
| A | – | – | 0.210 |
| A1 | 0.015 | – | – |
| A2 | 0.125 | 0.130 | 0.135 |
| D | 0.735 | 0.750 | 0.775 |
| E | | 0.300 BSC. | |
| E1 | 0.245 | 0.250 | 0.255 |
| L | 0.115 | 0.130 | 0.150 |
| $e_B$ | 0.335 | 0.355 | 0.375 |
| θ° | 0 | 7 | 15 |

UNIT : INCH

NOTES:

1.JEDEC OUTLINE : MS-001 AA

2."D","E1" DIMENSIONS DO NOT INCLUDE MOLD FLASH OR PROTRUSIONS.MOLD FLASH OR PROTRUSIONS SHALL NOT EXCEED .010 INCH.

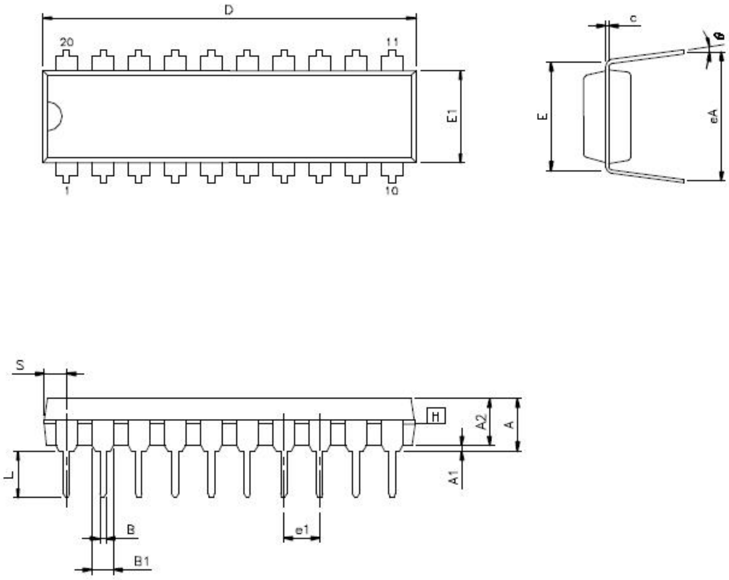
3.eB IS MEASURED AT THE LEAD TIPS WITH THE LEADS UNCONSTRAINED.

4.POINTED OR ROUNDED LEAD TIPS ARE PREFERRED TO EASE INSERTION.

5.DISTANCE BETWEEN LEADS INCLUDING DAM BAR PROTRUSIONS TO BE .005 INCH MININUM.

6.DATUM PLANE H COINCIDENT WITH THE BOTTOM OF LEAD, WHERE LEAD EXITS BODY

## 8-5. Lead of DIP20



| SYMBOLS | MIN | NOM | MAX |
|---|---|---|---|
| A | – | – | 0.175 |
| A1 | 0.015 | – | – |
| A2 | 0.125 | 0.130 | 0.135 |
| B | 0.016 | 0.018 | 0.020 |
| B1 | 0.058 | 0.060 | 0.064 |
| c | 0.008 | 0.010 | 0.011 |
| D | 1.012 | 1.026 | 1.040 |
| E | 0.290 | 0.300 | 0.310 |
| E1 | 0.245 | 0.250 | 0.255 |
| e1 | 0.090 | 0.100 | 0.110 |
| L | 0.120 | 0.130 | 0.140 |
| θ | 0 | – | 15 |
| eA | 0.335 | 0.355 | 0.375 |
| S | – | – | 0.075 |

UNIT : INCH

NOTES:

1.JEDEC OUTLINE : MS-001 AD

2 "D","E1" DIMENSIONS DO NOT INCLUDE MOLD FLASH OR PROTRUSIONS.MOLD FLASH OR PROTRUSIONS SHALL NOT EXCEED .010 INCH.

3.eA IS MEASURED AT THE LEAD TIPS WITH THE LEADS UNCONSTRAINED.

4.POINTED OR ROUNDED LEAD TIPS ARE PREFERRED TO EASE INSERTION.

5.DISTANCE BETWEEN LEADS INCLUDING DAM BAR PROTRUSIONS TO BE .005 INCH MININUM.

6.DATUM PLANE H COINCIDENT WITH THE BOTTOM OF LEAD, WHERE LEAD EXITS BODY.

Preliminary Information, subject to change without notice