



PFC232

Industrial Grade - 8bit MTP MCU (FPPA™)
with 12-bit Enhanced ADC

Datasheet

Version 0.07

August 21, 2026

Copyright © 2026 by PADAUK Technology Co., Ltd., all rights reserved.

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

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

IMPORTANT NOTICE

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

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

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

Table of Contents

Revision History	6
Usage Warning	6
1. Features	7
1.1. Special Features.....	7
1.2. System Features	7
1.3. CPU Features.....	7
1.4. Ordering/ Package Information	8
2. General Description and Block Diagram	9
3. Pin Definition and Functional Description	10
4. Central Processing Unit (CPU)	12
4.1. Functional Description	12
4.1.1. Two FPPA Units	12
4.1.2. Single FPPA Unit.....	13
4.1.3. Program Counter	13
4.1.4. Program Structure	14
4.1.5. Arithmetic and Logic Unit.....	15
4.2. Storage Memory.....	15
4.2.1. Program Memory – ROM	15
4.2.2. Data Memory – SRAM.....	17
4.2.3. System Register	18
4.2.3.1. ACC Status Flag Register (FLAG), address = 0x00	19
4.2.3.2. FPPA unit Enable Register (FPPEN), address = 0x01	19
4.2.3.3. MISC Register (MISC), address = 0x1F	19
4.3. The Stack	20
4.3.1. Stack Pointer Register (SP), address = 0x02	21
4.4. Code Options	21
5. Oscillator and System Clock	23
5.1. Internal High RC Oscillator and Internal Low RC Oscillator	23
5.2. External Crystal Oscillator.....	23
5.2.1. External Oscillator Setting Register (EOSCR), address = 0x0A	24
5.2.2. Usages and Precautions of External Oscillator	24
5.3. System Clock and IHRC Calibration	25
5.3.1. System Clock.....	25
5.3.1.1. Clock Mode Register (CLKMD), address = 0x03	26
5.3.2. Frequency Calibration.....	26
5.3.2.1. Special Statement.....	28
5.3.3. System Clock Switching	28
6. Reset	29
6.1. Power On Reset - POR	29
6.2. Low Voltage Reset - LVR.....	30
6.3. Watch Dog Timeout Reset.....	31
6.4. External Reset Pin - PRSTB	32
7. System Operating Mode	33
7.1. Power-Save Mode (“stopexe”)	33
7.2. Power-Down Mode (“stopsys”).....	34
7.3. Wake-Up.....	35
8. Interrupt	36
8.1. Interrupt Enable Register (INTEN), address = 0x04	37
8.2. Interrupt Request Register (INTRQ), address = 0x05.....	37
8.3. Interrupt Edge Select Register (INTEGS), address = 0x0C.....	38
8.4. Interrupt Work Flow	38
8.5. General Steps to Interrupt	39
8.6. Example for Using Interrupt.....	40

9. I/O Port.....	41
9.1. IO Related Registers	41
9.1.1. Port A Digital Input Enable Register (<i>PADIER</i>), address = 0x0D	41
9.1.2. Port B Digital Input Enable Register (<i>PBDIER</i>), address = 0x0E	41
9.1.3. Port A Data Registers (PA), address = 0x10	41
9.1.4. Port A Control Registers (PAC), address = 0x11	41
9.1.5. Port A Pull-High Registers (PAPH), address = 0x12	42
9.1.6. Port A Pull-Low Register (PAPL), address = 0x13	42
9.1.7. Port B Data Registers (PB), address = 0x14	42
9.1.8. Port B Control Registers (PBC), address = 0x15	42
9.1.9. Port B Pull-High Registers (PBPH), address = 0x16	42
9.1.10. Port B Pull-Low Register (PBPL), address = 0x17	42
9.2. IO Structure and Functions	43
9.2.1. IO Pin Structure.....	43
9.2.2. IO Pin Functions.....	43
9.2.3. IO Pin Usage and Setting	44
10. Timer / PWM Counter	45
10.1. 16-bit Timer (Timer16).....	45
10.1.1. Timer16 Introduction	45
10.1.2. Timer16 Time Out	46
10.1.3. Timer16 Mode Register (T16M), address = 0x06	47
10.2. 8-bit Timer with PWM Generation (Timer2, Timer3).....	48
10.2.1. Timer2, Timer3 Related Registers	49
10.2.1.1. Timer2 Bound Register (<i>TM2B</i>), address = 0x09	49
10.2.1.2. Timer2 Counter Register (<i>TM2CT</i>), address = 0x1D	49
10.2.1.3. Timer2 Scalar Register (<i>TM2S</i>), address = 0x1E	49
10.2.1.4. Timer2 Control Register (<i>TM2C</i>), address = 0x1C	50
10.2.1.5. Timer3 Counter Register (<i>TM3CT</i>), address = 0x33	50
10.2.1.6. Timer3 Scalar Register (<i>TM3S</i>), address = 0x34	50
10.2.1.7. Timer3 Bound Register (<i>TM3B</i>), address = 0x38	51
10.2.1.8. Timer3 Control Register (<i>TM3C</i>), address = 0x32	51
10.2.2. Using the Timer2 to Generate Periodical Waveform	52
10.2.3. Using the Timer2 to Generate 8-bit PWM Waveform	53
10.2.4. Using the Timer2 to Generate 6-bit PWM Waveform	54
10.3. 11-bit PWM Generation.....	55
10.3.1. PWM Waveform	55
10.3.2. Hardware and Timing Diagram	55
10.3.3. Equations for 11-bit PWM Generator.....	57
10.3.4. 11bit PWM Related Registers.....	58
10.3.4.1. PWMG0 control Register (PWMG0C), address = 0x20	58
10.3.4.2. PWMG0 Scalar Register (PWMG0S), address = 0x21	58
10.3.4.3. PWMG0 Duty Value High Register (PWMG0DTH), address = 0x22.....	58
10.3.4.4. PWMG0 Duty Value Low Register (PWMG0DTL), address = 0x23.....	58
10.3.4.5. PWMG0 Counter Upper Bound High Register (PWMG0CUBH), address = 0x24	59
10.3.4.6. PWMG0 Counter Upper Bound Low Register (PWMG0CUBL), address = 0x25	59
10.3.4.7. PWMG1 Control Register (PWMG1C), address = 0x26.....	59
10.3.4.8. PWMG1 Scalar Register (PWMG1S), address = 0x27	59
10.3.4.9. PWMG1 Duty Value High Register (PWMG1DTH), address = 0x28.....	60
10.3.4.10. PWMG1 Duty Value Low Register (PWMG1DTL), address = 0x29.....	60
10.3.4.11. PWMG1 Counter Upper Bound High Register (PWMG1CUBH), address = 0x2A	60
10.3.4.12. PWMG1 Counter Upper Bound Low Register (PWMG1CUBL), address = 0x2B	60
10.3.4.13. PWMG2 Control Register (PWMG2C), address = 0x2C	60
10.3.4.14. PWMG2 Scalar Register (PWMG2S), address = 0x2D.....	61
10.3.4.15. PWMG2 Duty Value High Register (PWMG2DTH), address = 0x2E	61
10.3.4.16. PWMG2 Duty Value Low Register (PWMG2DTL), address = 0x2F.....	61
10.3.4.17. PWMG2 Counter Upper Bound High Register (PWMG2CUBH), address = 0x30	61
10.3.4.18. PWMG2 Counter Upper Bound Low Register (PWMG2CUBL), address = 0x31	61
10.3.5. Examples of PWM Waveforms with Complementary Dead Zones	62
11. Special Functions	64

11.1.	Comparator	64
11.1.1.	Comparator Control Register (GPCC), address = 0x18	65
11.1.2.	Comparator Selection Register (GPCS), address = 0x19	65
11.1.3.	Internal Reference Voltage ($V_{\text{internal R}}$)	66
11.1.4.	Using the Comparator	68
11.1.5.	Using the Comparator and Bandgap 1.20V	69
11.2.	VDD/2 Bias Voltage Generator	70
11.3.	Operational Amplifier (OPA) module	71
11.3.1.	OPA Comparator Mode.....	71
11.3.2.	OPA Amplifier Mode.....	71
11.3.3.	OPA Control Register (OPAC), address = 0x1A.....	72
11.3.4.	OPA Offset Register (OPAOFS), address = 0x07	72
11.4.	Analog-to-Digital Conversion (ADC) module.....	73
11.4.1.	The input requirement for AD conversion.....	74
11.4.2.	Select the reference high voltage	75
11.4.3.	ADC clock selection.....	75
11.4.4.	Configure the analog pins.....	75
11.4.5.	Using the ADC	76
11.4.6.	ADC Related Registers	77
11.4.6.1.	ADC Control Register (ADCC), address = 0x35	77
11.4.6.2.	ADC Mode Register (ADCM), address = 0x36	77
11.4.6.3.	ADC Regulator Control Register (ADCRGC), address = 0x39	78
11.4.6.4.	ADC Result High Register (ADCRH), address = 0x37.....	78
11.4.6.5.	ADC Result Low Register (ADCRL), address = 0x38	78
11.5.	Multiplier.....	79
11.5.1.	Multiplier Operand Register (MULOP), address = 0x08	79
11.5.2.	Multiplier Result High Byte Register (MULRH), address = 0x09	79
12.	Notes for Emulation.....	80
13.	Program Writing.....	81
13.1.	Normal Programming Mode	81
13.2.	Limited-Voltage Programming Mode.....	81
13.3.	On-Board Writing.....	82
14.	Device Characteristics	83
14.1.	Absolute Maximum Ratings	83
14.2.	DC/AC Characteristics	83
14.3.	Typical ILRC frequency vs. VDD	86
14.4.	Typical IHRC frequency deviation vs. VDD(calibrated to 16MHz)	86
14.5.	Typical ILRC Frequency vs. Temperature	87
14.6.	Typical IHRC Frequency vs. Temperature (calibrated to 16MHz).....	87
14.7.	Typical operating current vs. VDD @ system clock = ILRC/n.....	88
14.8.	Typical operating current vs. VDD @ system clock = IHRC/n	88
14.9.	Typical operating current vs. VDD @ system clock = 32KHz EOSC / n.....	89
14.10.	Typical operating current vs. VDD @ system clock = 1MHz EOSC / n	89
14.11.	Typical operating current vs. VDD @ system clock = 4MHz EOSC / n	90
14.12.	Typical IO driving current (I_{OH}) and sink current (I_{OL})	90
14.13.	Typical IO input high/low threshold voltage (V_{IH}/V_{IL})	92
14.14.	Typical resistance of IO pull high/low device.....	93
14.15.	Typical power down current (I_{PD}) and power save current (I_{PS})	94
15.	Instructions	95
15.1.	Data Transfer Instructions	96
15.2.	Arithmetic Operation Instructions	99
15.3.	Shift Operation Instructions	101
15.4.	Logic Operation Instructions	102
15.5.	Bit Operation Instructions	104
15.6.	Conditional Operation Instructions	106
15.7.	System control Instructions	107
15.8.	Summary of Instructions Execution Cycle.....	109

Revision History

Revision	Date	Description
0.05	2024/08/29	Updated description of Special Features
0.06	2025/12/10	Updated description of Section 15
0.07	2026/08/20	1. Section 14.2: Adjusted V_{DD} condition to 4.0V for $f_{SYS} = 8\text{MHz}$. 2. Section 14.2: Adjusted typical value of f_{SYS} (ILRC) to 45K.

Usage Warning

User must read all application notes of the IC by detail before using it.

Please visit the official website to download and view the latest APN information associated with it.

<http://www.padauk.com.tw/en/product/show.aspx?num=111&kw=PFC232>

(The following picture are for reference only.)

◆◆ PFC232 ◆◆

- ◆ High EFT series
- ◆ Operating temperature : $-40^{\circ}\text{C} \sim 85^{\circ}\text{C}$

Feature Documents Software & Tools **Application Note**

Content	Description	Download (CN)	Download (EN)
APN001	Output impedance of ADC analog signal source		
APN002	Over voltage protection		
APN003	Over voltage protection		
APN004	Semi-Automatic writing handler		
APN005	Effects of over voltage input to ADC		
APN007	Setting up LVR level		
APN011	Semi-Automatic writing Handler improve writing stability		
APN013	Notification of crystal oscillator		
APN017	Improve IC anti-interference ability under power plug test		
APN019	E-PAD PCB layout guideline		

1. Features

1.1. Special Features

- ◆ High EFT series
- ◆ Operating temperature range: -40°C ~ 85°C
- ◆ ESD > 8 KV

1.2. System Features

- ◆ 2KW MTP program memory for both FPPA units (programming cycle at least 1,000 times)
- ◆ 128 Bytes data RAM for both FPPA units
- ◆ One hardware 16-bit timer
- ◆ Two hardware 8-bit timers with PWM generation
- ◆ Three hardware 11-bit PWM generators (PWMG0, PWMG1 & PWMG2)
- ◆ Provide one hardware comparator
- ◆ Provide one OP Amplifier (OPA)
- ◆ Provide 1T 8x8 hardware multiplier
- ◆ 14 IO pins with optional pull-high / pull-low resistor
- ◆ Every IO pin can be configured to enable wake-up function
- ◆ For every wake-up enabled IO, two optional wake-up speed are supported: normal and fast
- ◆ Bandgap circuit to provide 1.20V reference voltage
- ◆ Up to 12-channel 12-bit resolution ADC with one channel comes from internal Bandgap reference voltage or $0.25 \times V_{DD}$
- ◆ Provide ADC reference high voltage: external input, internal V_{DD} , Bandgap(1.20V), 4V, 3V, 2V
- ◆ Clock sources: internal high RC oscillator, internal low RC oscillator and external crystal oscillator
- ◆ Built-in $V_{DD}/2$ bias voltage generator to provide maximum 5×9 dots LCD display
- ◆ 8 selectable levels of LVR reset from 1.8V to 4.5V
- ◆ Four selectable external interrupt pins

1.3. CPU Features

- ◆ Operating modes: Two processing units FPPA™ mode or Traditional one processing unit mode
- ◆ 89 powerful instructions
- ◆ Most instructions are 1T execution cycle
- ◆ Programmable stack pointer and adjustable stack level
- ◆ Direct and indirect addressing modes for data access. Data memories are available for use as an index pointer of Indirect addressing mode
- ◆ Register space, memory space and MTP space are independent

1.4. Ordering/ Package Information

- ◆ PFC232-S08: SOP8 (150mil);
 - ◆ PFC232-D08: DIP8 (300mil);
 - ◆ PFC232-S14: SOP14 (150mil);
 - ◆ PFC232-D14: DIP14 (300mil);
 - ◆ PFC232-S16: SOP16 (150mil);
 - ◆ PFC232-D16: DIP16 (300mil)
-
- Please refer to the official website file for package size information: "Package information "

2. General Description and Block Diagram

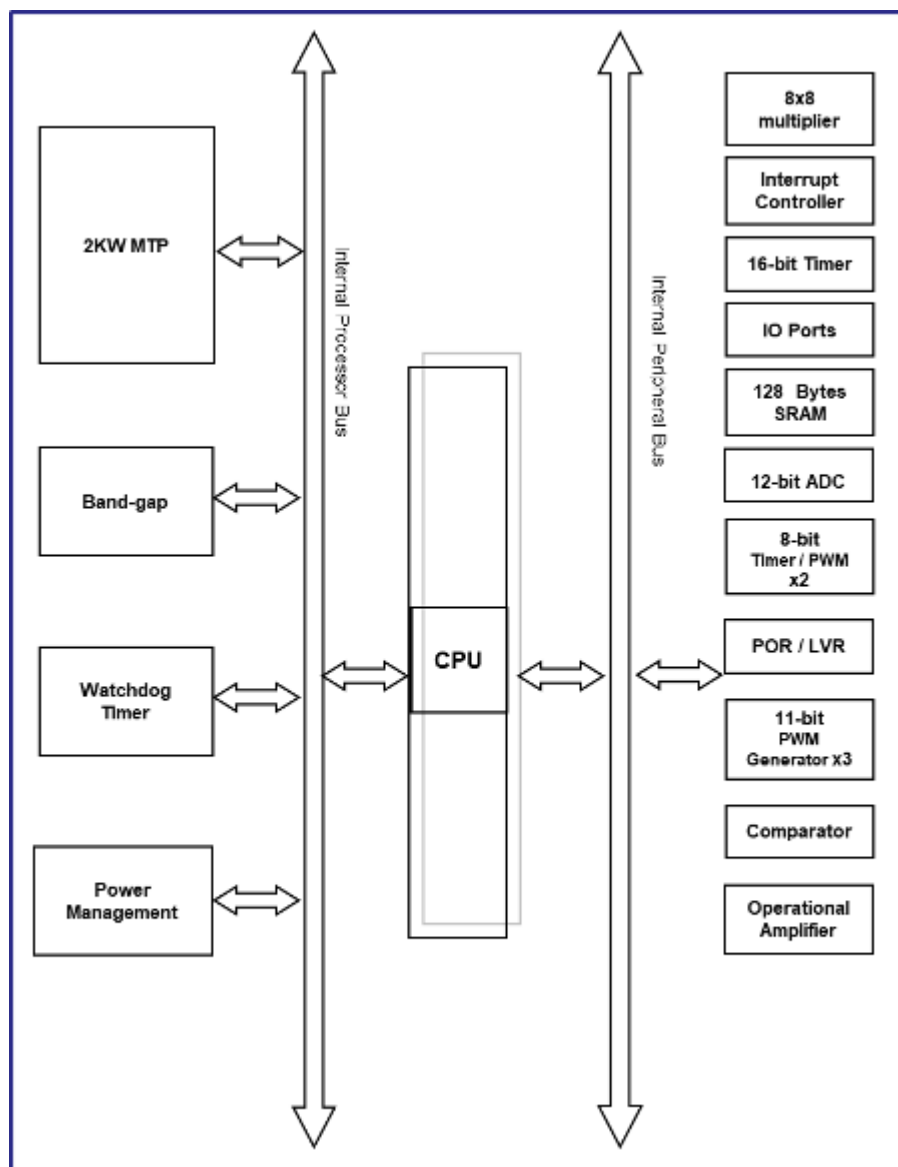
The PFC232 family is an ADC-Type of PADAUK's parallel processing, fully static, MTP-based CMOS 8 bit*2 processor array that can execute two peripheral functions in parallel. It employs RISC architecture based on patent pending FPPA™ (Field Programmable Processor Array) technology and all the instructions are executed in one cycle except that some instructions are two cycles that handle indirect memory access.

2KW MTP program memory and 128 bytes data SRAM are inside for two FPPA units using.

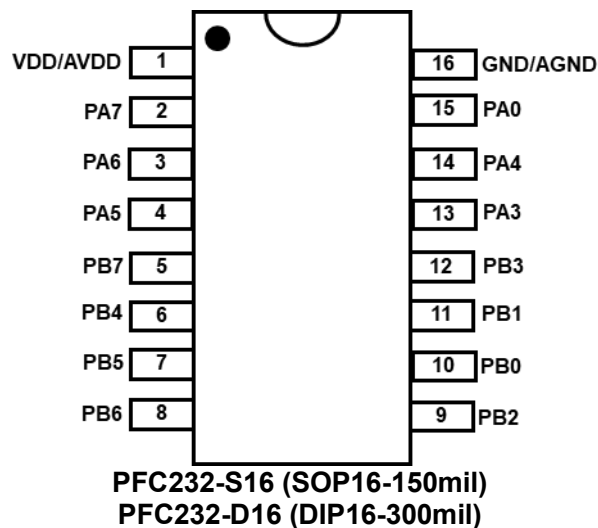
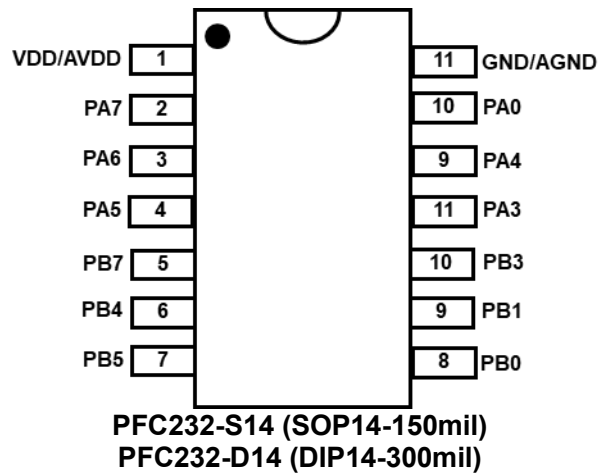
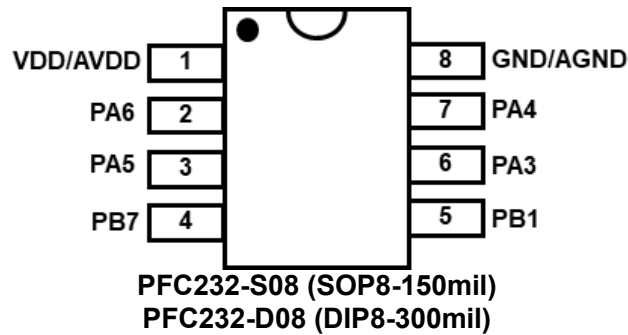
One up to 12 channels 12-bit ADC is built inside the chip with one channel for internal Bandgap reference voltage or $0.25 \times V_{DD}$.

PFC232 provides a hardware 16-bit timer, two hardware 8-bit timers with PWM generation (Timer2, Timer3) and Three hardware 11-bit timers with PWM generation (PWMG0, PWMG1, PWMG2).

PFC232 also supports an Operational Amplifier (OPA) and a hardware comparator, plus one $V_{DD}/2$ bias voltage generator for LCD display application and one 8x8 hardware multiplier to enhance hardware capability in arithmetic function.



3. Pin Definition and Functional Description



Caution: The pin assignment of PFC232 is NOT compatible with PMC232/PMS232.

Pin Name	Input / output				Special Functions								
	I / O	Pull High	Pull Low	Wake Up	Crystal	Comparator	PWM	VDD/2	ADC	OPA	External Interrupt	External Reset	Program
PA0	√	√	√	√		CO CIN- CIN+	PG0PWM	COM2	AD10	OPO	INT0		
PA3	√	√	√	√		CIN-	TM2PWM PG2PWM	COM4	AD8	OPIN-			√
PA4	√	√	√	√		CIN+ CIN-	PG1PWM	COM3	AD9	OPIN+ OPIN-	INT1A		
PA5	√	√	√	√			PG2PWM					√	√
PA6	√	√	√	√	√								√
PA7	√	√	√	√	√								
PB0	√	√	√	√				COM1	AD0		INT1		
PB1	√	√	√	√					AD1 Vref				
PB2	√	√	√	√			TM2PWM PG2PWM		AD2				
PB3	√	√	√	√			PG2PWM	COM5	AD3				
PB4	√	√	√	√			TM2PWM PG0PWM		AD4				
PB5	√	√	√	√			TM3PWM PG0PWM		AD5		INT0A		
PB6	√	√	√	√		CIN-	TM3PWM PG1PWM		AD6	OPIN-			
PB7	√	√	√	√		CIN-	TM3PWM PG1PWM		AD7	OPIN-			
VDD AVDD													√
GND AGND													√
Notice	1. All the I/O pins have: Schmitt Trigger input and CMOS voltage level. 2. IO function is automatically deactivated when a pin is used as PWM output port. 3. Please put 33Ω resistor in series to have high noise immunity when PA5 is in input mode. 4. ICE doesn't support the function PG2PWM output to PA5. 5. VDD is the IC power supply while AVDD is the Analog positive power supply. AVDD and VDD are double bonding internally and they have the same external pin. 6. GND is the IC ground pin while AGND is the Analog negative ground pin. AGND and GND are double bonding internally and they have the same external pin.												

4. Central Processing Unit (CPU)

4.1. Functional Description

There are two processing units (FPPA0 and FPPA1) inside the chip of PFC232. In each processing unit, it includes:

- its own Program Counter to control the program execution sequence
- its own Stack Pointer to store or restore the program counter for program execution
- its own accumulator
- status Flag to record the status of program execution.

Each FPPA 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, FPPA unit can execute its own program independently, thus parallel processing can be expected.

4.1.1. Two FPPA Units

These two FPPA units share the same 2K words MTP program memory, 128 bytes data SRAM and all the IO ports, these two FPPA units are operated at mutual exclusive clock cycles to avoid interference. One task switch is built inside the chip to decide which FPPA unit should be active for the corresponding cycle. The hardware diagram and basic timing diagram of FPPA units are illustrated in Fig. 1. For FPPA0 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 FPPA1 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.

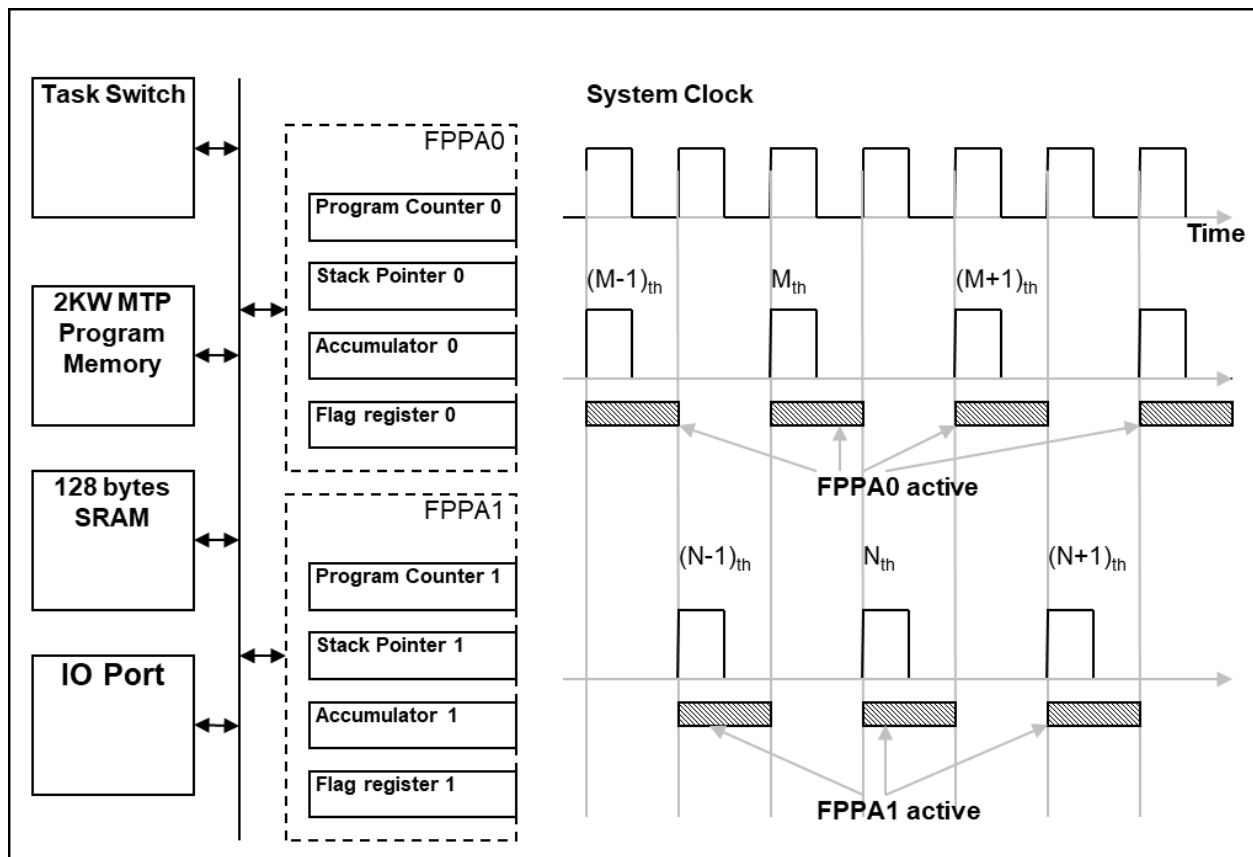


Fig. 1: Hardware and Timing Diagram of FPPA unit

Each FPPA unit has half computing power of whole system; for example, FPPA0 and FPPA1 will be operated at 4MHz if system clock is 8MHz. The FPPA unit can be enabled or disabled by programming the FPPA unit Enable Register, and only FPPA0 is enabled after power-on reset. The system initialization will be started from FPPA0 and FPPA1 unit can be enabled by user's program if necessary. Both FPPA0 and FPPA1 can be enabled or disabled by using any one FPPA unit.

4.1.2. Single FPPA Unit

For traditional MCU user who does not need parallel processing capability, PFC232 provides single FPPA mode optional to behave as traditional MCU. After single FPPA mode is selected, FPPA1 is always disabled and only FPPA0 is active. Fig.2 shows the timing diagram for each FPPA unit, FPPA1 is always disabled and only FPPA0 active.

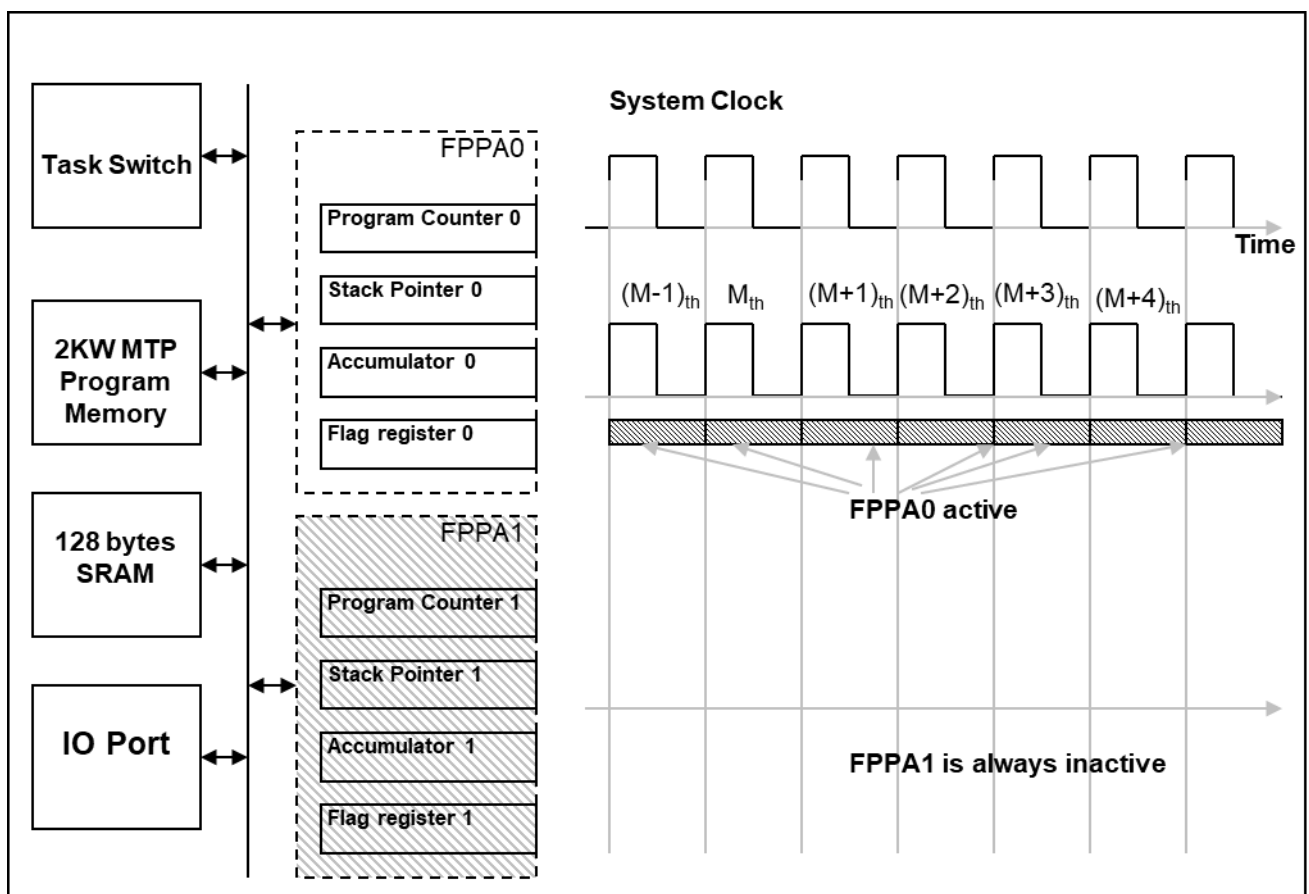


Fig. 2: Timing Diagram of single FPPA mode

4.1.3. 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 PFC232. The program counter of FPPA0 is 0 after hardware reset and 1 for FPPA1. Whenever an interrupt happens in FPPA0, the program counter will jump to 0x10 for interrupt service routine. Each FPPA unit has its own program counter to control the program execution sequence.

4.1.4. Program Structure

Program structure of two FPPA units mode

After power-up, the program starting address of FPPA0 is 0x000 and 0x001 for FPPA1. The 0x010 is the entry address of interrupt service routine, which belongs to FPPA0 only. The basic firmware structure for PFC232 is shown as Fig. 3, the program codes of two FPPA 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 FPPA0Boot will be executed first, which will include the system initialization and other FPPA units enabled.

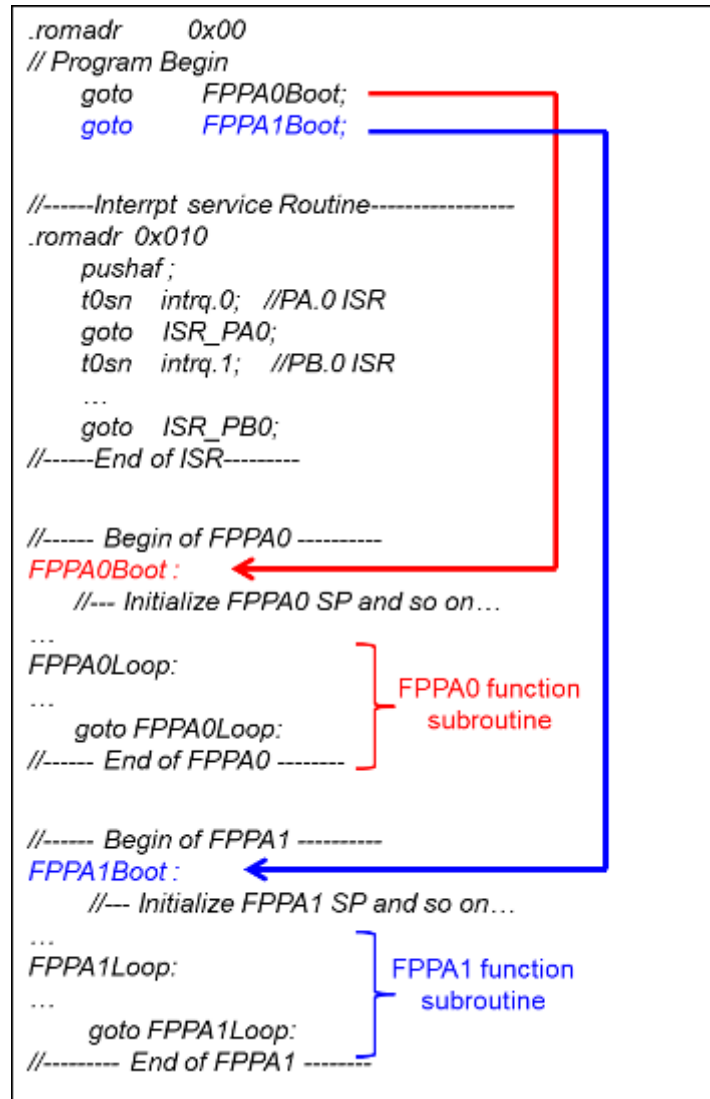


Fig. 3: Program Structure

Program structure of single FPPA mode

After power-up, the program starting address of FPPA0 is 0x000, 0x010 is the entry address of interrupt service routine. For single FPPA unit mode, the firmware structure is same as traditional MCU. After power-up, the program will be executed from address 0x000, then continuing the program sequence.

4.1.5. 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. FPPA0 and FPPA1 will share ALU for its corresponding operation.

4.2. Storage Memory

4.2.1. Program Memory – ROM

The PFC232 program memory is MTP (Multiple Time Programmable), used to store data (including: data, tables and interrupt entry) and program instructions to be executed. All program codes for each FPPA unit are stored in this MTP memory for both FPPA0 and FPPA1. The MTP program memory for PFC232 is 2K words that is partitioned as Table 1.

After reset, the initial address for FPPA0 is 0x000 and 0x001 for FPPA1. The interrupt entry is 0x010 if used and interrupt function is for FPPA0 only.

The MTP memory from address 0x7E0 to 0x7FF are for system using, address space from 0x001 to 0x00F and from 0x011 to 0x7DF are user program spaces. And the address 0x001 is the FPPA1 initial address for two FPPA units mode and user program for single FPPA unit mode.

The last 32 addresses are reserved for system using, like checksum, serial number, etc.

Address	Function
0x000	FPPA0 reset – <i>goto</i> instruction
0x001	FPPA1 reset – <i>goto</i> instruction
0x002	User program
•	•
0x00F	User program
0x010	Interrupt entry address
0x011	User program
•	•
0x7DF	User program
0x7E0	System Using
•	•
0x7FF	System Using

Table 1: Program Memory Organization

Example of Using Program Memory for Two FPPA mode

Table 2 shows one example of using program memory which two FPPA units are active.

Address	Function
0x000	FPPA0 reset – <i>goto</i> instruction (<i>goto</i> 0x020)
0x001	Begin of FPPA1 program
•	•
0x00F	<i>goto</i> 0x1A1 to continue FPPA1 program
0x010	Interrupt entry address (FPPA0 only)
•	•
0x01F	End of ISR (depend on user code size)
0x020	Begin of FPPA0 user program
•	•
0x1A0	End of FPPA0 user program
0x1A1	Continuing FPPA1 program
•	•
0x7DF	End of FPPA1 program
0x7E0	System Using
•	•
0x7FF	System Using

Table 2: Example of using Program Memory for two FPPA units mode

Example of Using Program Memory for Single FPPA mode

The entire user's program memory can be assigned to FPPA0 when single FPPA mode is selected.

Table 3 shows the example of program memory using for single FPPA unit mode.

Address	Function
0x000	FPPA0 reset
0x001	Begin of user program
0x002	user program
•	•
0x00F	<i>goto</i> instruction (<i>goto</i> 0x020)
0x010	Interrupt entry address
0x011	ISR
•	•
0x01F	End of ISR (depend on user code size)
0x020	user program
•	•
•	•
0x7DF	user program
0x7E0	System Using
•	•
0x7FF	System Using

Table 3: Example of using Program Memory for single FPPA mode

4.2.2. Data Memory – SRAM

Fig. 4 shows the SRAM data memory organization of PFC232, all the SRAM data memory could be accessed by FPPA0 and FPPA1 directly with 1T clock cycle. The access of data memory can be byte or bit operation. Besides data storage, the SRAM data memory is also served as data pointer of indirect access method and the stack memory for all FPPA 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, all the 128 bytes data memory of PFC232 can be accessed by indirect access mechanism.

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

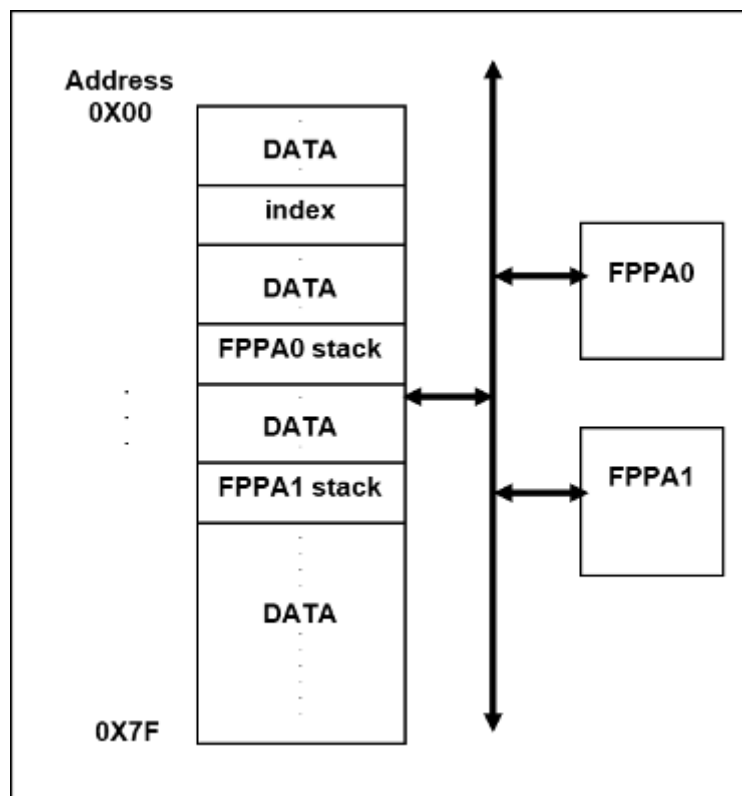


Fig. 4: Data Memory Organization

4.2.3. System Register

The register space of PFC232 is independent of SRAM space and MTP space.

The following is the PFC232 register address and brief description:

	+0	+1	+2	+3	+4	+5	+6	+7
0x00	<i>FLAG</i>	<i>FPPEN</i>	<i>SP</i>	<i>CLKMD</i>	<i>INTEN</i>	<i>INTRQ</i>	<i>T16M</i>	<i>OPAOFS</i>
0x08	<i>MULOP</i>	<i>MULRH / TM2B</i>	<i>EOSCR</i>	-	<i>INTEGS</i>	<i>PADIER</i>	<i>PBDIER</i>	-
0x10	<i>PA</i>	<i>PAC</i>	<i>PAPH</i>	<i>PAPL</i>	<i>PB</i>	<i>PBC</i>	<i>PBPH</i>	<i>PBPL</i>
0x18	<i>GPCC</i>	<i>GPCS</i>	<i>OPAC</i>	-	<i>TM2C</i>	<i>TM2CT</i>	<i>TM2S</i>	<i>MISC</i>
0x20	<i>PWMG0C</i>	<i>PWMG0S</i>	<i>PWMG0-DTH</i>	<i>PWMG0-DTL</i>	<i>PWMG0-CUBH</i>	<i>PWMG0-CUBL</i>	<i>PWMG1C</i>	<i>PWMG1S</i>
0x28	<i>PWMG1DTH</i>	<i>PWMG1DTL</i>	<i>PWMG1-CUBH</i>	<i>PWMG1-CUBL</i>	<i>PWMG2C</i>	<i>PWMG2S</i>	<i>PWMG2-DTH</i>	<i>PWMG2-DTL</i>
0x30	<i>PWMG2CUBH</i>	<i>PWMG2CUBL</i>	<i>TM3C</i>	<i>TM3CT</i>	<i>TM3S</i>	<i>ADCC</i>	<i>ADCM</i>	<i>ADCRH</i>
0x38	<i>TM3B / ADCRL</i>	<i>ADCRGC</i>	-	-	-	-	-	-

FLAG: ACC Status Flag Register

FPPEN: FPP unit Enable Register

SP: Stack Pointer Register

CLKMD: Clock Mode Register

INTEN: Interrupt Enable Register

INTRQ: Interrupt Request Register

T16M: Timer 16 mode Register

OPAOFS: OPA Offset Register

MULOP: Multiplier Operand Register

MULRH: Multiplier Result High Byte Register

TM2B / TM3B: Timer2 / Timer3 Bound Register

EOSCR: External Oscillator setting Register

INTEGS: Interrupt Edge Select Register

PADIER: Port A Digital Input Enable Register

PBDIER: Port B Digital Input Enable Register

PA: Port A Data Registers

PAC: Port A Control Registers

PAPH: Port A Pull-High Registers

PAPL: Port A Pull-Low Registers

PB: Port B Data Registers

PBC: Port B Control Registers

PBPH: Port B Pull-High Registers

PBPL: Port B Pull-Low Registers

GPCC: Comparator Control Register

GPCS: Comparator Selection Register

OPAC: OPA Control Register

TM2C / TM3C: Timer2 / Timer3 Control Register

TM2CT / TM3CT: Timer2 / Timer3 Counter Register

TM2S / TM3S: Timer2 / Timer3 Scalar Register

MISC: MISC Register

PWMG0C / PWMG1C / PWMG2C:

PWMG0 / PWMG1 / PWMG2 control Register

PWMG0S / PWMG1S / PWMG2S:

PWMG0 / PWMG1 / PWMG2 Scalar Register

PWMG0DTH / PWMG1DTH / PWMG2DTH:

PWMG0 / PWMG1 / PWMG2 Duty Value High Register

PWMG0DTL / PWMG1DTL / PWMG2DTL:

PWMG0 / PWMG1 / PWMG2 Duty Value Low Register

PWMG0CUBH / PWMG1CUBH / PWMG2CUBH:

PWMG0 / PWMG1 / PWMG2 Counter Upper Bound High Register

PWMG0CUBL / PWMG1CUBL / PWMG2CUBL:

PWMG0 / PWMG1 / PWMG2 Counter Upper Bound Low Register

ADCC: ADC Control Register

ADCM: ADC Mode Register

ADCRH / ADCRL: ADC Result High Low Register

ADCRGC: ADC Regulator Control Register

4.2.3.1.ACC Status Flag Register (*FLAG*), address = 0x00

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

4.2.3.2.FPPA unit Enable Register (*FPPEN*), address = 0x01

Bit	Reset	R/W	Description
7 – 2	-	-	Reserved. Please keep 0 for future compatibility.
1	0	R/W	FPPA1 enable. This bit is used to enable FPPA1. 0 / 1: disable / enable
0	1	R/W	FPPA0 enable. This bit is used to enable FPPA0. 0 / 1: disable / enable

4.2.3.3.MISC Register (*MISC*), address = 0x1F

Bit	Reset	R/W	Description
7 – 6	-	-	Reserved. (keep 0 for future compatibility)
5	0	WO	Enable fast Wake up. Fast wake-up is NOT supported when EOSC is enabled. 0: Normal wake up. The wake-up time is 3000 ILRC clocks (Not for fast boot-up) 1: Fast wake up. The wake-up time is 45 ILRC clocks.
4	0	WO	Enable VDD/2 bias voltage generator 0 / 1 : Disable / Enable (ICE cannot be dynamically switched)
3	-	-	Reserved.
2	0	WO	Disable LVR function. 0 / 1 : Enable / Disable
1 – 0	00	WO	Watch dog time out period 00: 8k ILRC clock period 01: 16k ILRC clock period 10: 64k ILRC clock period 11: 256k ILRC clock period

4.3. The Stack

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 register address 0x02. The bit number of stack pointer is 8 bit; the stack memory cannot be accessed over 128 bytes and should be defined within 128 bytes from 0x00 address. The stack memory of PFC232 for each FPPA unit can be assigned by user via stack pointer register, means that the depth of stack pointer for each FPPA 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
  ...
```

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. 5 shows that the status of stack before FPPA0 execution, system has calculated the required stack space and has reserved for the program.

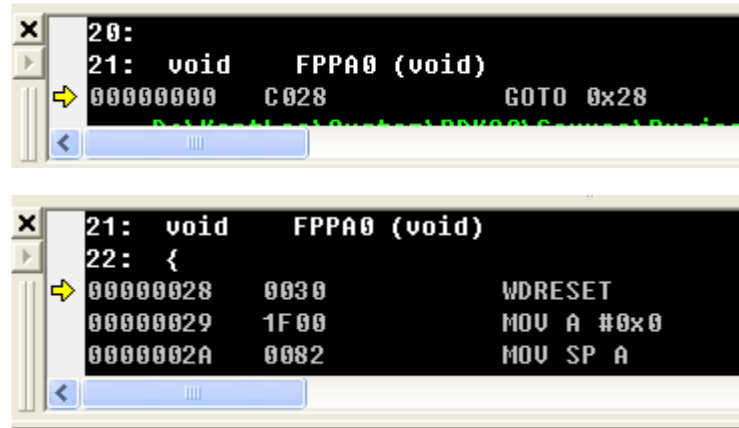


Fig. 5: Stack Assignment in Mini-C project

4.3.1. Stack Pointer Register (SP), address = 0x02

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

4.4. Code Options

Option	Selection	Description
Security	Enable	MTP content is protected 7/8 words
	Disable(default)	MTP content is not protected so program can be read back
LVR	4.0V	Select LVR = 4.0V
	3.75V	Select LVR = 3.75V
	3.0V	Select LVR = 3.0V
	2.7V	Select LVR = 2.7V
	2.5V(default)	Select LVR = 2.5V
	2.2V	Select LVR = 2.2V
	2.0V	Select LVR = 2.0V
	1.8V	Select LVR = 1.8V
GPC_P_In	PA.4(default)	Comparator plus input is from PA.4
	PA.0	Comparator plus input is from PA.0
GPC_PWM	Disable(default)	GPC / PWM are independent
	Enable	GPC output control PWM output (ICE does NOT Support.)

Option	Selection	Description
OPA_PWM	Disable(default)	OPA / PWM are independent
	Enable	OPA output control PWM output (ICE does NOT Support.)
Comparator_Edge	All_Edge(default)	The comparator will trigger an interrupt on the rising edge or falling edge
	Rising_Edge	The comparator will trigger an interrupt on the rising edge
	Falling_Edge	The comparator will trigger an interrupt on the falling edge
Interrupt Src0	PA.0(default)	<i>INTEN/ INTRQ</i> .Bit0 is from PA.0
	PB.5	<i>INTEN/ INTRQ</i> .Bit0 is from PB.5
Interrupt Src1	PB.0(default)	<i>INTEN/ INTRQ</i> .Bit1 is from PB.0
	PA.4	<i>INTEN/ INTRQ</i> .Bit1 is from PA.4
PB4_PB7_Drive	Normal	PB4 & PB7 Drive / Sink Current is Normal
	Strong(default)	PB4 & PB7 Drive / Sink Current is Strong
PWM_Source	16MHZ(default)	When <i>PWMG0C.0</i> = 1, PWMG0 clock source = IHRC = 16MHZ When <i>PWMG1C.0</i> = 1, PWMG1 clock source = IHRC = 16MHZ When <i>PWMG2C.0</i> = 1, PWMG2 clock source = IHRC = 16MHZ
	32MHZ	When <i>PWMG0C.0</i> = 1, PWMG0 clock source = IHRC*2 = 32MHZ When <i>PWMG1C.0</i> = 1, PWMG1 clock source = IHRC*2 = 32MHZ When <i>PWMG2C.0</i> = 1, PWMG2 clock source = IHRC*2 = 32MHZ (ICE does NOT Support.)
TMx_Source	16MHZ(default)	When <i>TM2C[7:4]</i> = 0010, TM2 clock source = IHRC = 16MHZ When <i>TM3C[7:4]</i> = 0010, TM3 clock source = IHRC = 16MHZ
	32MHZ	When <i>TM2C[7:4]</i> = 0010, TM2 clock source = IHRC*2 = 32MHZ When <i>TM3C[7:4]</i> = 0010, TM3 clock source = IHRC*2 = 32MHZ (ICE does NOT Support.)
TMx_Bit	6 Bit(default)	When <i>TM2S.7</i> =1, TM2 PWM resolution is 6 Bit When <i>TM3S.7</i> =1, TM3 PWM resolution is 6 Bit
	7 Bit	When <i>TM2S.7</i> =1, TM2 PWM resolution is 7 Bit When <i>TM3S.7</i> =1, TM3 PWM resolution is 7 Bit (ICE does NOT Support.)
EMI	Disable(default)	Disable EMI optimize option
	Enable	The system clock will be slightly vibrated for better EMI performance
FPPA	1-FPPA(default)	Single FPPA unit mode
	2-FPPA	Two FPPA units mode

5. Oscillator and System Clock

There are three oscillator circuits provided by PFC232: external crystal oscillator (EOSC), internal high RC oscillator (IHRC) and internal low RC oscillator (ILRC).

These three oscillators are enabled or disabled by registers *EOSCR.7*, *CLKMD.4* and *CLKMD.2* independently. User can choose one of these three oscillators as system clock source and use *CLKMD* register to target the desired frequency as system clock to meet different applications.

Oscillator Module	Enable / Disable
EOSC	<i>EOSCR.7</i>
IHRC	<i>CLKMD.4</i>
ILRC	<i>CLKMD.2</i>

Table 4: Three Oscillator Circuits provided by PFC232

5.1. Internal High RC Oscillator and Internal Low RC Oscillator

The frequency of IHRC / ILRC will vary by process, supply voltage and temperature. Please refer to the measurement chart for IHRC / ILRC frequency verse V_{DD} and IHRC / ILRC frequency verse temperature.

The PFC232 writer tool provides IHRC frequency calibration (usually up to 16MHz) to eliminate frequency drift caused by factory production. ILRC has no calibration operation. For applications that require accurate timing, please do not use the ILRC clock as a reference time.

5.2. External Crystal Oscillator

The range of operating frequency of crystal oscillator can be from 32 KHz to 4MHz, depending on the crystal placed on; higher frequency oscillator than 4MHz is NOT supported. Fig. 6 shows the hardware connection under this application.

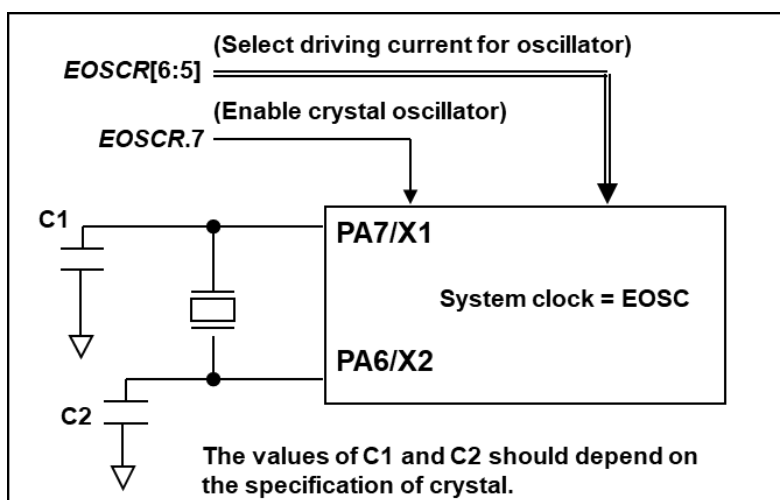


Fig. 6: Connection of crystal oscillator

5.2.1. External Oscillator Setting Register (*EOSCR*), address = 0x0A

Bit	Reset	R/W	Description
7	0	WO	Enable external crystal oscillator. 0 / 1 : Disable / Enable
6 – 5	00	WO	External crystal oscillator selection. 00 : reserved 01 : Low driving capability, for lower frequency, ex: 32KHz crystal oscillator 10 : Middle driving capability, for middle frequency, ex: 1MHz crystal oscillator 11 : High driving capability, for higher frequency, ex: 4MHz crystal oscillator
4 – 0	-	-	Reserved. Please keep 0 for future compatibility.

5.2.2. Usages and Precautions of External Oscillator

Besides crystal, external capacitor and options of PFC232 should be fine-tuned in *EOSCR* register to have good sinusoidal waveform. The *EOSCR.7* is used to enable crystal oscillator module. *EOSCR.6* and *EOSCR.5* are used to set the different driving current to meet the requirement of different frequency of crystal oscillator.

Table 5 shows the recommended values of C1 and C2 for different crystal oscillator; the measured start-up time under its corresponding conditions is also shown. Since the crystal or resonator had its own characteristic, the capacitors and start-up time may be slightly different for different type of crystal or resonator, please refer to its specification for proper values of C1 and C2.

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

Table 5: Recommend values of C1 and C2 for crystal and resonator oscillators

Configuration of PA7 and PA6 when using crystal oscillator:

- (1) PA7 and PA6 are set as input;
- (2) PA7 and PA6 internal pull-high resistors are set to close;
- (3) Set PA6 and PA7 as analog inputs with the *PADIER* register to prevent power leakage.

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

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

```
void  FPPA0 (void)
{
    . ADJUST_IC  SYSCLK=IHRC/16, IHRC=16MHz, VDD=5V
    ...
    $ EOSCR Enable, 4Mhz;           // EOSCR = 0b111_00000;
    $ T16M  EOSC, /1, BIT13; // while T16.Bit13 0 => 1, Intrq.T16 => 1
                                // suppose crystal eosc. is stable

    WORD count = 0;
    stt16      count;
    Intrq.T16  = 0;
    while (! Intrq.T16) NULL;      // count from 0x0000 to 0x2000, then trigger INTRQ.T16
    CLKMD = 0xB4;                  // switch system clock to EOSC;
    CLKMD.4 = 0;                   // disable IHRC
    ...
}
```

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

5.3. System Clock and IHRC Calibration

5.3.1. System Clock

The clock source of system clock comes from IHRC, ILRC or EOSC, the hardware diagram of system clock in the PFC232 is shown as Fig. 7.

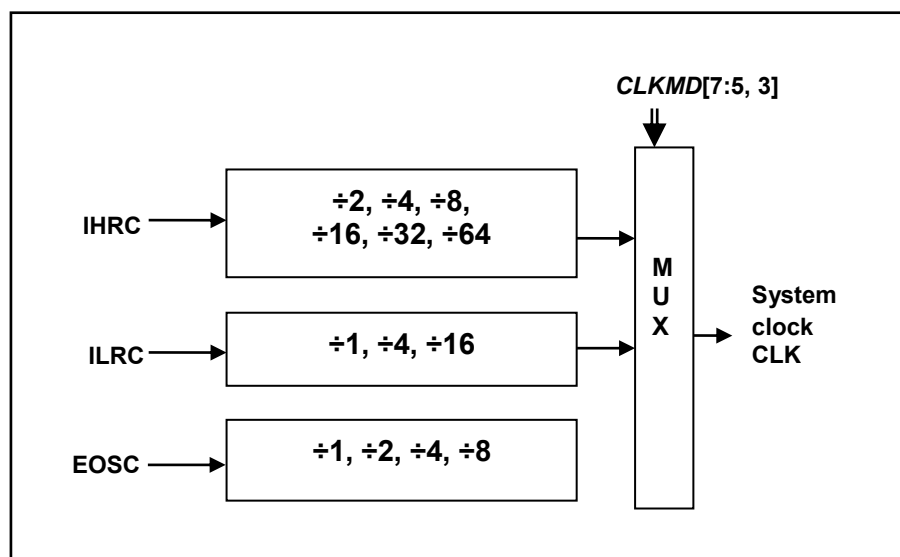


Fig. 7: Options of System Clock

5.3.1.1. Clock Mode Register (CLKMD), address = 0x03

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

5.3.2. Frequency Calibration

The IHRC frequency and Bandgap reference voltage may be different chip by chip due to manufacturing variation, PFC232 provide the IHRC frequency calibration to eliminate this variation, and this function can be selected when compiling user's program and the command will be inserted into user's program automatically.

The calibration command is shown as below:

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

Where,

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

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

p3=2.2 ~ 5.5; In order to calibrate the chip under different supply voltage.

Usually, .ADJUST_IC will be the first command after boot up, in order to set the target operating frequency whenever starting the system. The program code for IHRC frequency calibration is executed only one time that occurs in writing the codes into MTP memory; after then, it will not be executed again.

If the different option for IHRC calibration is chosen, the system status is also different after boot.

As shown in table 6:

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

Table 6: Options for IHRC Frequency Calibration

The following shows the different states of PFC232 under different options:

(1) .ADJUST_IC SYSClk=IHRC/4, IHRC=16MHz, V_{DD}=5V

After boot, CLKMD = 0x14:

- IHRC frequency is calibrated to 16MHz@V_{DD}=5V and IHRC module is enabled
- System CLK = IHRC/4 = 4MHz
- Watchdog timer is disabled, ILRC is enabled, PA5 is in input mode

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

After boot, CLKMD = 0x3C:

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

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

After boot, CLKMD = 0xE4:

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

(4) .ADJUST_IC DISABLE

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

- IHRC is not calibrated.
- System CLK = ILRC or IHRC/64
- Watchdog timer is enabled, ILRC is enabled, PA5 is in input mode

5.3.2.1. Special Statement

- (1) The IHRC frequency calibration is performed when IC is programmed by the writer.
- (2) Because the characteristic of the Epoxy Molding Compound (EMC) would some degrees affects the IHRC frequency (either for package or COB), if the calibration is done before molding process, the actual IHRC frequency after molding may be deviated or becomes out of spec. Normally, the frequency is getting slower a bit.
- (3) It usually happens in COB package or Quick Turnover Programming (QTP). And PADAUK would not take any responsibility for this situation.
- (4) Users can make some compensatory adjustments according to their own experiences. For example, users can set IHRC frequency to be 0.5% ~ 1% higher and aim to get better re-targeting after molding.

5.3.3. System Clock Switching

After IHRC calibration, the system clock of PFC232 can be switched among IHRC, ILRC and EOSC by setting the *CLKMD* register at any time, **but please notice that the original clock module can NOT be turned off at the same time as writing command to *CLKMD* register**. For example, when switching from A clock source to B clock source, you should first switch the system clock source to B and then close the A clock source. 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/4

```
...                               // system clock is ILRC
CLKMD.4      = 1;                // turn on IHRC first to improve anti-interference ability
CLKMD        = 0x14;            // switch to IHRC/4, ILRC CAN NOT be disabled here
// CLKMD.2 = 0;                // if need, ILRC CAN be disabled at this time
...
```

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

```
...                               // system clock is IHRC/4
CLKMD        = 0xB0;            // switch to EOSC, IHRC CAN NOT be disabled here
CLKMD.4      = 0;              // IHRC CAN be disabled at this time
...
```

Case 3: Switching system clock from IHRC/8 to IHRC/4

```
...                               // system clock is IHRC/8, ILRC is enabled here
CLKMD        = 0x14;            // switch to IHRC/4
...
```

Case 4: System may hang if it is to switch clock and turn off original oscillator at the same time

```
...                               // system clock is ILRC
CLKMD        = 0x10;            // CAN NOT switch clock from ILRC to IHRC/4 and turn off
                                // ILRC oscillator at the same time
...
```

6. Reset

PFC232 reset can be caused by four factors: power-on reset, LVR reset, watchdog timeout overflow reset, and PRSTB pin reset. After the reset, the system will restart. The program counter will jump to address 0x000 and most of all the registers in PFC232 will be set to the default values.

6.1. Power On Reset - POR

POR (Power-On-Reset) is used to reset PFC232 when power up. Time for boot-up is about 3000 ILRC clock cycles. The power up sequence is shown in the Fig. 8. Customer must ensure the stability of supply voltage after power up no matter which option is chosen.

The PFC232 data memory is in an uncertain state when the power on reset occurs.

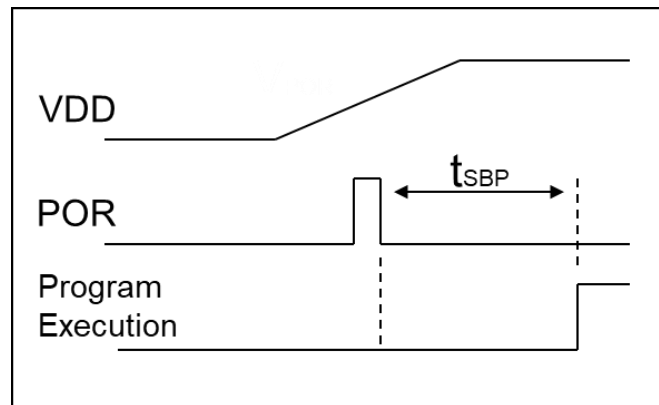


Fig. 8: Power Up Sequence

Fig. 9 shows the typical program flow after boot up. Please notice that the FPPA1 is disabled after reset and recommend NOT enabling FPPA1 before system and FPPA0 initialization.

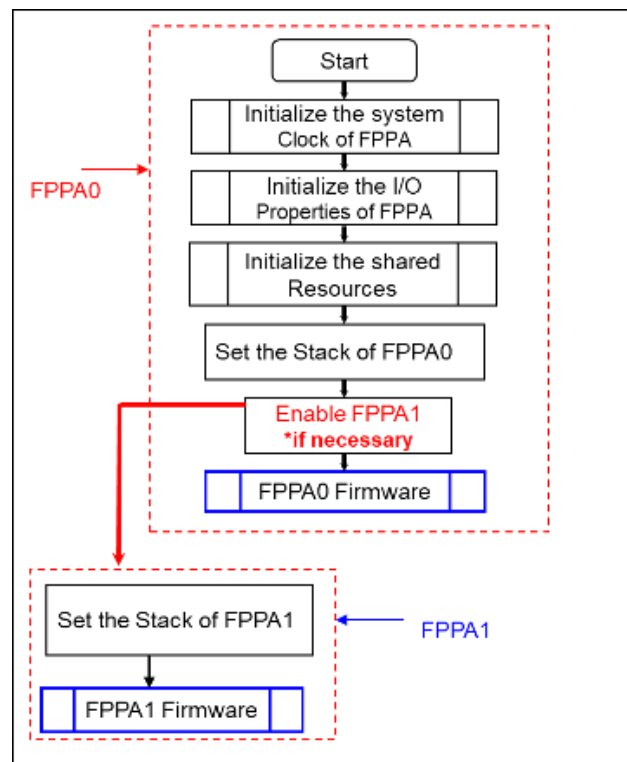


Fig. 9: Boot Procedure

6.2. Low Voltage Reset - LVR

If VDD drops below the Voltage level of LVR (Low Voltage Reset), LVR Reset will occur in the system. The LVR reset timing diagram is shown in figure 10. After LVR reset, the SRAM data will be kept when $VDD > V_{DR}$ (SRAM data retention voltage). However, if SRAM is cleared after power-on again, the data cannot be kept, the data memory is in an uncertain state when $VDD < V_{DR}$.

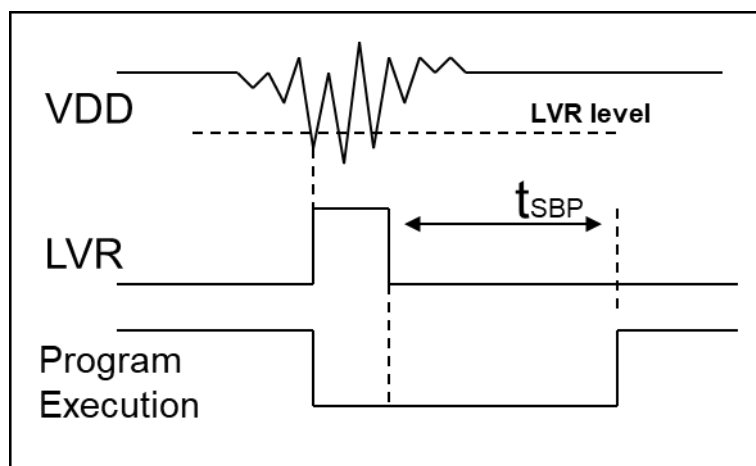


Fig. 10: Low Voltage Reset Sequence

LVR level selection is done at compile time. User must select LVR based on the system working frequency and power supply voltage to make the MCU work stably.

The following are Suggestions for setting operating frequency, power supply voltage and LVR level:

SYSCLK	VDD	LVR
8MHz	$\geq 3.75V$	3.75V
4MHz	$\geq 2.5V$	2.5V
2MHz	$\geq 2.2V$	2.2V

Table 7: LVR setting for reference

- (1) The setting of LVR (1.8V ~ 4.0V) will be valid just after successful power-on process.
- (2) User can set MISC.2 as "1" to disable LVR. However, VDD must be kept as exceeding the lowest working voltage of chip; Otherwise IC may work abnormally.
- (3) The LVR function will be invalid when IC in stopexe or stopsys mode.

MISC Register (<i>MISC</i>), address = 0x08			
Bit	Reset	R/W	Description
7 – 6	-	-	Reserved. (keep 0 for future compatibility)
5	0	WO	Enable fast Wake-up.
4	0	WO	Enable VDD/2 bias voltage generator 0 / 1 : Disable / Enable
3	-	-	Reserved. (keep 0 for future compatibility)
2	0	WO	Disable LVR function. 0 / 1 : Enable / Disable
1 – 0	00	WO	Watch dog time out period 00: 8k ILRC clock period 01: 16k ILRC clock period 10: 64k ILRC clock period 11: 256k ILRC clock period

6.3. Watch Dog Timeout Reset

The watchdog timer (WDT) is a counter with clock coming from ILRC, so it will be invalid if ILRC is off. The frequency of ILRC may drift a lot due to the variation of manufacture, supply voltage and temperature. User should reserve guard band for safe operation.

Besides, the watchdog period will also be shorter than expected after Reset or Wakeup events. It is suggested to clear WDT by *wdreset* command after these events to ensure enough clock periods before WDT timeout.

To ensure the watchdog is cleared before the timeout overflow, the instruction *wdreset* can be used to clear the WDT within a safe time. WDT can be cleared by power-on-reset or by command *wdreset* at any time.

When WDT is timeout, PFC232 will be reset to restart the program execution. The relative timing diagram of watchdog timer is shown as Fig. 11.

The PFC232 data memory will be reserved when the WDT reset occurs.

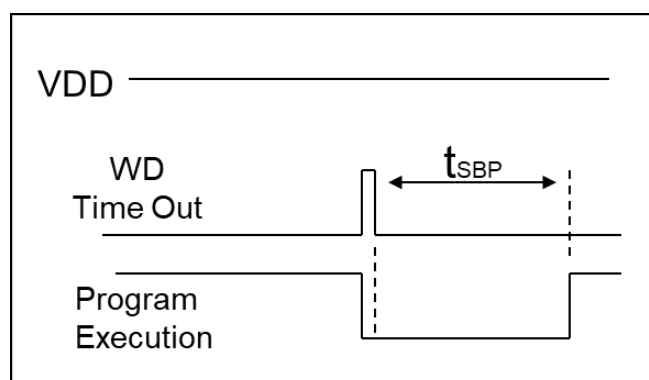


Fig. 11: Sequence of Watch Dog Timeout reset

There are four different timeout periods of watchdog timer can be chosen by setting the *MISC*[1:0]. And watchdog timer can be disabled by *CLKMD*.1.

Clock Mode Register (<i>CLKMD</i>), address = 0x03			
Bit	Reset	R/W	Description
7 – 5	111	R/W	System clock selection
4	1	R/W	IHRC oscillator Enable. 0 / 1: disable / enable
3	0	R/W	Clock Type Select.
2	1	R/W	ILRC Enable. 0 / 1: disable / enable If ILRC is disabled, watchdog timer is also disabled.
1	1	R/W	Watch Dog Enable. 0 / 1: disable / enable
0	0	R/W	Pin PA5/PRSTB function. 0 / 1: PA5 / PRSTB

6.4. External Reset Pin - PRSTB

The PFC232 supports external reset and its external reset pin shares the same IO port with PA5. Using external reset function requires:

- (1) Set PA5 as input;
- (2) Set *CLKMD*.0 = 1 to make PA5 as the external PRSTB input pin.

When the PRSTB pin is high, the system is in normal working state. Once the reset pin detects a low level, the system will be reset. The timing diagram of PRSTB reset is shown in figure 6.

The PFC232 data memory will be reserved when the PRSTB reset occurs.

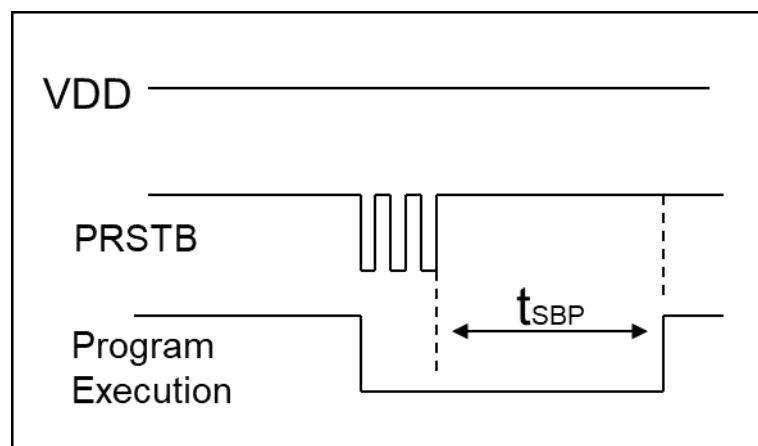


Fig. 12: Sequence of PRSTB reset

7. System Operating Mode

There are three operational modes defined by hardware:

- (1) ON mode
- (2) Power-Save mode
- (3) Power-Down mode

ON mode is the state of normal operation with all functions ON.

Power-Save mode (*stopexe*) is the state to reduce operating current and CPU keeps ready to continue.

Power-Down mode (*stopsys*) is used to save power deeply.

Therefore, Power-Save mode is used in the system which needs low operating power with wake-up periodically and Power-Down mode is used in the system which needs power down deeply with seldom wake-up.

7.1. Power-Save Mode (“*stopexe*”)

Using *stopexe* instruction to enter the Power-Save mode, only system clock is disabled, remaining all the oscillator modules be active. So only the CPU stops executing instructions. Wake-up from input pins can be considered as a continuation of normal execution.

The detail information for Power-Save mode shown below:

- (1) IHRC and oscillator modules: No change, keep active if it was enabled
- (2) ILRC oscillator modules: must remain enabled, need to start with ILRC when waking up
- (3) System clock: Disable, therefore, CPU stops execution
- (4) MTP memory is turned off.
- (5) Timer counter: Stop counting if its clock source is system clock or the corresponding oscillator module is disabled; otherwise, it keeps counting. (The Timer contains TM16, TM2, TM3, PWMG0, PWMG1, PWMG2.)
- (6) Wake-up sources:
 - a. IO toggle wake-up: IO toggling in digital input mode (*PxC* bit is 1 and *PxDIER* bit is 1)
 - b. Timer wake-up: If the clock source of Timer is not the SYSCLK, the system will be awakened when the Timer counter reaches the set value.
 - c. Comparator wake-up: It need setting *GPCC.7*=1 and *GPCS.6*=1 to enable the comparator wake-up function at the same time. Please note: the internal 1.20V bandgap reference voltage is not suitable for the comparator wake-up function.

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

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

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

7.2. Power-Down Mode (“*stopsys*”)

Power-Down mode is the state of deeply power-saving with turning off all the oscillator modules. By using the *stopsys* instruction, this chip will be put on Power-Down mode directly. It is recommending to set *GPCC.7*=0 to disable the comparator before the command *stopsys*.

Wake-up from input pins can be considered as a continuation of normal execution. To minimize power consumption, all the I/O pins should be carefully manipulated before entering Power-Down mode.

The following shows the internal status of PFC232 in detail when *stopsys* command is issued:

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

The reference sample program for power down mode is shown as below:

```

CMKMD =  0xF4;      //  Change clock from IHRC to ILRC, disable watchdog timer
CLKMD.4  =  0;      //  disable IHRC
...
while (1)
{
    STOPSYS;          //  enter Power-Down mode
    if (...) break;    //  if wake-up happen and check OK, then return to high speed,
                        //  else stay in Power-Down mode again.
}
CLKMD =  0x14;      //  Change clock from ILRC to IHRC/4

```

7.3. Wake-Up

After entering the Power-Down or Power-Save modes, the PFC232 can be resumed to normal operation by toggling IO pins. Wake-up from timer are available for Power-Save mode ONLY. Table 8 shows the differences in wake-up sources between *stopsys* and *stopexe*.

Differences in wake-up sources between <i>stopsys</i> and <i>stopexe</i>			
	IO Toggle	Timer wake-up	Comparator wake-up
<i>stopsys</i>	Yes	No	No
<i>stopexe</i>	Yes	Yes	Yes

Table 8: Differences in wake-up sources between Power-Save mode and Power-Down mode

When using the IO pins to wake-up the PFC232, registers *PxDIER* should be properly set to enable the wake-up function for every corresponding pin. The time for normal wake-up is about 3000 ILRC clocks counting from wake-up event; fast wake-up can be selected to reduce the wake-up time by *MISC* register, and the time for fast wake-up is about 45 ILRC clocks from IO toggling.

Suspend mode	Wake-up mode	Wake-up time (t_{WUP}) from IO toggle
<i>stopexe</i> suspend or <i>stopsys</i> suspend	Fast wake-up	$45 * T_{ILRC}$, Where T_{ILRC} is the time period of ILRC
<i>stopexe</i> suspend or <i>stopsys</i> suspend	Normal wake-up	$3000 * T_{ILRC}$, Where T_{ILRC} is the clock period of ILRC

8. Interrupt

There are eight interrupt lines for PFC232:

- ◆ External interrupt PA0/PB5
- ◆ Timer16 interrupt
- ◆ External interrupt PB0/PA4
- ◆ Timer2 interrupt
- ◆ ADC interrupt
- ◆ Timer3 interrupt
- ◆ GPC interrupt
- ◆ PWMG0 interrupt

Every interrupt request line to CPU has its own corresponding interrupt control bit to enable or disable it. The hardware diagram of interrupt controller is shown as Fig. 13. All the interrupt request flags are set by hardware and cleared by writing *INTRQ* register. When the request flags are set, it can be rising edge, falling edge or both, depending on the setting of register *INTEGS*. All the interrupt request lines are also controlled by *engint* instruction (enable global interrupt) to enable interrupt operation and *disgint* instruction (disable global interrupt) to disable it.

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

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

Note: the external interrupt source can be switched through *Interrupt Src0* or *Interrupt Src1* in the Code Option.

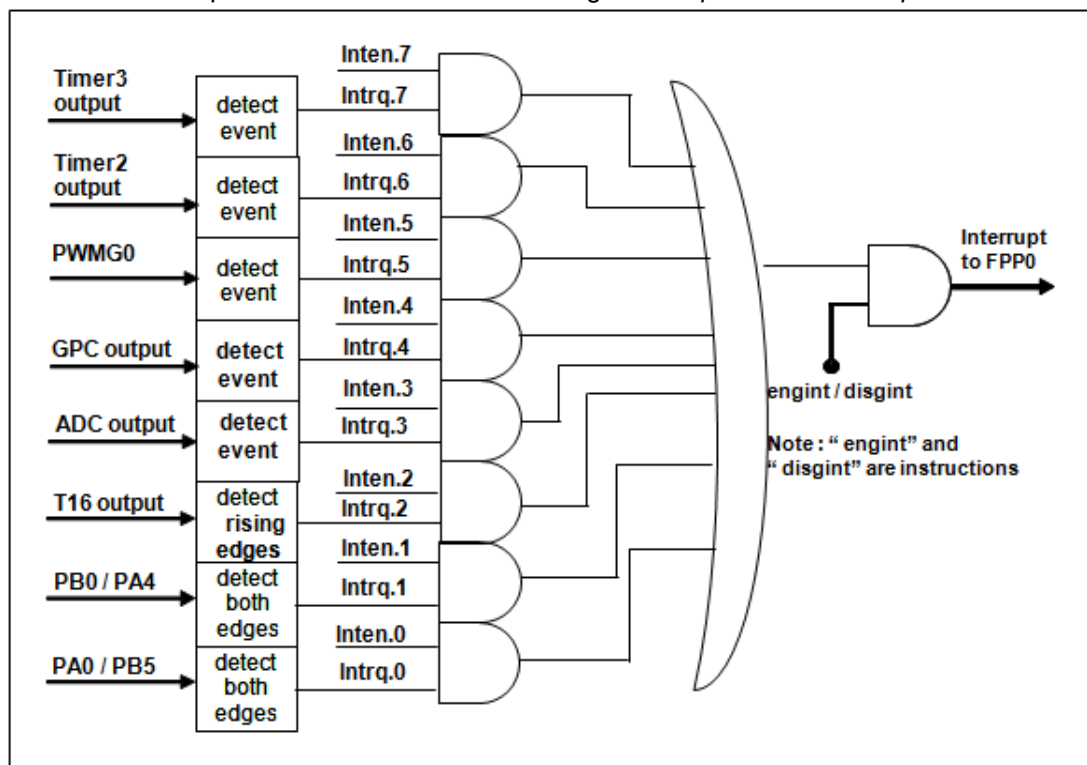


Fig. 13: Hardware diagram of Interrupt controller

8.1. Interrupt Enable Register (*INTEN*), address = 0x04

Bit	Reset	R/W	Description
7	0	R/W	Enable interrupt from Timer3. 0 / 1: disable / enable.
6	0	R/W	Enable interrupt from Timer2. 0 / 1: disable / enable.
5	0	R/W	Enable interrupt from PWMG0. 0 / 1: disable / enable.
4	0	R/W	Enable interrupt from comparator. 0 / 1: disable / enable.
3	0	R/W	Enable interrupt from ADC. 0 / 1: disable / enable.
2	0	R/W	Enable interrupt from Timer16 overflow. 0 / 1: disable / enable.
1	0	R/W	Enable interrupt from PB0/PA4. 0 / 1: disable / enable. (by code option Interrupt Src1)
0	0	R/W	Enable interrupt from PA0/PB5. 0 / 1: disable / enable. (by code option Interrupt Src0)

8.2. Interrupt Request Register (*INTRQ*), address = 0x05

Bit	Reset	R/W	Description
7	-	R/W	Interrupt Request from Timer3, this bit is set by hardware and cleared by software. 0 / 1: No request / Request
6	-	R/W	Interrupt Request from Timer2, this bit is set by hardware and cleared by software. 0 / 1: No request / Request
5	-	R/W	Interrupt Request from PWMG0, this bit is set by hardware and cleared by software. 0 / 1: No request / Request
4	-	R/W	Interrupt Request from comparator, this bit is set by hardware and cleared by software. 0 / 1: No request / Request
3	-	R/W	Interrupt Request from ADC, this bit is set by hardware and cleared by software. 0 / 1: No request / Request
2	-	R/W	Interrupt Request from Timer16, this bit is set by hardware and cleared by software. 0 / 1: No request / Request
1	-	R/W	Interrupt Request from pin PB0/PA4, this bit is set by hardware and cleared by software. 0 / 1: No request / Request
0	-	R/W	Interrupt Request from pin PA0/PB5, this bit is set by hardware and cleared by software. 0 / 1: No Request / request

8.3. Interrupt Edge Select Register (*INTEGS*), address = 0x0C

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

Note:

- (1) *INTEN* and *INTRQ* have no initial values. Please set required value before enabling interrupt function. Even if *INTEN*=0, *INTRQ* will be still triggered by the interrupt source.
- (2) PA4 and PB5 can be used as external interrupt pins. When using the PA4 as external interrupt pin, the setting method of *INTEN* / *INTRQ* / *INTEGS* registers are same as that of PB0, the only difference is to choose PB0 or PA4 as source of interrupt_Src1 in CODE_OPTION. Similarly, when using the PB5 as external interrupt pin, the setting method of *INTEN* / *INTRQ* / *INTEGS* registers are same as that of PA0, the only difference is to choose PA0 or PB5 as source of interrupt_Src0 in CODE_OPTION.

8.4. Interrupt Work Flow

Once the interrupt occurs, its operation will be:

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

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

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

8.5. General Steps to Interrupt

When using the interrupt function, the procedure should be:

Step1: Set *INTEN* register, enable the interrupt control bit.

Step2: Clear *INTRQ* register.

Step3: In the main program, using *engint* to enable CPU interrupt function.

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

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

When interrupt service routine starts, use *pushaf* instruction to save *ALU* and *FLAG* register. *Popaf* instruction is to restore *ALU* and *FLAG* register before *reti* as below:

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

* Use *disgint* in the main program can disable all interrupts.

8.6. Example for Using Interrupt

User must reserve enough stack memory for interrupt, two bytes stack memory for one level interrupt and four bytes for two levels interrupt.

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

```

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

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

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

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

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

    POPAF             // restore ALU and FLAG register
}

```

9. I/O Port

9.1. IO Related Registers

9.1.1. Port A Digital Input Enable Register (*PADIER*), address = 0x0D

Bit	Reset	R/W	Description
7 - 6	11	WO	Enable PA7~PA6 digital input and wake-up event. 1 / 0 : enable / disable. These bits should be set to low to prevent leakage current when external crystal oscillator is used.
5	1	WO	Enable PA5 digital input and wake-up event. 1 / 0 : enable / disable.
4	1	WO	Enable PA4 digital input and wake-up event and interrupt request. 1 / 0 : enable / disable.
3	1	WO	Enable PA3 digital input and wake-up event. 1 / 0 : enable / disable.
2 - 1	-	WO	Reserved. (Please keep 00 for future compatibility)
0	1	WO	Enable PA0 digital input and wake-up event and interrupt request. 1 / 0: enable / disable.

9.1.2. Port B Digital Input Enable Register (*PBDIER*), address = 0x0E

Bit	Reset	R/W	Description
7 - 6	11	WO	Enable PB7~PB6 digital input and wake up event. 1 / 0: enable / disable.
5	1	WO	Enable PB5 digital input and wake-up event and interrupt request. 1 / 0: enable / disable.
4 - 1	1111	WO	Enable PB4~PB1 digital input and wake up event. 1 / 0: enable / disable.
0	1	WO	Enable PB0 digital input and wake-up event and interrupt request. 1 / 0: enable / disable.

9.1.3. Port A Data Registers (*PA*), address = 0x10

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

9.1.4. Port A Control Registers (*PAC*), address = 0x11

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

9.1.5. Port A Pull-High Registers (PAPH), address = 0x12

Bit	Reset	R/W	Description
7 – 3	00000	R/W	Port PA7 ~ PA3 pull-high registers. This register is used to enable the internal pull-high device on each corresponding pin of port A. 0 / 1 : disable / enable
2 – 1	-	-	Reserved. Please keep 0 for future compatibility.
0	0	R/W	Port PA0 pull-high registers. This register is used to enable the internal pull-high device on each corresponding pin of port A. 0 / 1 : disable / enable

9.1.6. Port A Pull-Low Register (PAPL), address = 0x13

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

9.1.7. Port B Data Registers (PB), address = 0x14

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

9.1.8. Port B Control Registers (PBC), address = 0x15

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

9.1.9. Port B Pull-High Registers (PBPH), address = 0x16

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

9.1.10. Port B Pull-Low Register (PBPL), address = 0x17

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

(2) Wake-up function:

When PFC232 put in Power-Down or Power-Save mode, every IO pin can be used to wake-up system by toggling its state. Therefore, those pins needed to wake-up system must be set to input mode and set the corresponding bits of registers *PxDIER* to high.

(3) External interrupt function:

When the IO acts as an external interrupt pin, the corresponding bit of *PxDIER* should be set to high. For example, *PADIER.0* should be set to high when PA0 is used as external interrupt pin.

(4) Drive capability optional:

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

9.2.3. IO Pin Usage and Setting

(1) IO pin as digital input

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

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

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

(3) PA5 is set to be PRSTB input pin.

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

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

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

10. Timer / PWM Counter

10.1. 16-bit Timer (Timer16)

10.1.1. Timer16 Introduction

PFC232 provide a 16-bit hardware timer (Timer16/T16) and its block diagram is shown in Fig. 15.

The clock source of timer16 is selected by register *T16M*[7:5]. Before sending clock to the 16-bit counter (counter16), a pre-scaling logic with divided-by-1, 4, 16 or 64 is selected by *T16M*[4:3] for wide range counting.

T16M[2:0] is used to select the interrupt request. The interrupt request from Timer16 will be triggered by the selected bit which comes from bit[15:8] of this 16-bit counter. Rising edge or falling edge can be optional chosen by register *INTEGS.4*.

The 16-bit counter performs up-counting operation only, the counter initial values can be stored from data memory by issuing the *stt16* instruction and the counting values can be loaded to data memory by issuing the *ldt16* instruction.

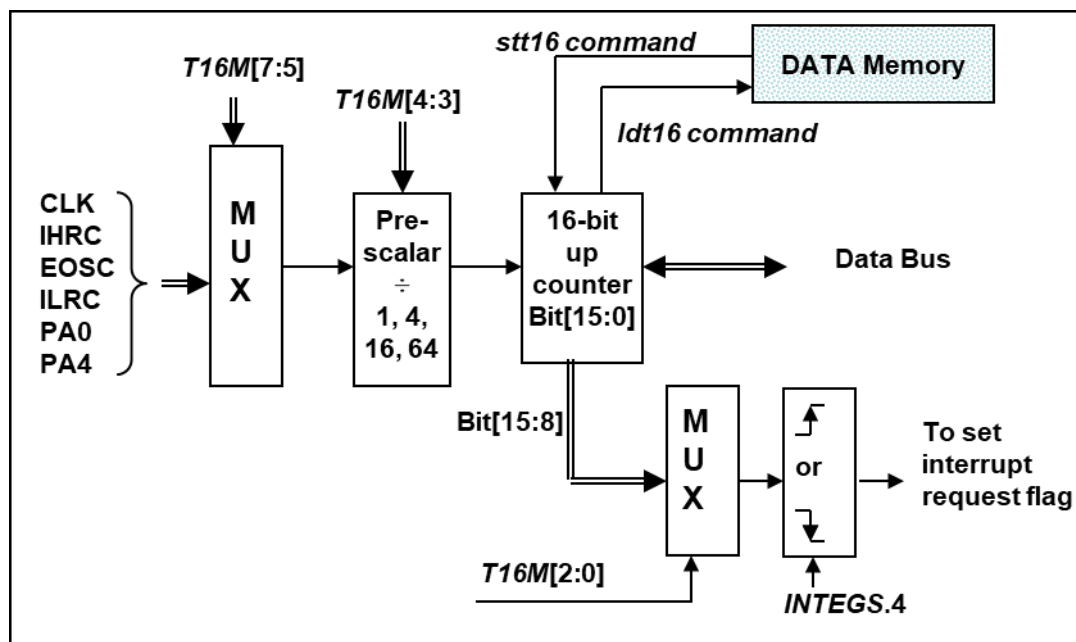


Fig. 15: Hardware diagram of Timer16

There are three parameters to define the Timer16 using; 1st parameter is used to define the clock source of Timer16, 2nd parameter is used to define the pre-scalar and the 3rd one is to define the interrupt source.

```

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

```

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

```
$ T16M  SYSCLK, /64, BIT15;
// choose (SYSCLK/64) as clock source, every 2^16 clock to set INTRQ.2=1
// if system clock SYSCLK = IHRC / 4 = 4 MHz
// SYSCLK/64 = 4 MHz/64 = 16 uS, about every 1 S to generate INTRQ.2=1

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

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

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

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

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

P is the selection of *T16M* [4:3]; (1, 4, 16, 64)

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

10.1.2. Timer16 Time Out

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

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

10.1.3. Timer16 Mode Register (T16M), address = 0x06

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

10.2. 8-bit Timer with PWM Generation (Timer2, Timer3)

Two 8-bit hardware timers (Timer2/TM2, Timer3/TM3) with PWM generation are implemented in the PFC232. Timer2 is used as the example to describe its function due to these two 8-bit timers are the same. Fig. 16 shows the Timer2 hardware diagram.

Bit[7:4] of register *TM2C* are used to select the clock source of Timer2. And the output of Timer2 is selected by *TM2C*[3:2]. The clock frequency divide module is controlled by bit [6:5] of *TM2S* register. *TM2B* register controls the upper bound of 8-bit counter. It will be clear to zero automatically when the counter values reach for upper bound register. The counter values can be set or read back by *TM2CT* register.

There are two operating modes for Timer2: period mode and PWM mode. Period mode is used to generate periodical output waveform or interrupt event, and PWM mode is used to generate PWM output waveform with optional 6-bit ~ 8-bit PWM resolution.

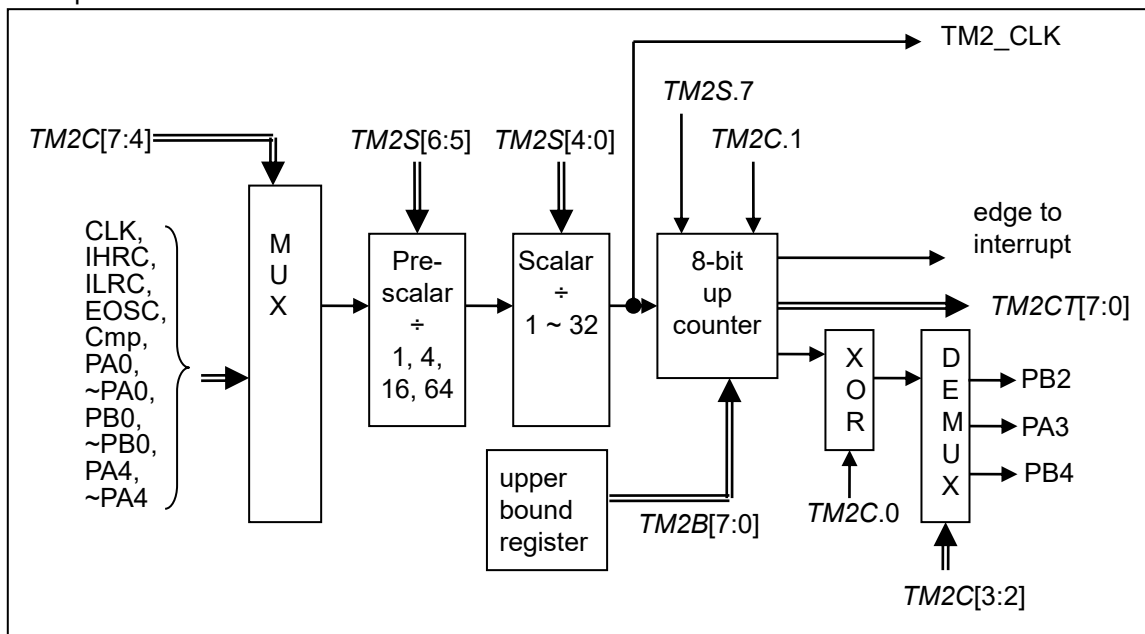


Fig. 16: Timer2 hardware diagram

The output of Timer3 can be sent to pin PB5, PB6 or PB7.

Fig. 17 shows the timing diagram of Timer2 for both period mode and PWM mode.

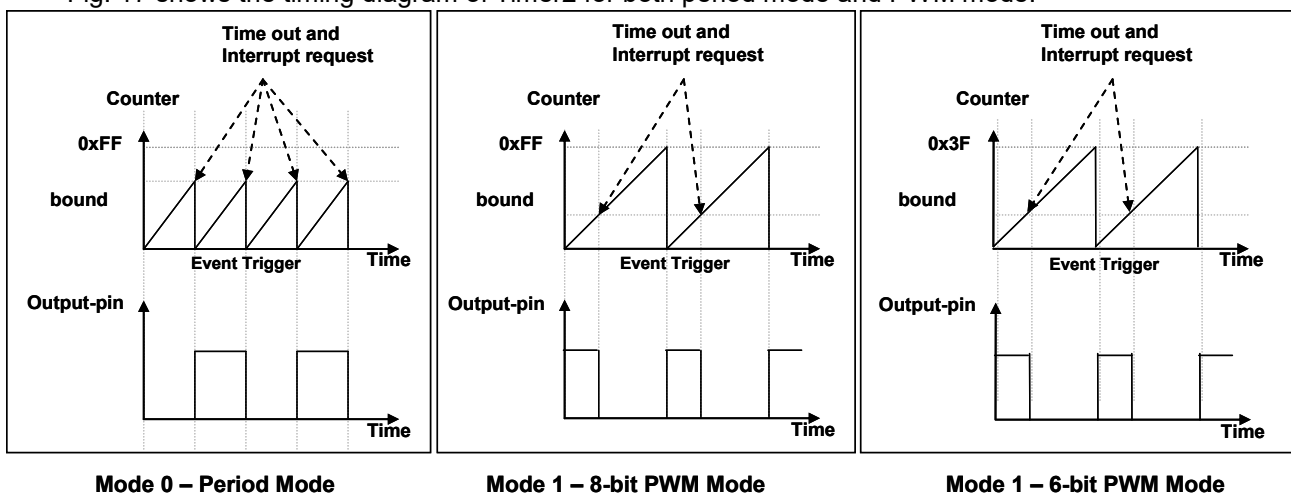


Fig. 17: Timing diagram of Timer2 in period mode and PWM mode (*TM2C.1*=1)

Code Options GPC_PWM and OPA_PWM are for the applications which need the generated PWM waveform to be controlled by the comparator result. If the Code Option GPC_PWM or OPA_PWM is selected, the PWM output stops while the comparator output is 1 and then the PWM output turns on while the comparator output goes back to 0, as shown in Fig. 18.

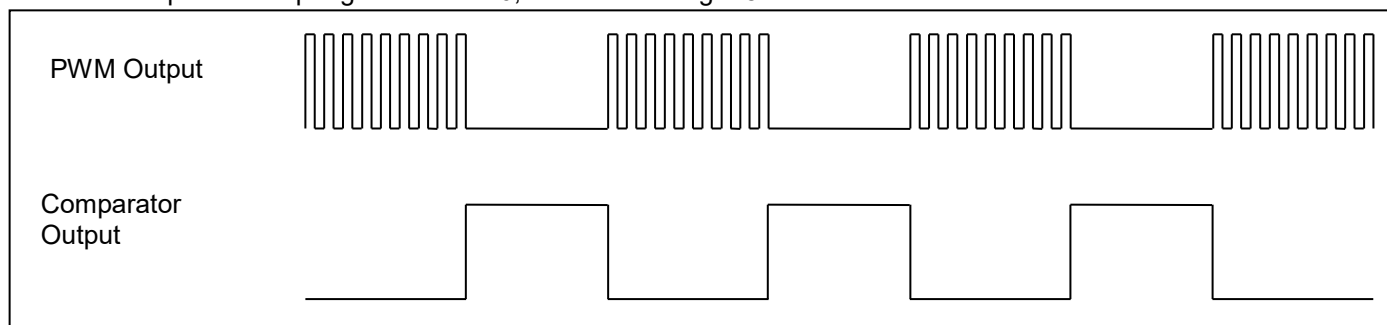


Fig.18: Comparator controls the output of PWM waveform

10.2.1. Timer2, Timer3 Related Registers

10.2.1.1. Timer2 Bound Register (*TM2B*), address = 0x09

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

10.2.1.2. Timer2 Counter Register (*TM2CT*), address = 0x1D

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

10.2.1.3. Timer2 Scalar Register (*TM2S*), address = 0x1E

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

10.2.1.4. Timer2 Control Register (*TM2C*), address = 0x1C

Bit	Reset	R/W	Description
7 – 4	0000	R/W	Timer2 clock selection. 0000 : disable 0001 : CLK 0010 : IHRC or IHRC *2 (by code option TMx_source) 0011 : EOSC 0100 : ILRC 0101 : comparator output 0110 : OPA (Comparator mode) compare output 1000 : PA0 (rising edge) 1001 : ~PA0 (falling edge) 1010 : PB0 (rising edge) 1011 : ~PB0 (falling edge) 1100 : PA4 (rising edge) 1101 : ~PA4 (falling edge) Others: reserved Notice: In ICE mode and IHRC is selected for Timer2 clock, the clock sent to Timer2 does NOT be stopped, Timer2 will keep counting when ICE is in halt state.
3 – 2	00	R/W	Timer2 output selection. 00 : disable 01 : PB2 10 : PA3 11 : PB4
1	0	R/W	Timer2 mode selection. 0 / 1 : period mode / PWM mode
0	0	R/W	Enable to inverse the polarity of Timer2 output. 0 / 1: disable / enable

10.2.1.5. Timer3 Counter Register (*TM3CT*), address = 0x33

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

10.2.1.6. Timer3 Scalar Register (*TM3S*), address = 0x34

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

10.2.1.7. Timer3 Bound Register (*TM3B*), address = 0x38

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

10.2.1.8. Timer3 Control Register (*TM3C*), address = 0x32

Bit	Reset	R/W	Description
7 – 4	0000	R/W	Timer3 clock selection. 0000 : disable 0001 : CLK 0010 : IHRC or IHRC *2 (by code option TMx_source) 0011 : EOSC 0100 : ILRC 0101 : comparator output 0110 : OPA (Comparator mode) compare output 1000 : PA0 (rising edge) 1001 : ~PA0 (falling edge) 1010 : PB0 (rising edge) 1011 : ~PB0 (falling edge) 1100 : PA4 (rising edge) 1101 : ~PA4 (falling edge) Others: reserved Notice: In ICE mode and IHRC is selected for Timer3 clock, the clock sent to Timer3 does NOT be stopped, Timer3 will keep counting when ICE is in halt state.
3 – 2	00	R/W	Timer3 output selection. 00 : disable 01 : PB5 10 : PB6 11 : PB7
1	0	R/W	Timer3 mode selection. 0 / 1 : period mode / PWM mode
0	0	R/W	Enable to inverse the polarity of Timer3 output. 0 / 1: disable / enable

10.2.2. Using the Timer2 to Generate Periodical Waveform

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

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

Where,

Y = *TM2C*[7:4] : frequency of selected clock source

K = *TM2B*[7:0] : bound register in decimal

S1 = *TM2S*[6:5] : pre-scalar (S1 = 1, 4, 16, 64)

S2 = *TM2S*[4:0] : scalar register in decimal (S2 = 0 ~ 31)

Example 1:

TM2C = 0b0001_1100, Y=4MHz

TM2B = 0b0111_1111, K=127

TM2S = 0b0_00_00000, S1=1, S2=0

frequency of output = 4MHz ÷ [2 × (127 + 1) × 1 × (0 + 1)] = 15.625KHz

Example 2:

TM2C = 0b0001_1000, Y=4MHz

TM2B = 0b0000_0001, K=1

TM2S = 0b0_00_00000, S1=1, S2=0

frequency = 4MHz ÷ [2 × (1 + 1) × 1 × (0 + 1)] = 1MHz

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

```
voidFPPA0 (void)
{
    . ADJUST_IC    SYSCLK=IHRC/4, IHRC=16MHz, VDD=5V
    ...
    TM2CT = 0x00;
    TM2B = 0x7f;
    TM2S = 0b0_00_00001;           // 8-bit PWM, pre-scalar = 1, scalar = 2
    TM2C = 0b0001_10_0_0;         // system clock, output=PA3, period mode
    while(1)
    {
        nop;
    }
}
```

10.2.3. 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:

$$\text{Frequency of Output} = Y \div [256 \times S1 \times (S2+1)]$$

$$\text{Duty of Output} = [(K+1) \div 256] \times 100\%$$

Where,

$Y = TM2C[7:4]$: frequency of selected clock source

$K = TM2B[7:0]$: bound register in decimal

$S1 = TM2S[6:5]$: pre-scalar ($S1 = 1, 4, 16, 64$)

$S2 = TM2S[4:0]$: scalar register in decimal ($S2 = 0 \sim 31$)

Example 1:

$TM2C = 0b0001_1010$, $Y=4\text{MHz}$

$TM2B = 0b0111_1111$, $K=127$

$TM2S = 0b0_00_00000$, $S1=1$, $S2=0$

frequency of output = $4\text{MHz} \div (256 \times 1 \times (0+1)) = 15.625\text{KHz}$

duty of output = $[(127+1) \div 256] \times 100\% = 50\%$

Example 2:

$TM2C = 0b0001_1010$, $Y=4\text{MHz}$

$TM2B = 0b0000_1001$, $K = 9$

$TM2S = 0b0_00_00000$, $S1=1$, $S2=0$

frequency of output = $4\text{MHz} \div (256 \times 1 \times (0+1)) = 15.625\text{KHz}$

duty of output = $[(9+1) \div 256] \times 100\% = 3.9\%$

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

```
void FPPA0(void)
{
    .ADJUST_IC    SYSCLK=IHRC/4, IHRC=16MHz, VDD=5V
    wdreset;
    TM2CT = 0x00;
    TM2B = 0x7f;
    TM2S = 0b0_00_00001;    // 8-bit PWM, pre-scalar = 1, scalar = 2
    TM2C = 0b0001_10_1_0;    // system clock, output=PA3, PWM mode
    while(1)
    {
        nop;
    }
}
```

10.2.4. 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:

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

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

Where,

$TM2C[7:4] = Y$: frequency of selected clock source

$TM2B[7:0] = K$: bound register in decimal

$TM2S[6:5] = S1$: pre-scalar ($S1 = 1, 4, 16, 64$)

$TM2S[4:0] = S2$: scalar register in decimal ($S2 = 0 \sim 31$)

Example 1:

$TM2C = 0b0001_1010$, $Y=4\text{MHz}$

$TM2B = 0b0011_1111$, $K=63$

$TM2S = 0b1_00_00000$, $S1=1$, $S2=0$

frequency of output = $4\text{MHz} \div (64 \times 1 \times (0+1)) = 62.5\text{KHz}$

duty of output = $[(63+1) \div 64] \times 100\% = 100\%$

Example 2:

$TM2C = 0b0001_1010$, $Y=4\text{MHz}$

$TM2B = 0b0000_0000$, $K=0$

$TM2S = 0b1_00_00000$, $S1=1$, $S2=0$

Frequency = $4\text{MHz} \div (64 \times 1 \times (0+1)) = 62.5\text{KHz}$

Duty = $[(0+1) \div 64] \times 100\% = 1.5\%$

10.3. 11-bit PWM Generation

Three 11-bit hardware PWM generators (PWMG0, PWMG1 & PWMG2) are implemented in the PFC232. PWMG0 is used as the example to describe its functions due to all of them are almost the same. Their individual outputs are listed as below:

- (1) PWMG0 – PA0, PB4, PB5
- (2) PWMG1 – PA4, PB6, PB7
- (3) PWMG2 – PA3, PB2, PB3, PA5 (Only PA5 ICE does not support)

10.3.1. PWM Waveform

A PWM output waveform (Fig. 19) has a time-base ($T_{\text{Period}} = \text{Time of Period}$) and a time with output high level (Duty Cycle). The frequency of the PWM output is the inverse of the period ($f_{\text{PWM}} = 1/T_{\text{Period}}$), the resolution of the PWM is the clock count numbers for one period ($N \text{ bits resolution}, 2^N \times T_{\text{clock}} = T_{\text{Period}}$).

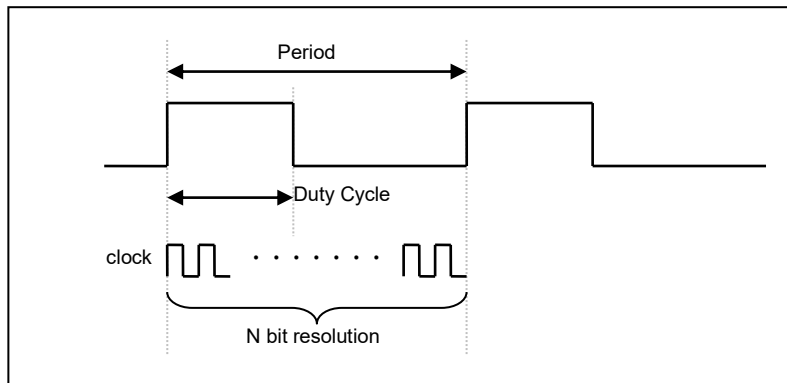


Fig. 19: PWM Output Waveform

10.3.2. Hardware and Timing Diagram

Fig. 20 shows the hardware diagram of 11-bit Timer. The clock source can be IHRC or system clock. The PWM output pin is selected by register *PWMG0C*. The period of PWM waveform is defined in the PWM upper bond high and low registers (*PWMG0CUBH* and *PWMG0CUBL*), the duty cycle of PWM waveform is defined in the PWM duty high and low registers (*PWMG0DTH* and *PWMG0DTL*).

Selecting code option GPC_PWM can also control the generated PWM waveform by the comparator result. Please refer the section of Timer2 for further information.

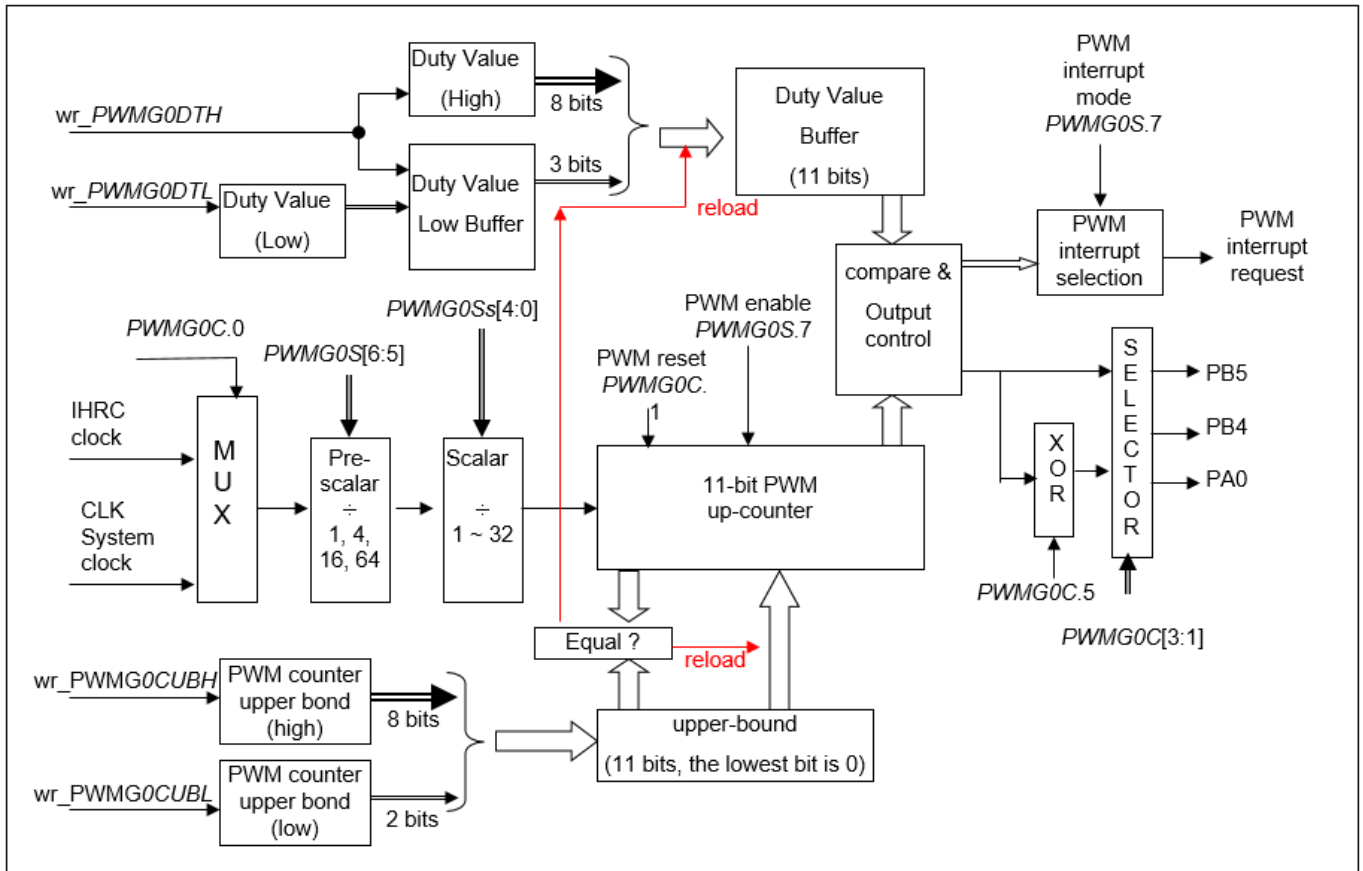


Fig. 20: Hardware Diagram of 11-bit PWM Generator 0 (PWMG0)

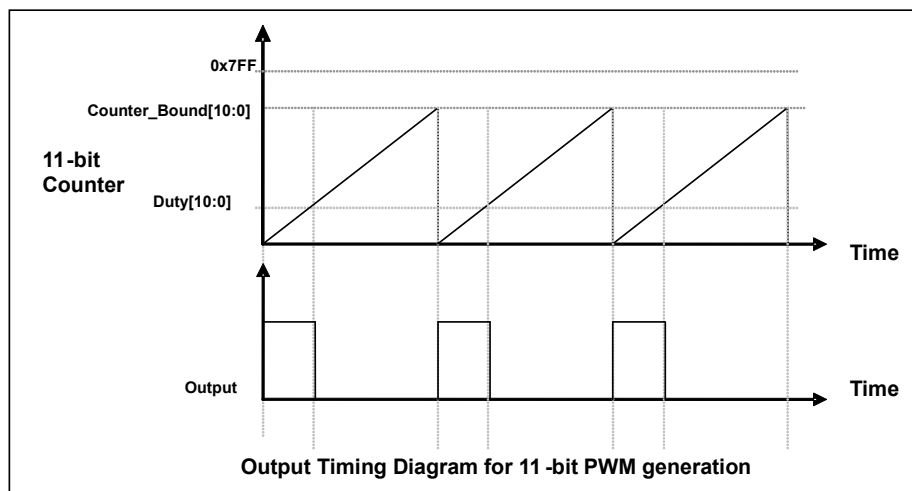


Fig. 21: Output Timing Diagram of 11-bit PWM Generator

10.3.3. Equations for 11-bit PWM Generator

The frequency and duty cycle of 11bit PWM can be obtained from the following formula:

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

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

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

Where,

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

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

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

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

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

10.3.4. 11bit PWM Related Registers

10.3.4.1. PWMG0 control Register (*PWMG0C*), address = 0x20

Bit	Reset	R/W	Description
7	0	R/W	Enable PWMG0 generator. 0 / 1: disable / enable
6	-	RO	Output status of PWMG0 generator.
5	0	R/W	Enable to inverse the polarity of PWMG0 generator output. 0 / 1: disable / enable.
4	0	R/W	PWMG0 counter reset. Writing "1" to clear PWMG0 counter and this bit will be self-clear to 0 after counter reset.
3 – 1	0	R/W	Select PWM output pin for PWMG0. 000: none 001: PB5 011: PA0 100: PB4 Others: reserved
0	0	R/W	Clock source of PWMG0 generator. 0: CLK, 1: IHRC or IHRC*2 (by code option PWM_source)

10.3.4.2. PWMG0 Scalar Register (*PWMG0S*), address = 0x21

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

10.3.4.3. PWMG0 Duty Value High Register (*PWMG0DTH*), address = 0x22

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

10.3.4.4. PWMG0 Duty Value Low Register (*PWMG0DTL*), address = 0x23

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

Note: It's necessary to write *PWMG0DTL* Register before writing *PWMG0DTH* Register.

10.3.4.5. PWMG0 Counter Upper Bound High Register (*PWMG0CUBH*), address = 0x24

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

10.3.4.6. PWMG0 Counter Upper Bound Low Register (*PWMG0CUBL*), address = 0x25

Bit	Reset	R/W	Description
7 – 6	-	WO	Bit[2:1] of PWMG0 counter upper bound.
5	-	WO	Bit[0] of PWMG0 counter upper bound.
4 – 0	-	-	Reserved

10.3.4.7. PWMG1 Control Register (*PWMG1C*), address = 0x26

Bit	Reset	R/W	Description
7	0	R/W	Enable PWMG1. 0 / 1: disable / enable
6	-	RO	Output of PWMG1.
5	0	R/W	Enable to inverse the polarity of PWMG1 output. 0 / 1 : disable / enable.
4	0	R/W	PWMG1 counter reset. Writing “1” to clear PWMG1 counter and this bit will be self clear to 0 after counter reset.
3 – 1	0	R/W	Select PWMG1 output pin. 000: none 001: PB6 011: PA4 100: PB7 Others: reserved
0	0	R/W	Clock source of PWMG1. 0: CLK, 1: IHRC or IHRC*2 (by code option PWM_source)

10.3.4.8. PWMG1 Scalar Register (*PWMG1S*), address = 0x27

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

10.3.4.9. PWMG1 Duty Value High Register (*PWMG1DTH*), address = 0x28

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

10.3.4.10. PWMG1 Duty Value Low Register (*PWMG1DTL*), address = 0x29

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

Note: It's necessary to write *PWMG1DTL* Register before writing *PWMG1DTH* Register.

10.3.4.11. PWMG1 Counter Upper Bound High Register (*PWMG1CUBH*), address = 0x2A

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

10.3.4.12. PWMG1 Counter Upper Bound Low Register (*PWMG1CUBL*), address = 0x2B

Bit	Reset	R/W	Description
7 – 6	00	WO	Bit[2:1] of PWMG1 counter upper bound.
5	0	WO	Bit[0] of PWMG1 counter upper bound.
4 – 0	-	-	Reserved

10.3.4.13. PWMG2 Control Register (*PWMG2C*), address = 0x2C

Bit	Reset	R/W	Description
7	0	R/W	Enable PWMG2. 0 / 1: disable / enable.
6	-	RO	Output of PWMG2.
5	0	R/W	Enable to inverse the polarity of PWMG2 output. 0 / 1: disable / enable.
4	0	R/W	PWMG2 counter reset. Writing "1" to clear PWMG2 counter and this bit will be self-clear to 0 after counter reset.
3 – 1	0	R/W	Select PWMG2 output pin. 000: disable 001: PB3 011: PA3 100: PB2 101: PA5 (ICE does NOT Support.) Others: reserved
0	0	R/W	Clock source of PWMG2. 0: CLK, 1: IHRC or IHRC*2 (by code option PWM_source)

10.3.4.14. PWMG2 Scalar Register (*PWMG2S*), address = 0x2D

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

10.3.4.15. PWMG2 Duty Value High Register (*PWMG2DTH*), address = 0x2E

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

10.3.4.16. PWMG2 Duty Value Low Register (*PWMG2DTL*), address = 0x2F

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

Note: It's necessary to write *PWMG2DTL* Register before writing *PWMG2DTH* Register.

10.3.4.17. PWMG2 Counter Upper Bound High Register (*PWMG2CUBH*), address = 0x30

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

10.3.4.18. PWMG2 Counter Upper Bound Low Register (*PWMG2CUBL*), address = 0x31

Bit	Reset	R/W	Description
7 – 6	00	WO	Bit[2:1] of PWMG2 counter upper bound.
5	0	WO	Bit[0] of PWMG2 counter upper bound.
4 – 0	-	-	Reserved

10.3.5. Examples of PWM Waveforms with Complementary Dead Zones

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

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

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

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

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

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

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

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

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

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

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

The PWM0 / PWM1 waveform obtained by the above program is shown in Fig. 22.

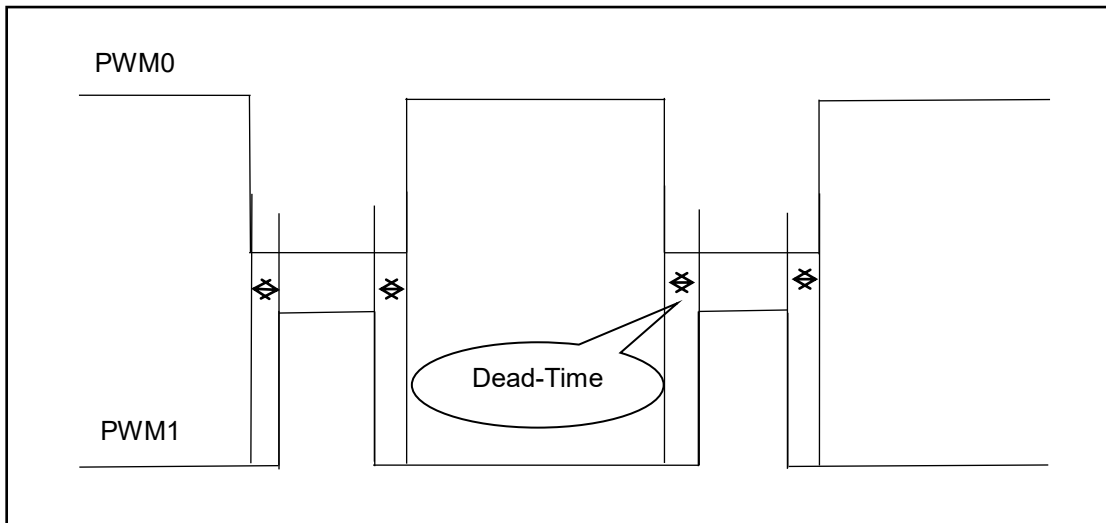


Fig. 22: Two complementary PWM waveforms with dead zones

Users can modify the **dead_zone_R** and **dead_zone_F** values in the program to adjust the dead-time. Table 10 provides data corresponding to different dead-time for users' reference. Where, if dead-time = 4us, then there are dead zones of 4us before and after PWM1 high level.

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

Table 10: The value of dead-time for reference

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

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

11. Special Functions

11.1. Comparator

One hardware comparator is built inside the PFC232; Fig. 23 shows its hardware diagram. It can compare signals between two input pins. The two signals to be compared, one is the plus input and the other one is the minus input. The plus input pin is selected by register *GPCC.0*, and the minus input pin is selected by *GPCC[3:1]*.

The output result can be:

- (1) read back by *GPCC.6*;
- (2) inversed the polarity by *GPCC.4*;
- (3) sampled by Time2 clock (TM2_CLK) which comes from *GPCC.5*;
- (4) enabled to output to PA0 directly by *GPCS.7*;
- (5) used to request interrupt service.

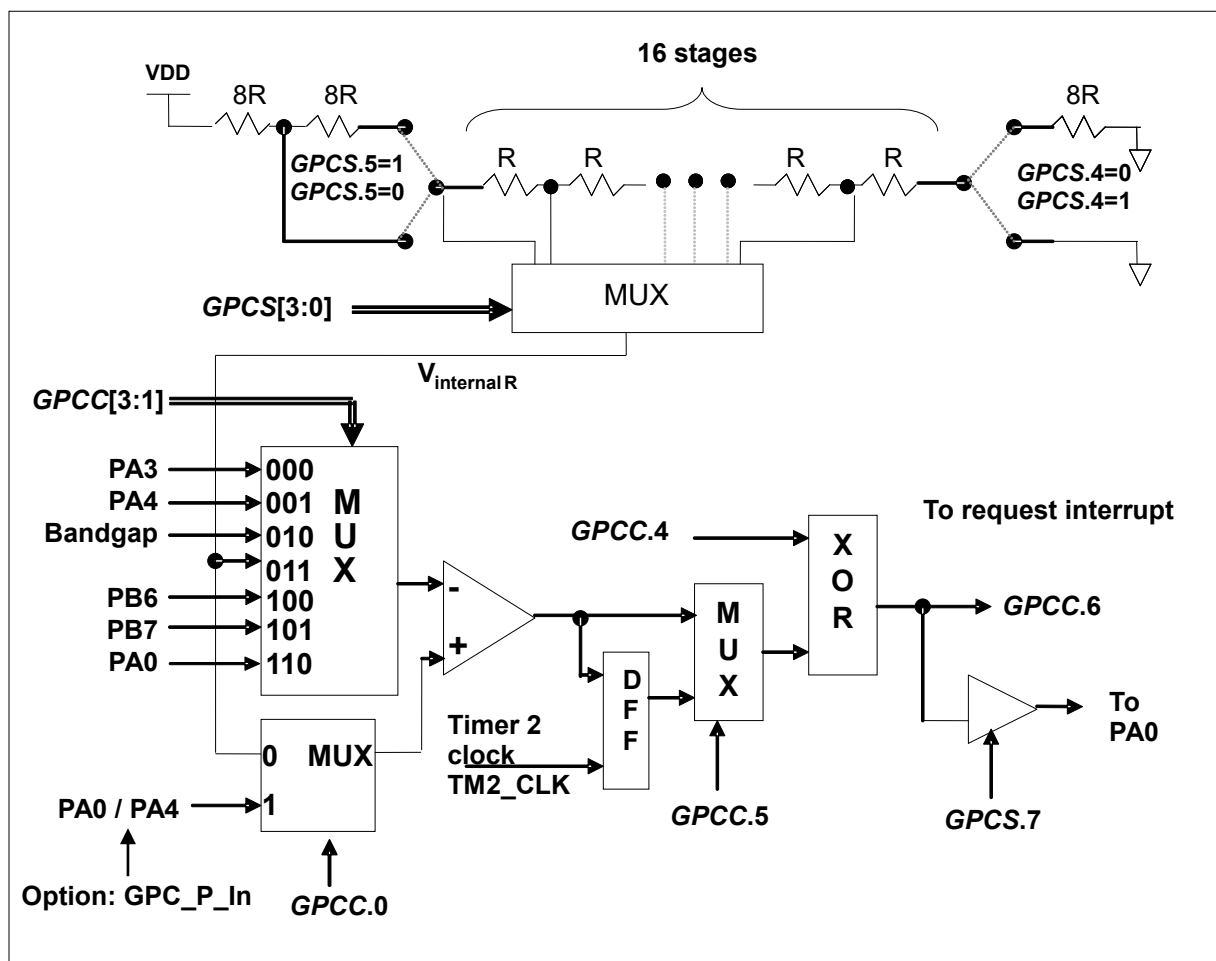


Fig. 23: Hardware diagram of comparator

11.1.1. Comparator Control Register (GPCC), address = 0x18

Bit	Reset	R/W	Description
7	0	R/W	Enable comparator. 0 / 1 : disable / enable When this bit is set to enable, please also set the corresponding input pins to be digital disable to prevent IO leakage.
6	-	RO	Comparator result. 0: plus input < minus input 1: plus input > minus input
5	0	R/W	Select whether the comparator result output will be sampled by TM2_CLK? 0: result output NOT sampled by TM2_CLK 1: result output sampled by TM2_CLK
4	0	R/W	Inverse the polarity of result output of comparator. 0: polarity is NOT inversed. 1: polarity is inversed.
3 – 1	000	R/W	Selection the minus input (-) of comparator. 000: PA3 001: PA4 010: Internal 1.20 volt Bandgap reference voltage (not suitable for the comparator wake-up function) 011: V _{internal R} 100: PB6 101: PB7 110 : PA0 111: reserved
0	0	R/W	Selection the plus input (+) of comparator. 0 : V _{internal R} 1 : PA4 or PA0 (by code option GPC_P_In)

11.1.2. Comparator Selection Register (GPCS), address = 0x19

Bit	Reset	R/W	Description
7	0	WO	Comparator output enable (to PA0). 0 / 1 : disable / enable Before setting this bit to enable the comparator, please make sure the OPA is not enabled to prevent signal fighting at PA0.
6	0	WO	Wakeup by comparator enable. (The comparator wakeup effectively when gpcc.6 electrical level changed) 0 / 1 : disable / enable
5	0	WO	Selection of high range of comparator.
4	0	WO	Selection of low range of comparator.
3 – 0	0000	WO	Selection the voltage level of comparator. 0000 (lowest) ~ 1111 (highest)

11.1.3. Internal Reference Voltage ($V_{\text{internal R}}$)

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

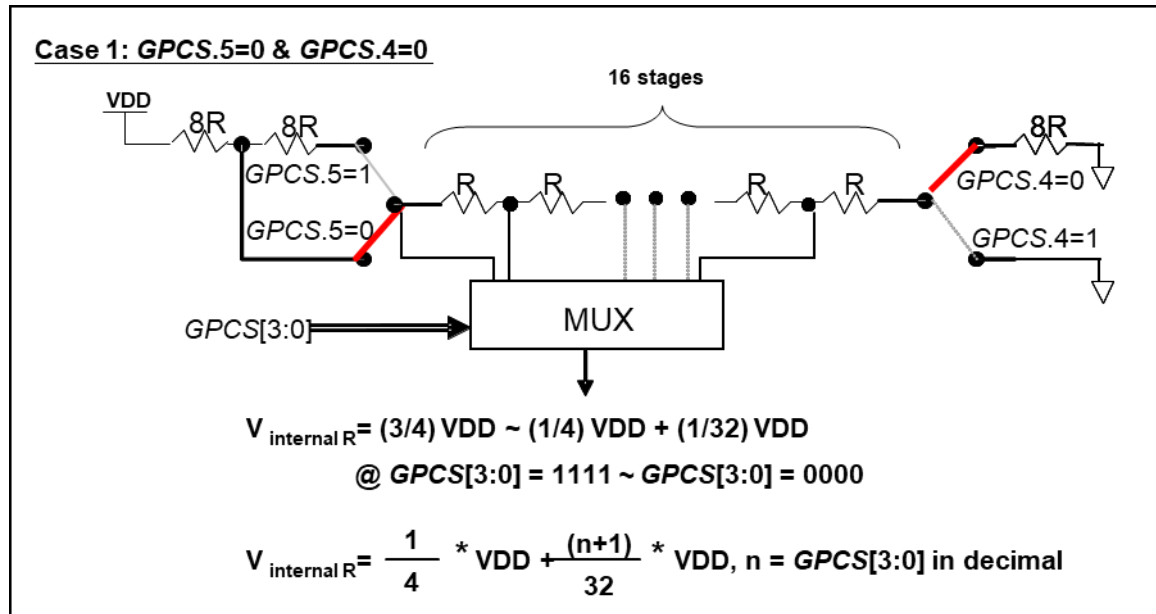


Fig. 24: $V_{\text{internal R}}$ hardware connection if $GPCS.5=0$ and $GPCS.4=0$

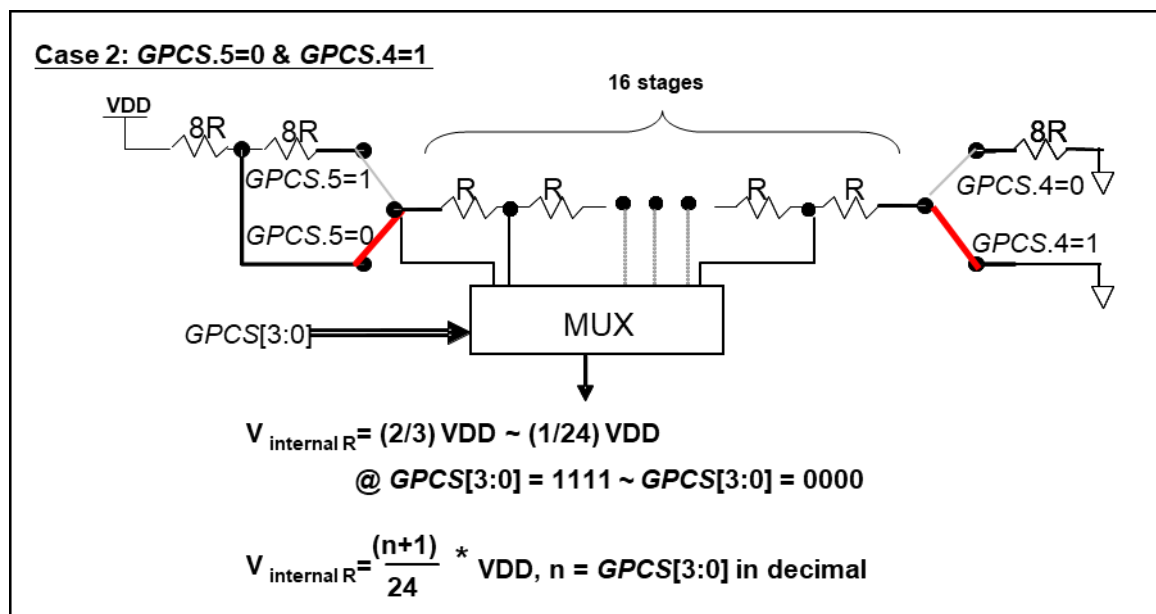


Fig. 25: $V_{\text{internal R}}$ hardware connection if $GPCS.5=0$ and $GPCS.4=1$

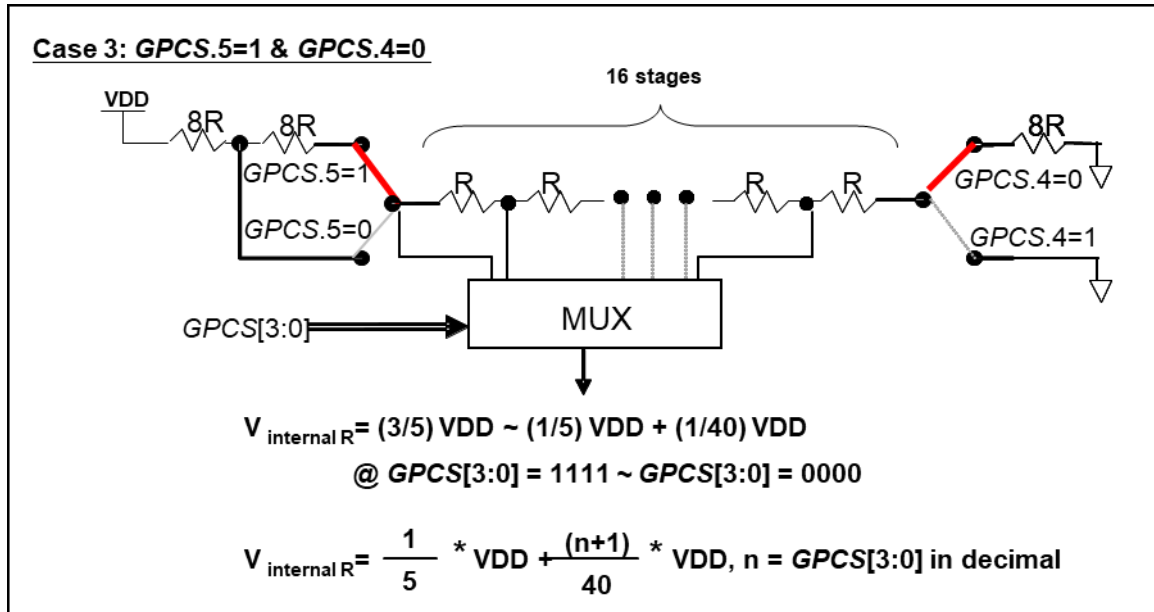


Fig. 26: $V_{internal R}$ hardware connection if $GPCS.5=1$ and $GPCS.4=0$

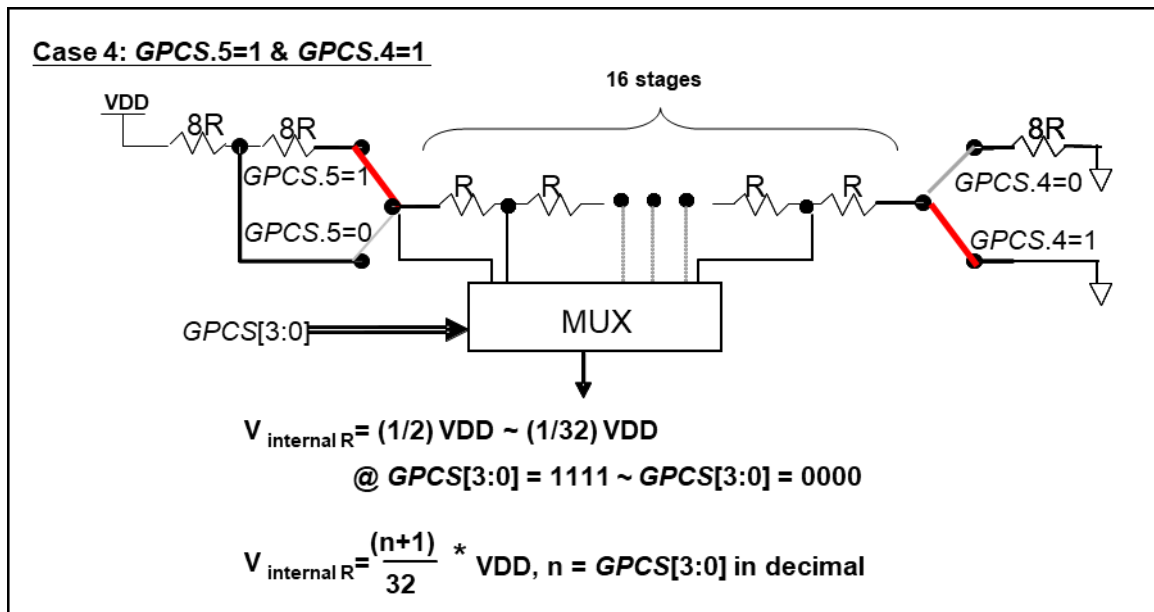


Fig. 27: $V_{internal R}$ hardware connection if $GPCS.5=1$ and $GPCS.4=1$

11.1.4. Using the Comparator

Case 1:

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

```
GPCS  = 0b0_0_00_1001;    // Vinternal R = VDD*(18/32)
GPCC  = 0b1_0_0_0_000_0;  // enable comp, - input: PA3, + input: Vinternal R
PADIER = 0bxxxx_0_xxx;    // disable PA3 digital input to prevent leakage current
```

or

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

Case 2:

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

```
GPCS  = 0b1_0_10_1101;    // output to PA0, Vinternal R = VDD*(22/40)
GPCC  = 0b1_0_0_1_011_1;  // Inverse output, - input: Vinternal R, + input: PA4
PADIER = 0bxxxx_0_xxx;    // disable PA4 digital input to prevent leakage current
```

or

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

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

11.1.5. Using the Comparator and Bandgap 1.20V

The internal Bandgap module provides a stable 1.20V output, and it can be used to measure the external supply voltage level. The Bandgap 1.20V is selected as minus input of comparator and $V_{\text{internal R}}$ is selected as plus input, the supply voltage of $V_{\text{internal R}}$ is VDD, the VDD voltage level can be detected by adjusting the voltage level of $V_{\text{internal R}}$ to compare with Bandgap.

If N ($GPCS[3:0]$ in decimal) is the number to let $V_{\text{internal R}}$ closest to Bandgap 1.20 volt, the supply voltage VDD can be calculated by using the following equations:

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

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

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

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

Case 1:

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

11.2. VDD/2 Bias Voltage Generator

The five pins of PFC232: PA0, PA3, PA4, PB0 and PB3, can generate VDD/2 as the COM function when driving the LCD display. This function can be enabled by setting the register *MISC.4* as 1.

MISC Register (<i>MISC</i>), address = 0x08			
Bit	Reset	R/W	Description
7 – 5	-	-	Reserved. (keep 0 for future compatibility)
4	0	WO	Enable VDD/2 bias voltage generator 0 / 1 : Disable / Enable (ICE cannot be dynamically switched)
3	-	-	Reserved.
2	0	WO	Disable LVR function. 0 / 1 : Enable / Disable
1 – 0	00	WO	Watch dog time out period

The COM port can generate VDD/2 by switching it to input mode (*PAC.x* / *PBC.x*=0). However, keep in mind to turn off the pull-high / pull-low resistor (*PxPH.x* / *PxPL.x*=0) and digital input from *PADIER.x* / *PBDIER.x* register to prevent the output voltage level from disturbing. Fig.28 shows how to use this function.

The output function of COM port is the same as other normal IO.

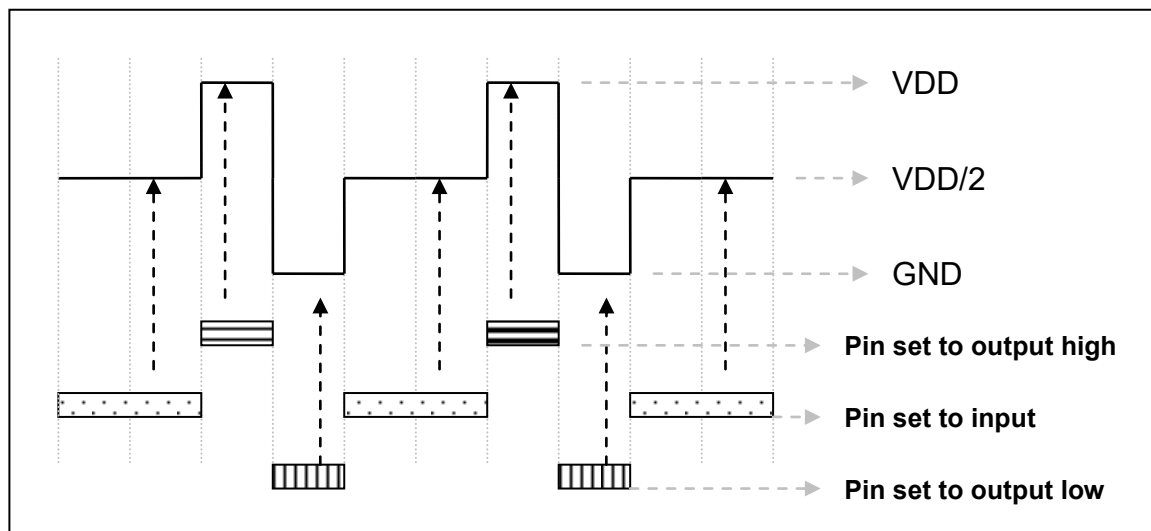
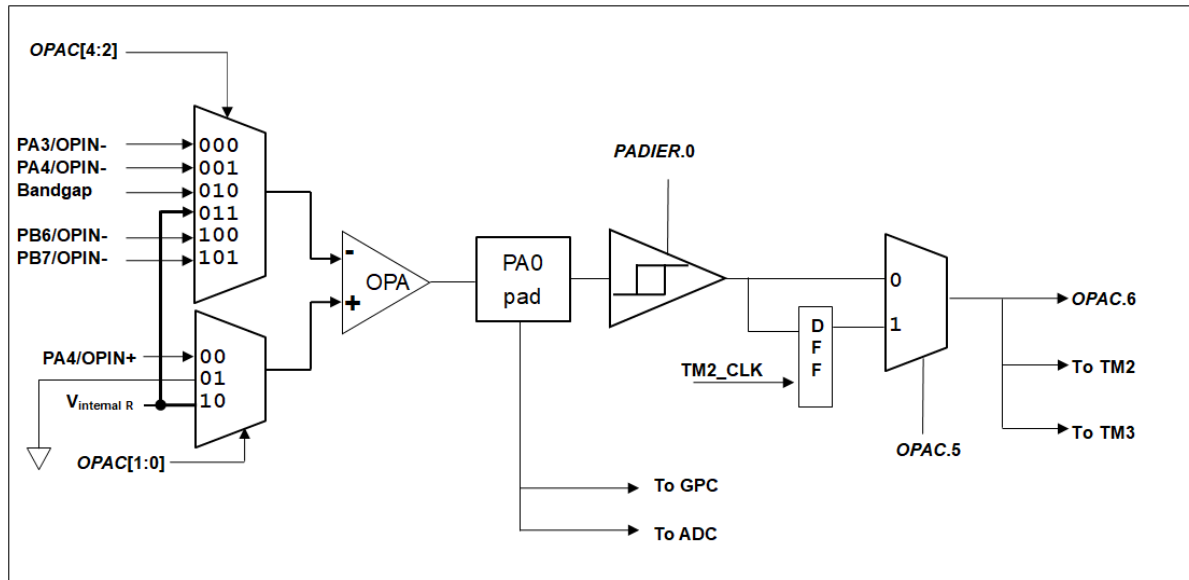


Fig. 28: Using VDD/2 bias voltage generator

ICE does NOT support VDD/2 function of PB3.

11.3. Operational Amplifier (OPA) module

An Operational Amplifier (OPA) is built-in in the PFC232, the basic configuration is shown in the following diagram. There are 2 different configurations, one is called OPA Comparator mode and the other is Amplifier mode.



When user turns on the OPA by enabling the *OPAC.7*, the IO port PA0 turns to be the output of OPA. User can measure the analog PA0 voltage by GPC or ADC in Amplifier mode, or read the digital OPA compare result from PA0 directly in Comparator mode.

ICE does NOT support OPA.

11.3.1. OPA Comparator Mode

The open-loop configuration shown in the above diagram is called OPA Comparator mode. There is no feedback path between input and output (PA0) of OPA. In this mode, user can enable *PADIER.0* to read the compare result from *OPAC.6*.

Like the comparator (GPC), the compare result of OPA can also be the counting source of TM2 or TM3. Also like GPC, the OPA compare result can also be used to control generated PWM waveform by enable code option *OPA_PWM*.

Moreover, user can select the internal reference voltage $V_{\text{internal R}}$ which generated in GPC as one of the plus or minus input of OPA as the compare reference voltage.

11.3.2. OPA Amplifier Mode

Another configuration is called Amplifier mode, which needs some external components to make a feedback amplifier. When it is configured as a feedback amplifier, PA0 becomes an analog output pad. Always keep in mind to disable *PADIER.0* to prevent it from current leakage.

PA0 can be selected as the input of comparator (GPC) or ADC, therefore the output voltage of OPA can be compared by GPC or measured by ADC as well.

11.3.3. OPA Control Register (OPAC), address = 0x1A

Bit	Reset	R/W	Description
7	0	R/W	Enable OPA. 0 / 1 : disable / enable When this bit is set to enable, please also set the corresponding analog input pins to be digital disable to prevent IO leakage. Please note PA0 will be assigned to the output of OPA when it is enabled.
6	-	RO	Compare result of OPA in Comparator mode. 0: plus input < minus input 1: plus input > minus input
5	0	R/W	Select whether the compare result of OPA output is sampled by TM2_CLK? 0: result output NOT sampled by TM2_CLK 1: result output sampled by TM2_CLK
4 – 2	000	R/W	Selection the minus input (-) of OPA. 000 : PA3 001 : PA4 010 : Internal 1.20 volt Bandgap reference voltage 011 : V _{internal R} 100 : PB6 101 : PB7 11X: reserved
1 – 0	00	R/W	Selection the plus input (+) of OPA. 00 : PA4 01 : GND 10 : V _{internal R} from the comparator (refer to GPCS register) 11 : reserved

11.3.4. OPA Offset Register (OPAOFS), address = 0x07

Bit	Reset	R/W	Description
7 – 4	-	-	Reserved
3 – 0	0000	R/W	Select the offset level of OPA. 0000 : +1mV 0001 : +2mV 0010 : +5mV 0011 : +10mV 0100 : +15mV 0101 : +20mV 0110 : +25mV 0111 : +30mV 1000 : -1mV 1001 : -2mV 1010 : -5mV 1011 : -10mV 1100 : -15mV 1101 : -20mV 1110 : -25mV 1111 : -30mV

11.4. Analog-to-Digital Conversion (ADC) module

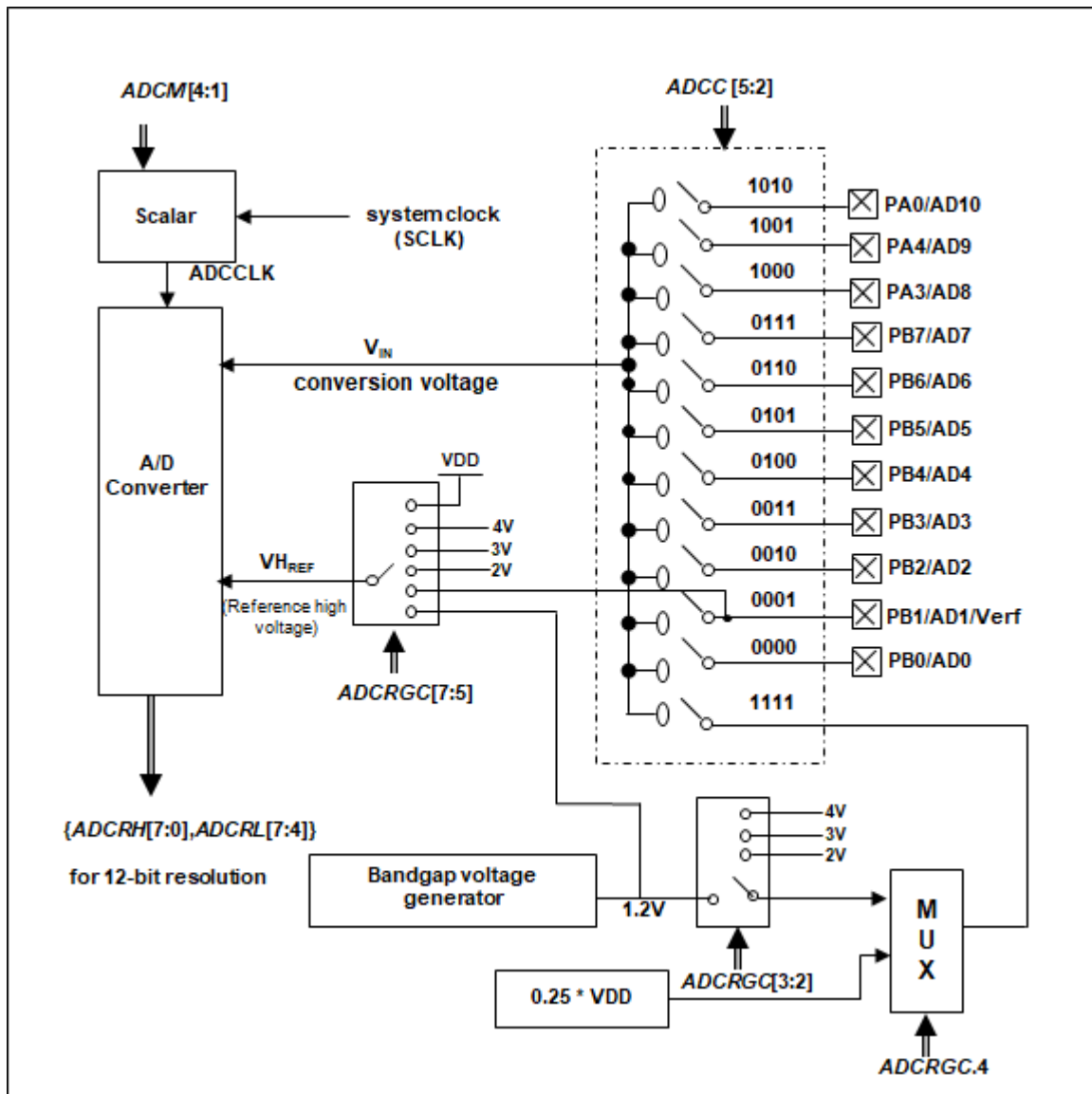


Fig.29: ADC Block Diagram

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

- (1) Configure the voltage reference high by *ADCRGC* register
- (2) Configure the AD conversion clock by *ADCM* register
- (3) Configure the pin as analog input by *PADIER*, *PBDIER* register
- (4) Select the ADC input channel by *ADCC* register
- (5) Enable the ADC module by *ADCC* register
- (6) Delay a certain amount of time after enabling the ADC module

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

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

- (7) Execute the AD conversion and check if ADC data is ready
set '1' to *ADCC.6* to start the conversion and check whether *ADCC.6* is '1'
- (8) Read the ADC result registers:
First read the *ADCRH* register and then read the *ADCRL* register.

11.4.1. The input requirement for AD conversion

Legend

C_{PIN}	= input capacitance
V_T	= threshold voltage
$I_{leakage}$	= leakage current at the pin due to various junctions
R_{IC}	= interconnect resistance
SS	= sampling switch
C_{HOLD}	= sample/hold capacitance

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

11.4.2. Select the reference high voltage

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

11.4.3. ADC clock selection

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

11.4.4. Configure the analog pins

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

The measurement signals of ADC belong to small signal; it should avoid the measured signal to be interfered during the measurement period. The selected pin should:

- (1) be set to input mode
- (2) turn off weak pull-high / pull-low resistor
- (3) set the corresponding pin to analog input by port A/B digital input disable register (*PADIER* / *PBDIER*).

11.4.5. Using the ADC

The following example shows how to use ADC with PB0~PB3.

First, defining the selected pins:

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

Next, setting ADCC register, example as below:

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

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

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

Next, delay 400us, example as below:

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

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

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

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

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

Then, start the ADC conversion:

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

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

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

The ADC can be disabled by using the following method:

```
$ ADCC Disable;
```

or

```
ADCC = 0;
```

11.4.6. ADC Related Registers

11.4.6.1. ADC Control Register (ADCC), address = 0x35

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

11.4.6.2. ADC Mode Register (ADCM), address = 0x36

Bit	Reset	R/W	Description
7 – 5	000	WO	Bit Resolution. 100:12-bit, AD 12-bit result [11:0] = { <i>adcrh</i> [7:0], <i>adcl</i> [7:4] }. Others: reserved,
4	-	-	Reserved (keep 0 for future compatibility)
3 – 1	000	WO	ADC clock source selection. 000: CLK (system clock) ÷ 1, 001: CLK (system clock) ÷ 2, 010: CLK (system clock) ÷ 4, 011: CLK (system clock) ÷ 8, 100: CLK (system clock) ÷ 16, 101: CLK (system clock) ÷ 32, 110: CLK (system clock) ÷ 64, 111: CLK (system clock) ÷ 128,
0	-	-	Reserved

11.4.6.3. ADC Regulator Control Register (*ADCRGC*), address = 0x39

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

11.4.6.4. ADC Result High Register (*ADCRH*), address = 0x37

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.

11.4.6.5. ADC Result Low Register (*ADCRL*), address = 0x38

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

11.5. Multiplier

There is an 8x8 multiplier on-chip to enhance hardware capability in arithmetic function, its multiplication is an 8 x 8 unsigned operation and can be finished in one clock cycle. Before issuing the *mul* command, both multiplicand and multiplier must be put on *ACC* and register *MULOP* (0x08); After *mul* command, the high byte result will be put on register *MULRH* (0x09) and low byte result on *ACC*. The hardware diagram of this multiplier is shown as Fig.31.

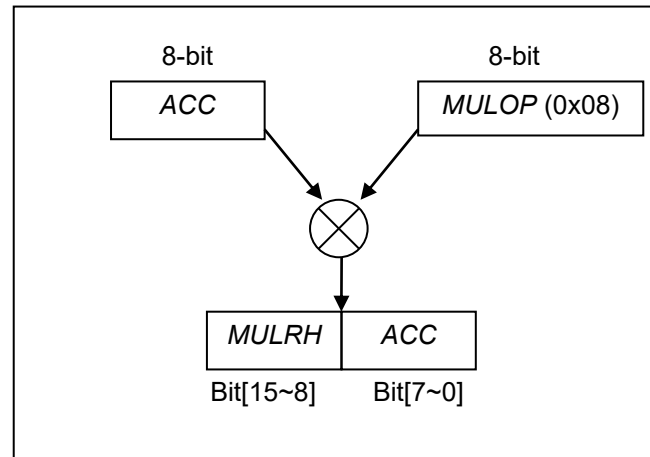


Fig.31: Block diagram of hardware multiplier

11.5.1. Multiplier Operand Register (MULOP), address = 0x08

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

11.5.2. Multiplier Result High Byte Register (MULRH), address = 0x09

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

12. Notes for Emulation

It is recommended to use 5S-I-S01/2(B) for emulation of PFC232. Please notice the following items while simulating:

- (1) 5S-I-S01/2(B) is a single-core emulator, so it can only simulate the prime core (FPPA0) function, and does NOT support the multi-core simulation.
- (2) 5S-I-S01/2(B) doesn't support the instruction *nadd, comp*.
- (3) 5S-I-S01/2(B) doesn't support $SYSCLK = ILRC/16$.
- (4) 5S-I-S01/2(B) doesn't support the function *TM2C.GPCRS/TM3C.GPCRS*.
- (5) 5S-I-S01/2(B) doesn't support the OPA.
- (6) 5S-I-S01/2(B) doesn't support the function *PAPL/PBPL*.
- (7) 5S-I-S01/2(B) doesn't support the function *GPCC.P_PA0*.
- (8) 5S-I-S01/2(B) doesn't support the dynamic setting of function *MISC.4* (Only fix to 0 or 1).
- (9) 5S-I-S01/2(B) doesn't support the function *PWMG2C.PA5*.
- (10) 5S-I-S01/2(B) doesn't support the function *ADCRGC.BG_2V/BG_3V/BG_4V*, and fix *BG_1V2* only.
- (11) 5S-I-S01/2(B) doesn't support PB1 as an external reference voltage of ADC.
- (12) 5S-I-S01/2(B) doesn't support the code options: *PB4_PB7_Drive*, *GPC_P_In*, *OPA_PWM*, *GPC_PWM*, *PWM_Source*, *TMx_Source* and *TMx_bit*.
- (13) The ILRC frequency of the 5S-I-S01/2(B) simulator is different from the actual IC and is uncalibrated, with a frequency range of about 34K~38KHz.
- (14) 5S-I-S01/2(B) doesn't support *VDD/2* function of PB3.
- (15) The PA3 output function will be affected when *GPCS* selects output to PA0 output.
- (16) When using PB1 in *ADCRGC*, PA1 must float.
- (17) When using 5S-I-S01/2(B) for simulation, changing the value of *tm2ct/tm3ct* will affect the duty during timer2/timer3 period mode. But it will not be affected for the actual IC.
- (18) Fast Wakeup time is different from 5S-I-S01/2(B): 128 $SYSCLK$, PFC232: 45 $ILRC$
- (19) Watch dog time out period is different from 5S-I-S01/2(B):

WDT period	5S-I-S01/2(B)	PFC232
<i>MISC</i> [1:0]=00	$2048 * T_{ILRC}$	$8192 * T_{ILRC}$
<i>MISC</i> [1:0]=01	$4096 * T_{ILRC}$	$16384 * T_{ILRC}$
<i>MISC</i> [1:0]=10	$16384 * T_{ILRC}$	$65536 * T_{ILRC}$
<i>MISC</i> [1:0]=11	$256 * T_{ILRC}$	$262144 * T_{ILRC}$

13. Program Writing

Please use 5S-P-003x to program. 3S-P-002 or older versions do not support programming PFC232.

Jumper connection: Please follow the instruction inside the writer software to connect the jumper.

Please select the following program mode according to the actual situation.

13.1. Normal Programming Mode

Range of application:

- Single-Chip-Package IC with programming at the writer IC socket or on the handler.
- Multi-Chip-Package(MCP) with PFC232. Be sure its connected IC and devices will not be damaged by the following voltages, and will not clam the following voltages.

The voltage conditions in normal programming mode:

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

Important Cautions:

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

13.2. Limited-Voltage Programming Mode

Range of application:

- On-Board writing. Its peripheral circuits and devices will not be damaged by the following voltages, and will not clam the following voltages. Please refer to Chapter 13.3 for more details about On-Board Writing.
- Multi-Chip-Package(MCP) with PFC232. Please be sure that its connected IC and devices will not be damaged by the following voltages, and will not clam the following voltages.

The voltage conditions in Limited-Voltage programming mode:

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

Please select "MTP On-Board VDD Limitation" or "On-Board Program" on the writer screen to enable the limited-voltage programming mode. (Please refer to the file of Writer "5S-P-003x UM").

13.3. On-Board Writing

PFC232 can support On-board writing. On-Board Writing is known as the situation that the IC have to be programmed when the IC itself and other peripheral circuits and devices have already been mounted on the PCB. Five wires of 5S-P-003x are used for On-Board Writing: ICPCK, ICPDA, VDD, GND and ICVPP. They are used to connect PA3, PA6, VDD, GND and PA5 of the IC correspondingly.

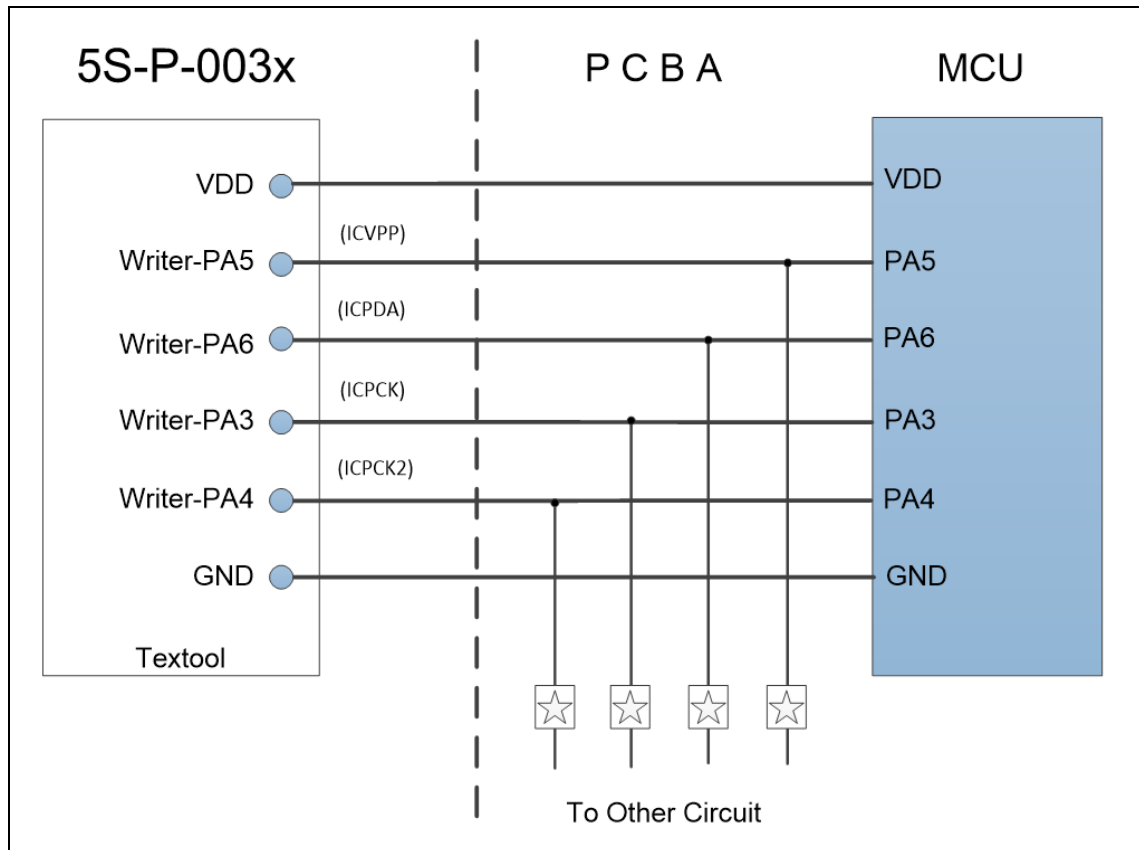


Fig. 32: Schematic Diagram of On-Board Wiring

The symbol ☆ on Fig. 32 can be either resistors or capacitors. They are used to isolate the programming signal wires from the peripheral circuit. it should be $\geq 10K\Omega$ for resistance while $\leq 220pF$ for capacitance.

Notice:

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

14. Device Characteristics

14.1. Absolute Maximum Ratings

Name	Min	Typ.	Max	Unit	Notes
Supply Voltage (VDD)	2.2		5.5	V	Exceed the maximum rating may cause permanent damage !!
Input Voltage	-0.3		$V_{DD} + 0.3$	V	
Operating Temperature	-40		85	°C	
Storage Temperature	-50		125	°C	
Junction Temperature		150		°C	

14.2. DC/AC Characteristics

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

Symbol	Description	Min	Typ	Max	Unit	Conditions (Ta=25°C)
V_{DD}	Operating Voltage	2.2 [#]	5.0	5.5	V	[#] Subject to LVR tolerance
LVR%	Low Voltage Reset Tolerance	-5		5	%	
f_{SYS}	System clock (CLK)* = IHRC/2 IHRC/4 IHRC/8 ILRC	0 0 0	 45K	8M 4M 2M	Hz	$V_{DD} \geq 4.0V$ $V_{DD} \geq 2.5V$ $V_{DD} \geq 2.2V$ $V_{DD} = 5.0V$
P_{cycle}	Program cycle	1000			cycles	
I_{OP}	Operating Current		1 65		mA uA	$f_{SYS}=IHRC/16=1MIPS@5V$ $f_{SYS}=ILRC=40KHz@5V$
I_{PD}	Power Down Current (by <i>stopsys</i> command)		1		uA	$f_{SYS}=0Hz$, $V_{DD}=5V$
I_{PS}	Power Save Current (by <i>stopexe</i> command)		3		uA	$V_{DD}=5V$; $f_{SYS}=ILRC$ Only ILRC module is enabled.
V_{IL}	Input low voltage for IO lines	0		$0.1 V_{DD}$	V	
V_{IH}	Input high voltage for IO lines	$0.7 V_{DD}$		V_{DD}	V	
I_{OL}	IO lines sink current PA0, PA3, PA4, PB2 PA5, PA6, PA7, PB0, PB1, PB3, PB5, PB6 PB4, PB7 (Strong) PB4, PB7 (Normal)		23 17 23 37 23		mA	$V_{DD}=5V$, $V_{OL}=0.5V$
I_{OH}	IO lines drive current PB4, PB7 (Normal) PB4, PB7 (Low) Others IO		-26 -10 -10		mA	$V_{DD}=5V$, $V_{OH}=4.5V$
V_{IN}	Input voltage	-0.3		$V_{DD} + 0.3$	V	
$I_{INJ} (PIN)$	Injected current on pin		1		uA	$V_{DD} + 0.3 \geq V_{IN} \geq -0.3$

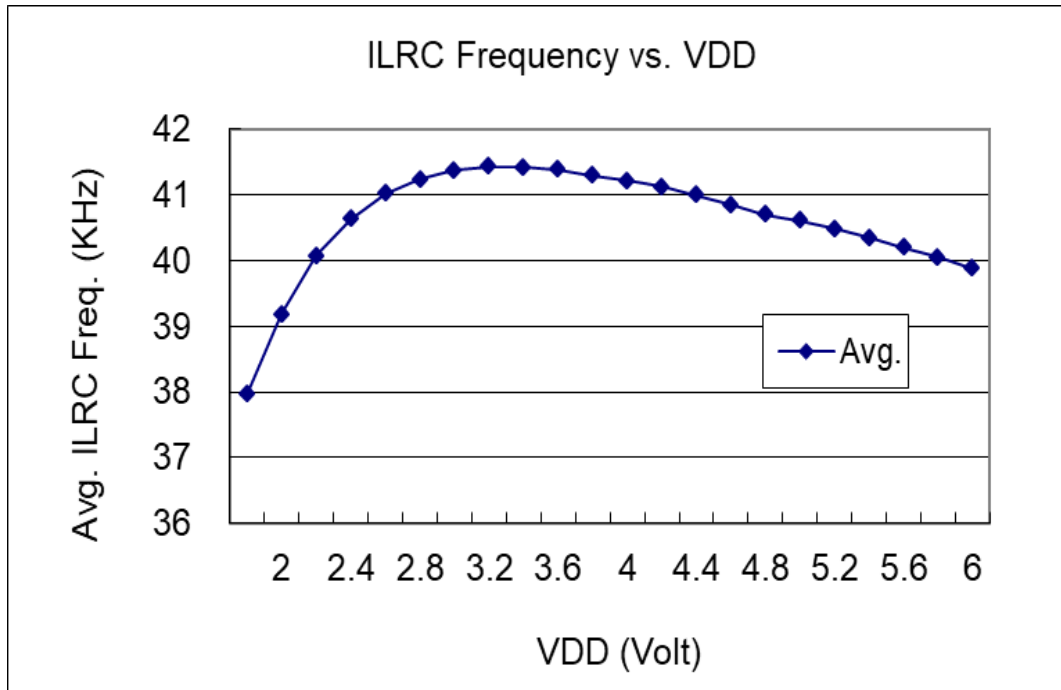
Symbol	Description	Min	Typ	Max	Unit	Conditions (Ta=25°C)
R _{PH}	Pull-high Resistance		82		KΩ	V _{DD} =5V
R _{PL}	Pull-low Resistance		82		KΩ	V _{DD} =5V
V _{BG}	Bandgap Reference Voltage	1.145*	1.20*	1.255*	V	V _{DD} =2.2V ~ 5.5V -40°C < Ta < 85°C*
f _{IHRC}	Frequency of IHRC after calibration *	15.84*	16*	16.16*	MHz	VDD=5V, Ta=25 °C
		15.20*	16*	16.80*		VDD =2.2V~5.5V, -40 °C <Ta<85 °C *
t _{INT}	Interrupt pulse width	30			ns	V _{DD} = 5V
V _{AD}	AD Input Voltage	0		V _{DD}	V	
ADrs	ADC resolution		12		bit	
ADcs	ADC current consumption		0.8 0.75		mA	@5V @3V
ADclk	ADC clock period		2		us	2.2V ~ 5.5V
t _{ADCONV}	ADC conversion time (T _{ADCLK} is the period of the selected AD conversion clock)		16		T _{ADCLK}	12-bit resolution
AD DNL	ADC Differential NonLinearity		±2*		LSB	
AD INL	ADC Integral NonLinearity		±4*		LSB	
ADos	ADC offset*		2		mV	V _{DD} =3V
V _{REFH}	ADC reference high voltage					V _{DD} =5V, 25 °C
	4V	3.90	4	4.10		
	3V	2.93	3	3.07		
	2V	1.95	2	2.05		
V _{DR}	RAM data retention voltage*	1.5			V	in power-down mode
t _{WDT}	Watchdog timeout period		8k		T _{ILRC}	misc[1:0]=00 (default)
			16k			misc[1:0]=01
			64k			misc[1:0]=10
			256k			misc[1:0]=11
t _{WUP}	Wake-up time period for fast wake-up		45		T _{ILRC}	Where T _{ILRC} is the time period of ILRC
	Wake-up time period for normal wake-up		3000			
t _{SBP}	System boot-up period from power-on		72		ms	V _{DD} =5V
t _{RST}	External reset pulse width	120			us	V _{DD} =5V

Symbol	Description	Min	Typ	Max	Unit	Conditions (Ta=25°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		20		uA	V _{DD} = 3.3V
OPAcn	OPA input common mode*	0		VDD-1.3	V	
OPAos	OPA offset*		±10		mV	V _{DD} =5V
I _{OPA}	OPA output current*	200			uA	
OPAgain	OPA DC gain*		80		dB	

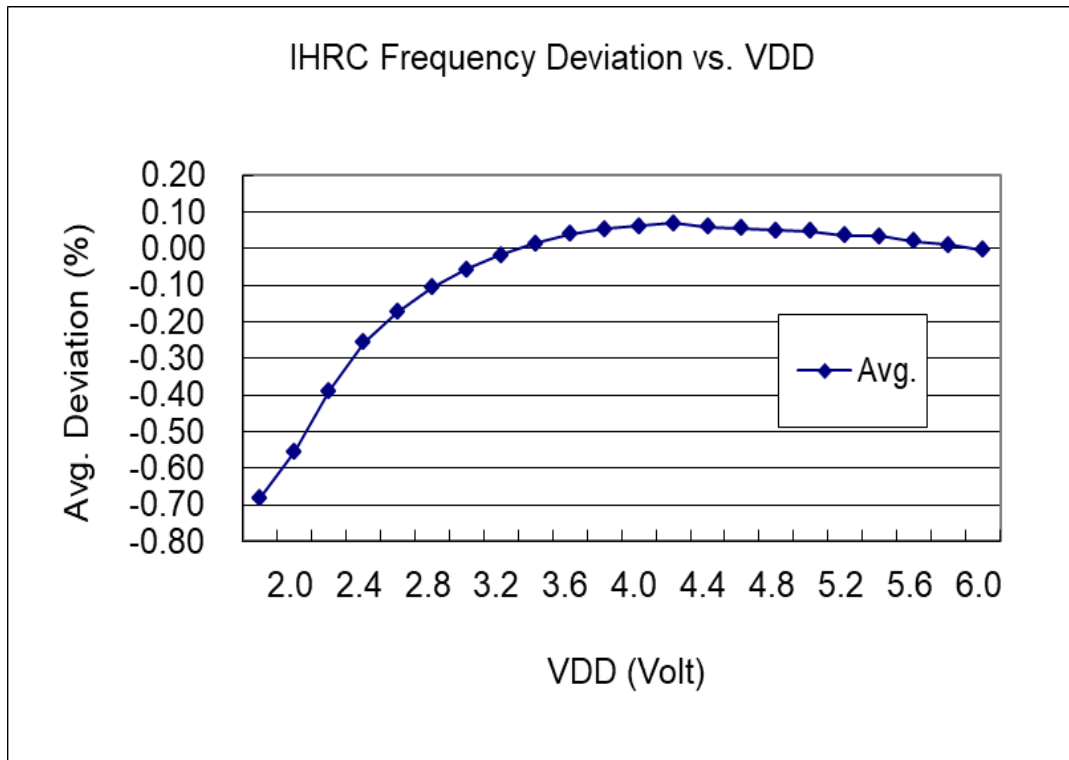
*These parameters are for design reference, not tested for every chip.

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

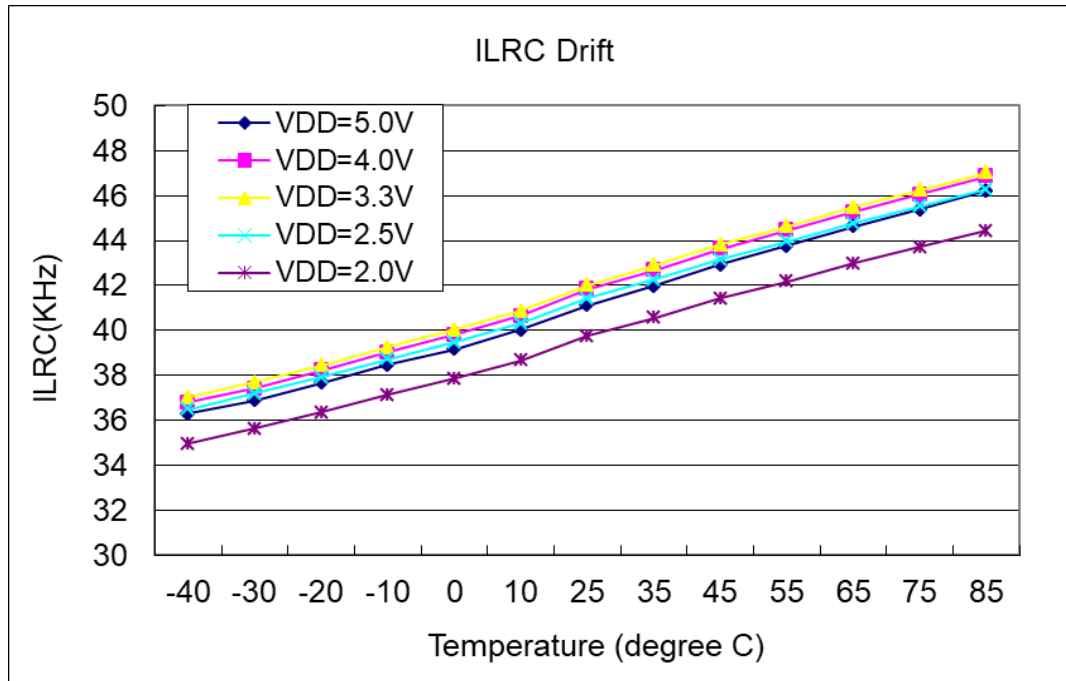
14.3. Typical ILRC frequency vs. VDD



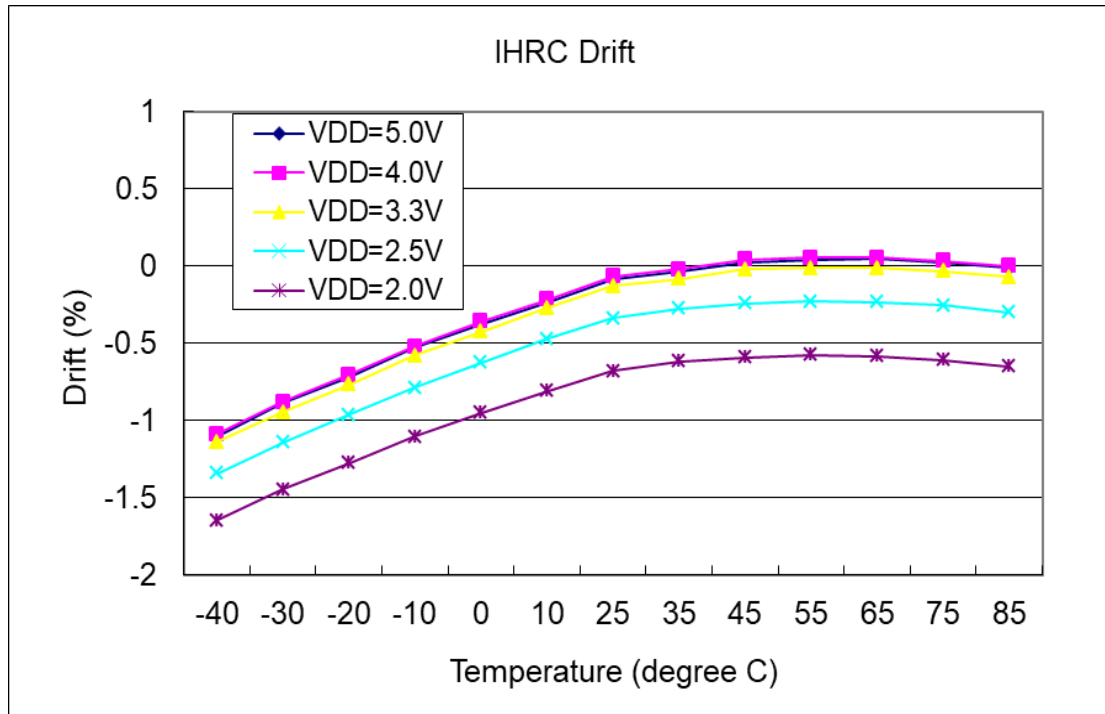
14.4. Typical IHRC frequency deviation vs. VDD(calibrated to 16MHz)



14.5. Typical ILRC Frequency vs. Temperature



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

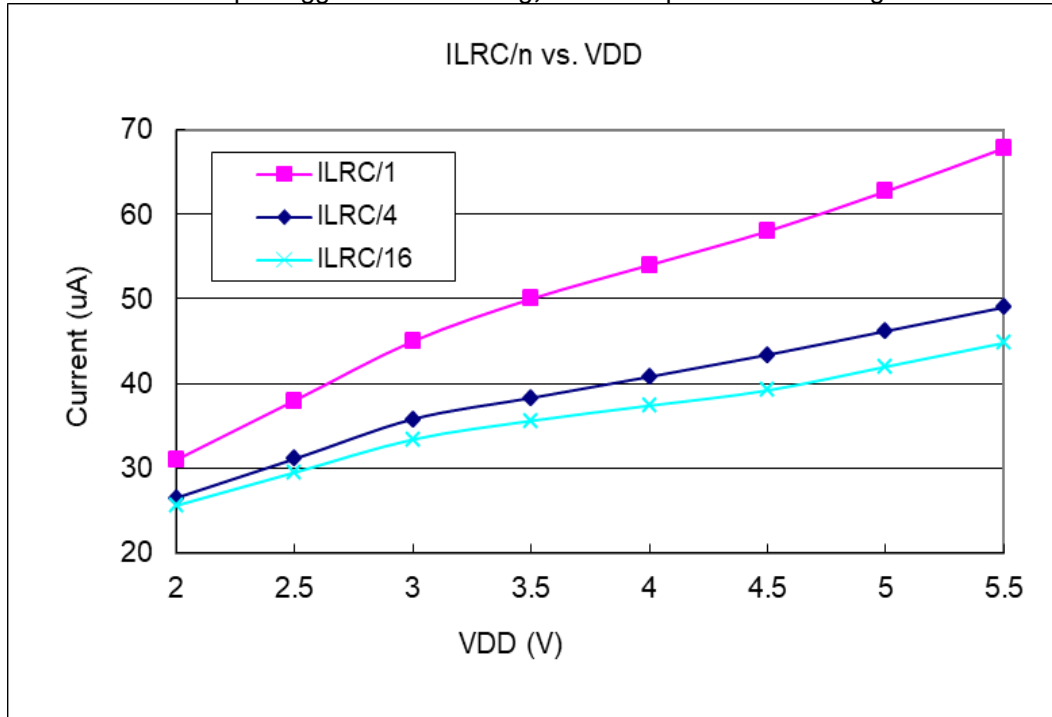


14.7. Typical operating current vs. VDD @ system clock = ILRC/n

Conditions: 2-FPPA (FPPA0: tog PA0, FPPA1: idle)

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

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

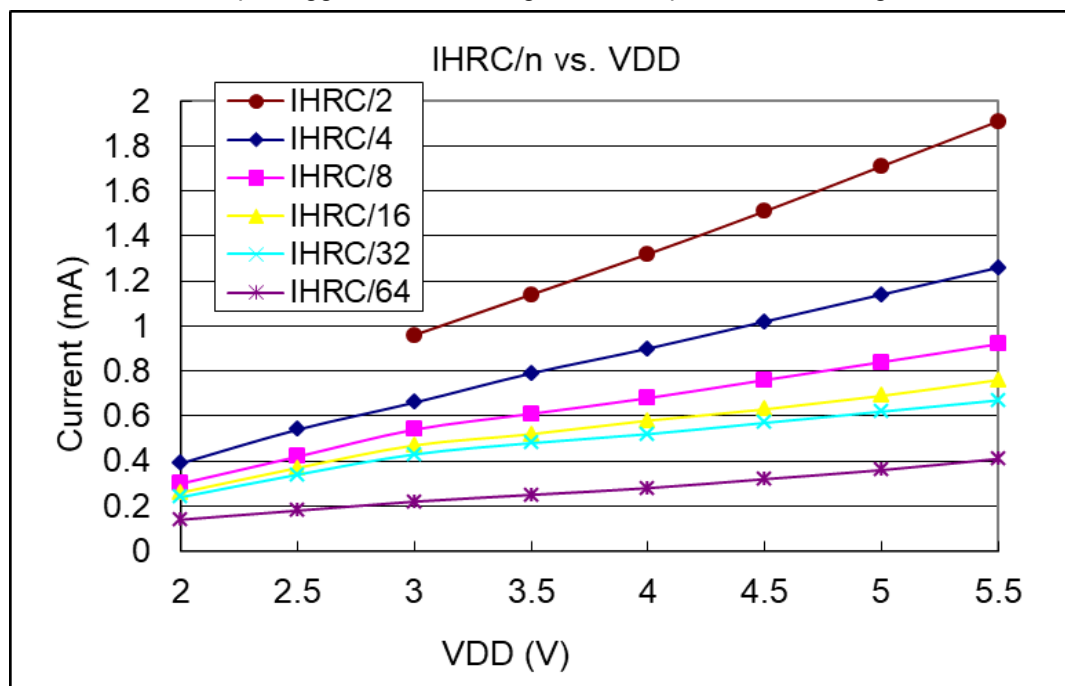


14.8. Typical operating current vs. VDD @ system clock = IHRC/n

Conditions: 2-FPPA (FPPA0: tog PA0, FPPA1: idle)

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

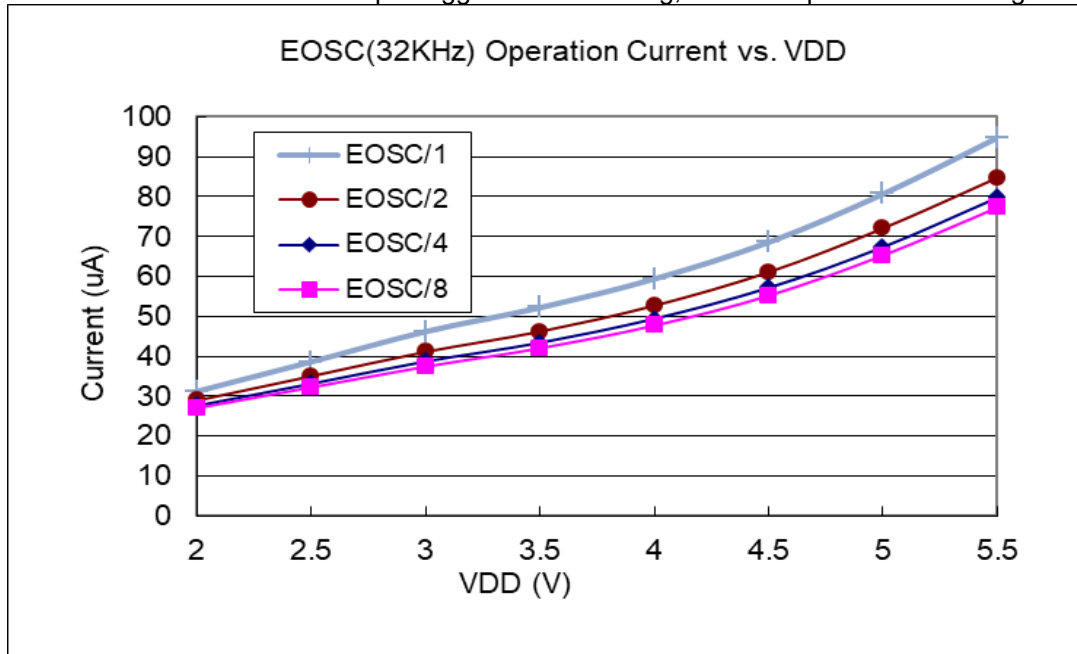
IO: PA0:0.5Hz output toggle and no loading, **others:** input and no floating



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

Conditions: **ON**: EOSC, MISC.6 = 1, Bandgap, LVR; **OFF**: IHRC, ILRC, T16, TM2, TM3, ADC modules;

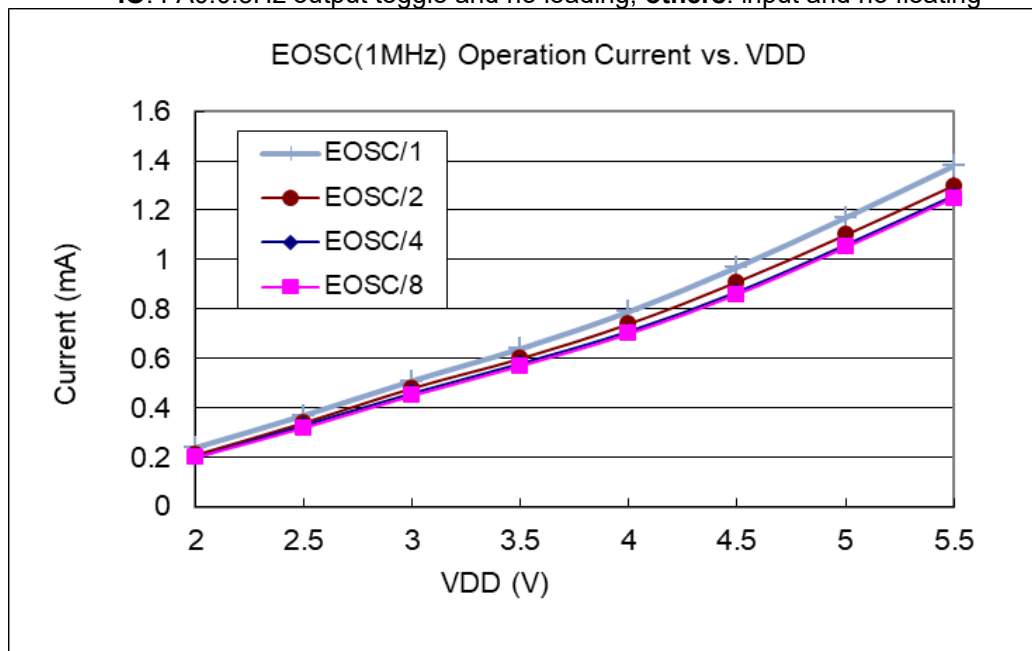
IO: PA0:0.5Hz output toggle and no loading, **others**: input and no floating



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

Conditions: **ON**: EOSC, MISC.6 = 1, Bandgap, LVR; **OFF**: IHRC, ILRC, T16, TM2, TM3, ADC modules;

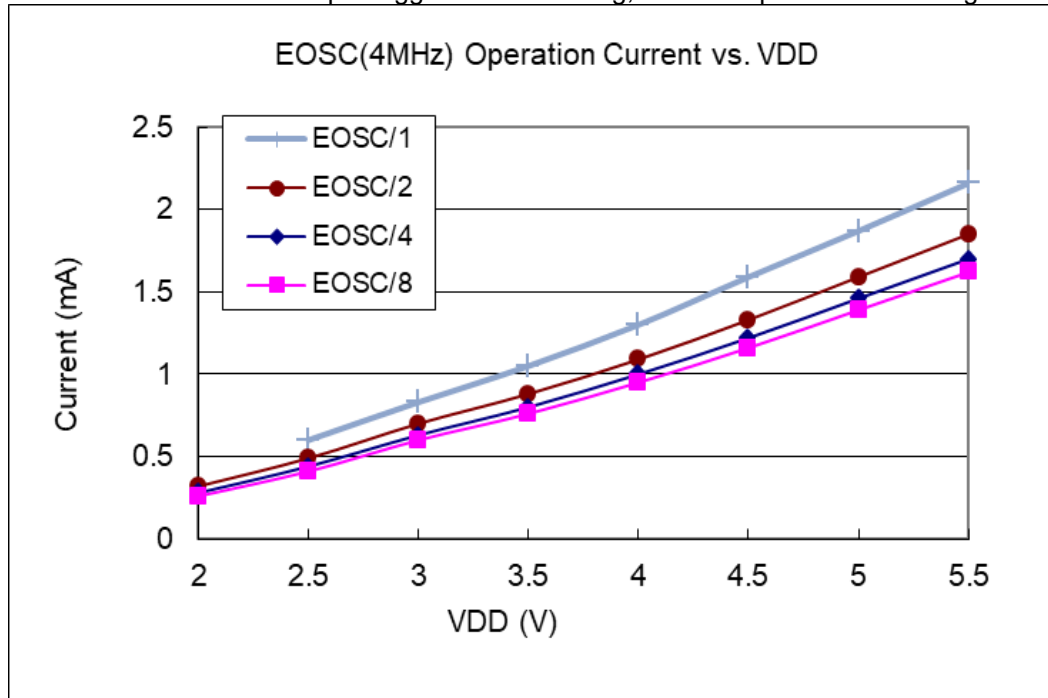
IO: PA0:0.5Hz output toggle and no loading, **others**: input and no floating



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

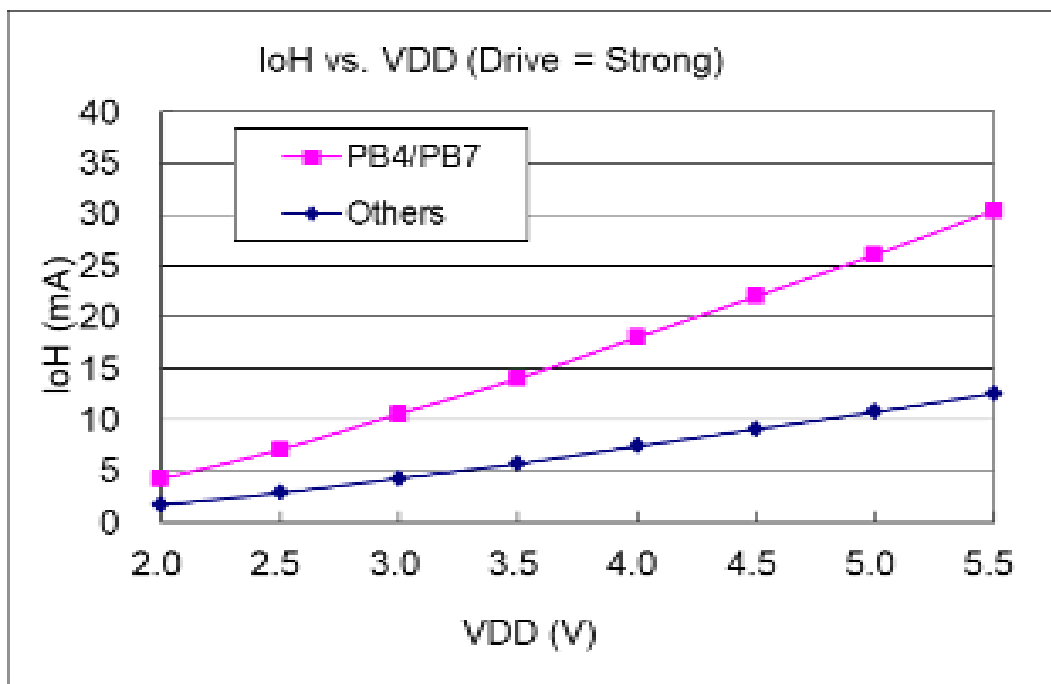
Conditions: **ON**: EOSC, MISC.6 = 1, Bandgap, LVR; **OFF**: IHRC, ILRC, T16, TM2, TM3, ADC modules;

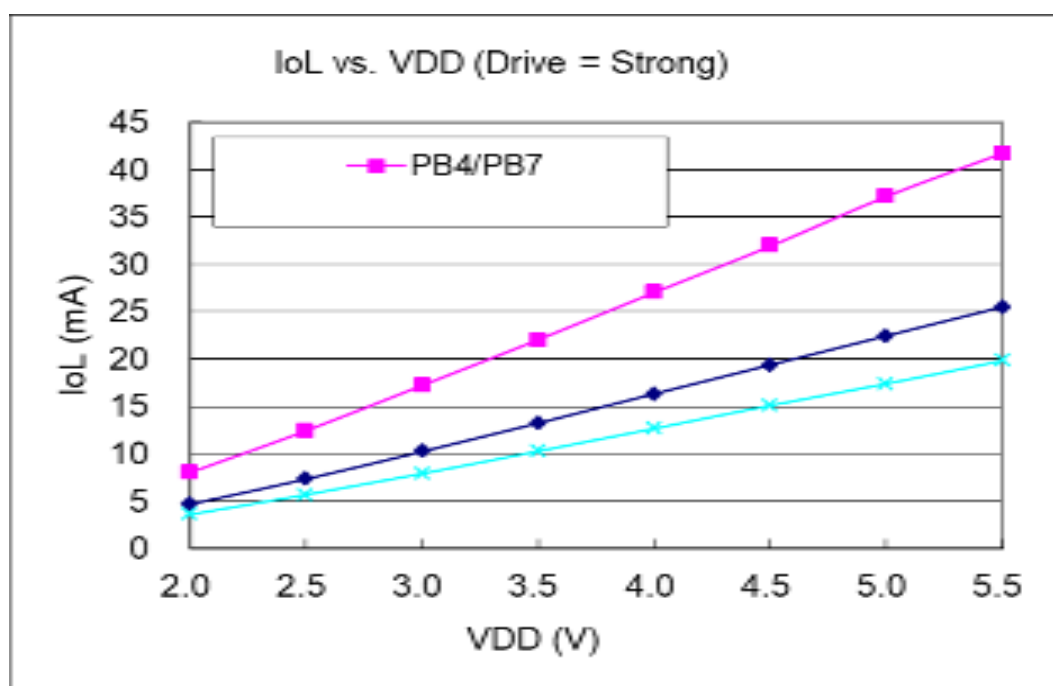
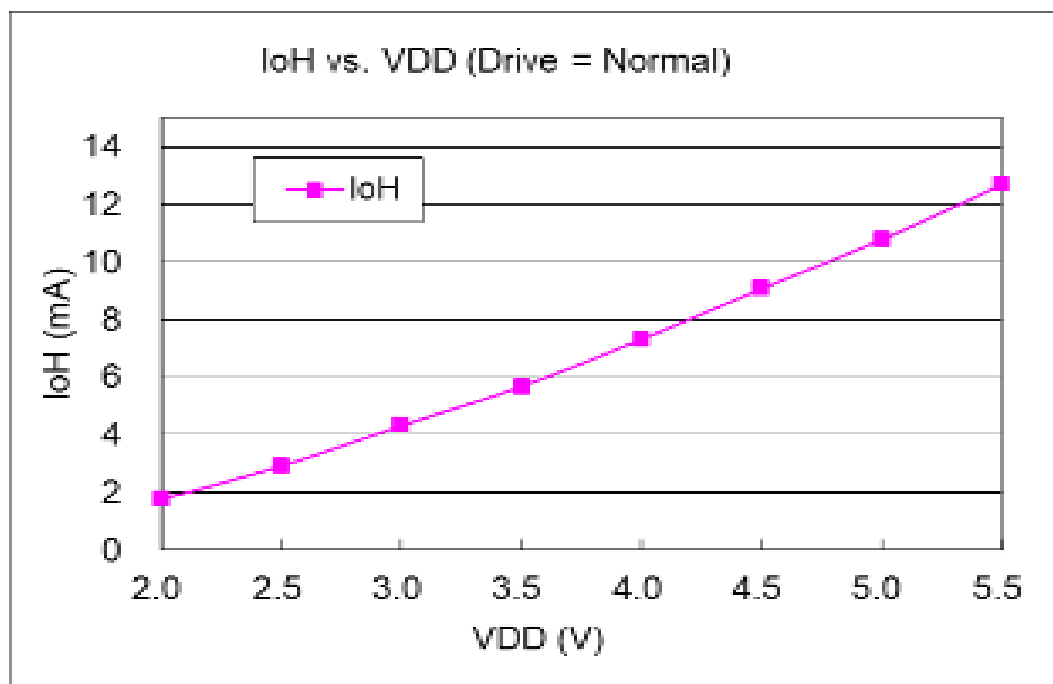
IO: PA0:0.5Hz output toggle and no loading, **others**: input and no floating

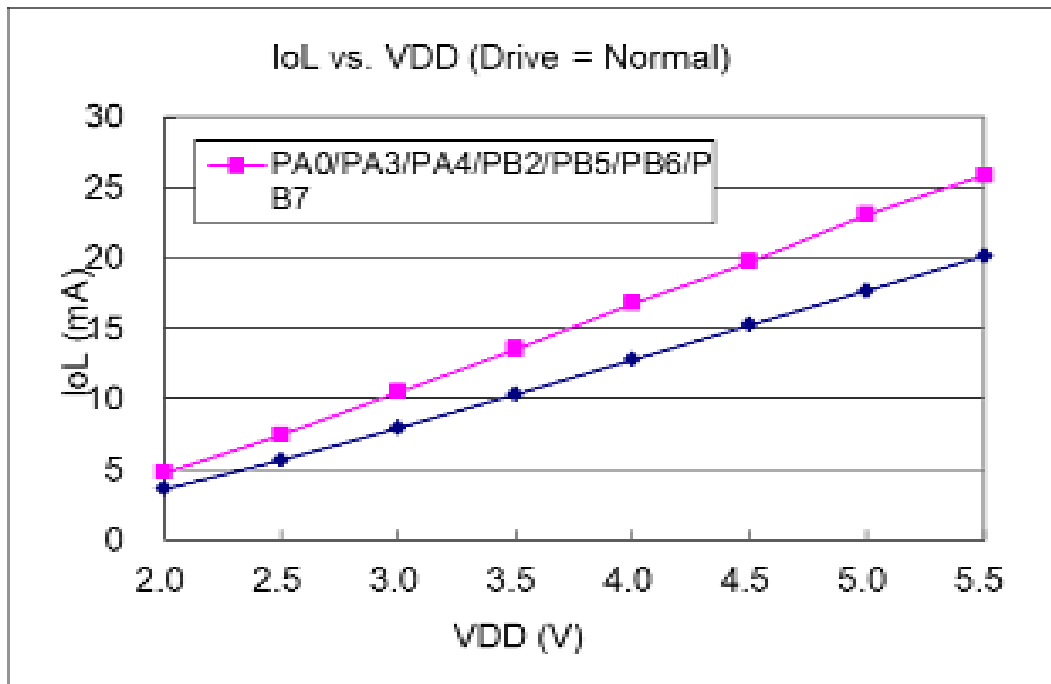


14.12. Typical IO driving current (I_{OH}) and sink current (I_{OL})

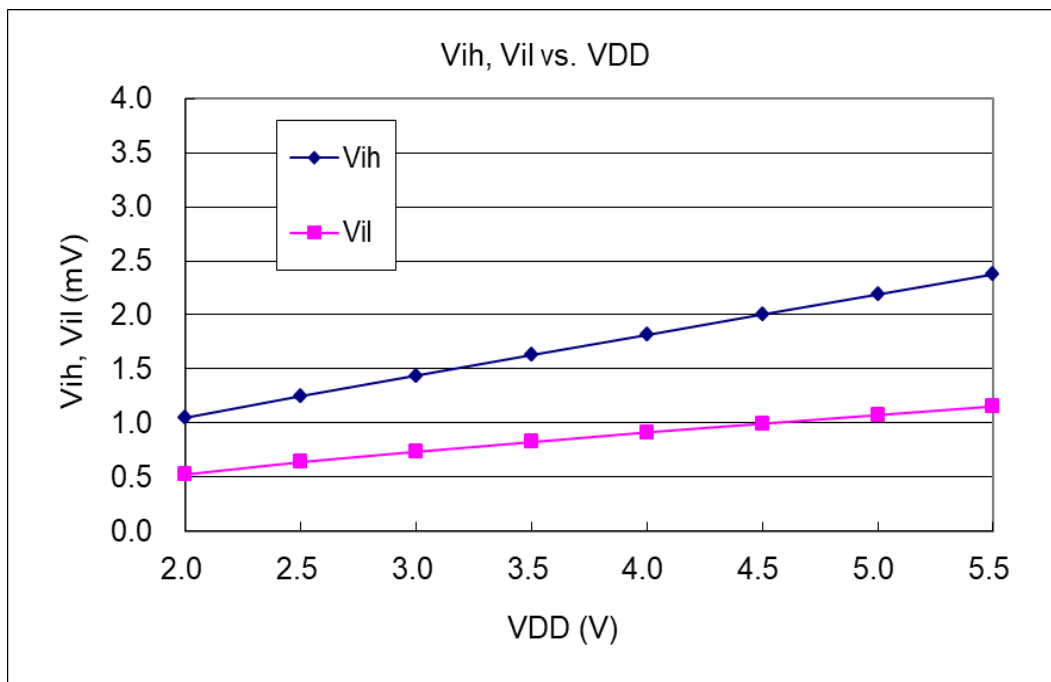
($V_{OH}=0.9 \cdot V_{DD}$, $V_{OL}=0.1 \cdot V_{DD}$)



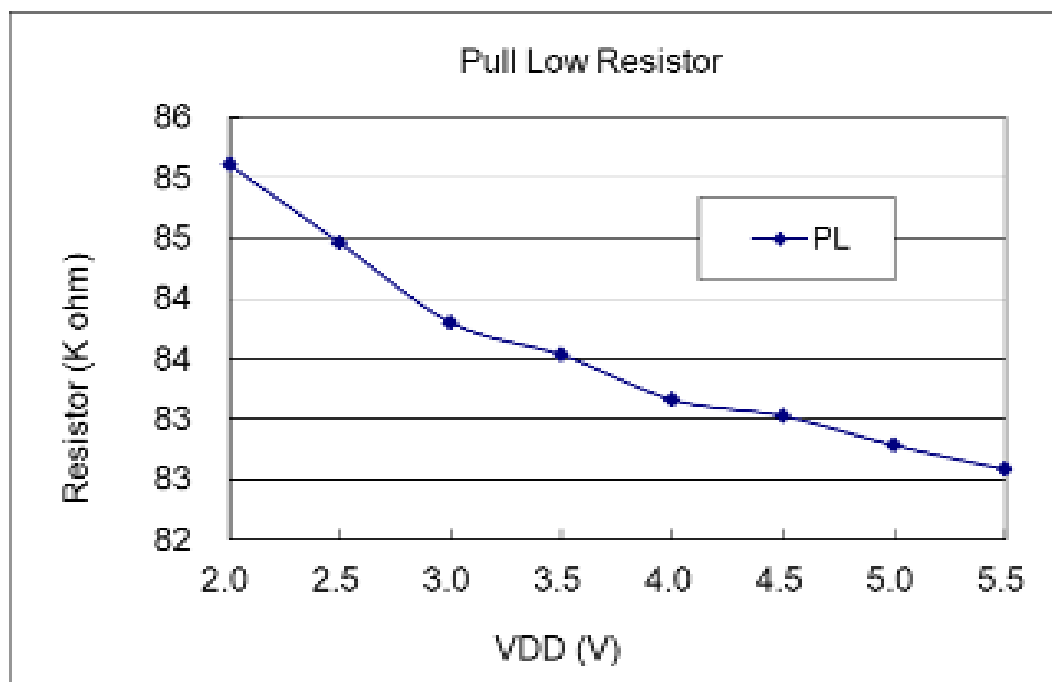
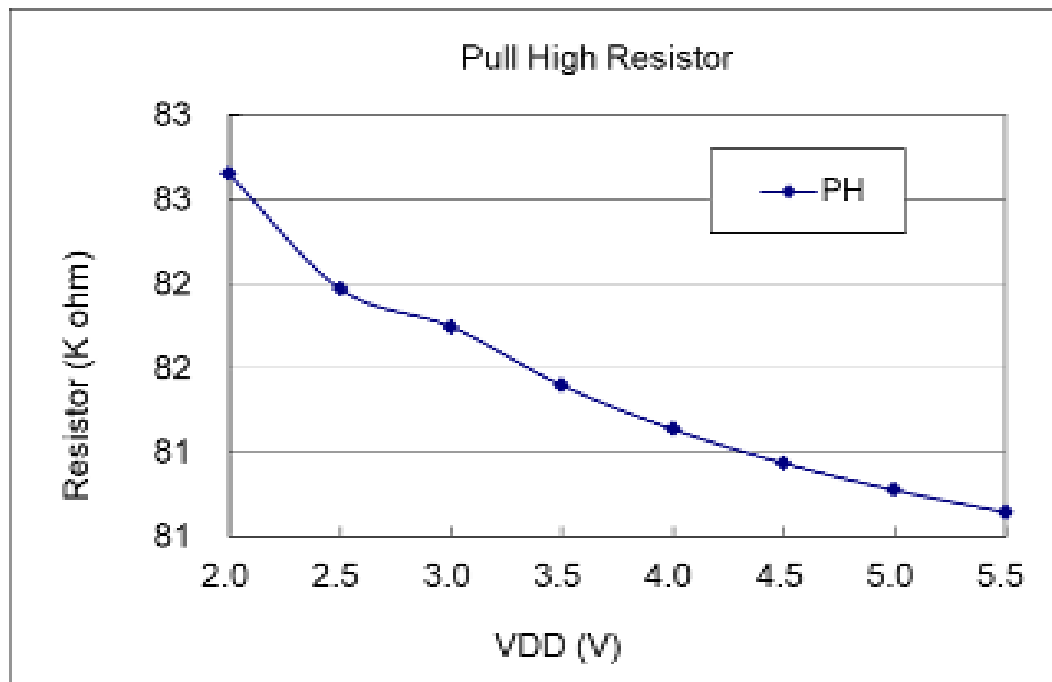




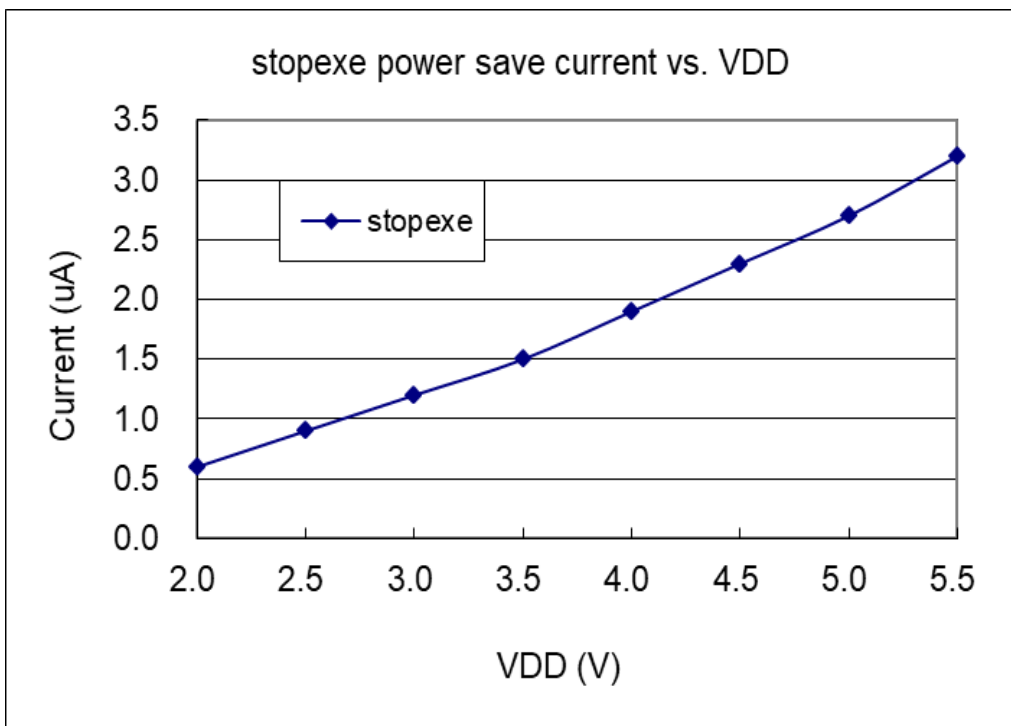
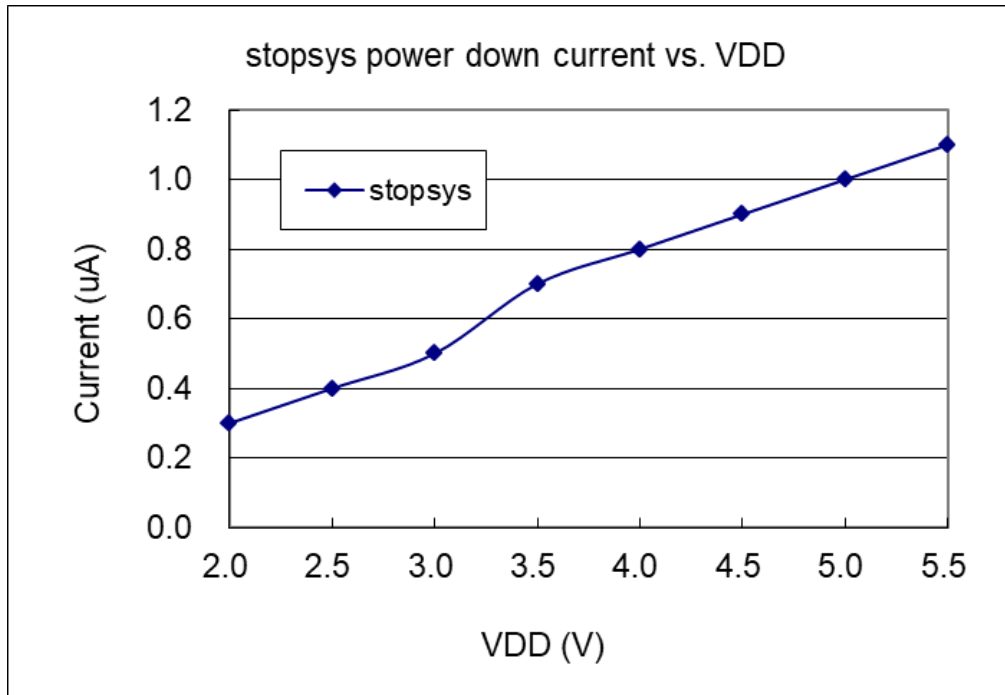
14.13. Typical IO input high/low threshold voltage (V_{IH}/V_{IL})



14.14. Typical resistance of IO pull high/low device



14.15. Typical power down current (I_{PD}) and power save current (I_{PS})



15. Instructions

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

15.1. Data Transfer Instructions

<i>mov</i> <i>a, l</i>	Move immediate data into ACC. Example: <i>mov a, 0x0f</i>; Result: $a \leftarrow 0fh$; Affected flags: 『N』 Z 『N』 C 『N』 AC 『N』 OV
<i>mov</i> <i>M, a</i>	Move data from ACC into memory Example: <i>mov MEM, a</i>; Result: $MEM \leftarrow a$ Affected flags: 『N』 Z 『N』 C 『N』 AC 『N』 OV
<i>mov</i> <i>a, M</i>	Move data from memory into ACC Example: <i>mov a, MEM</i> ; Result: $a \leftarrow MEM$; Flag Z is set when MEM is zero. Affected flags: 『Y』 Z 『N』 C 『N』 AC 『N』 OV
<i>mov</i> <i>a, IO</i>	Move data from IO into ACC Example: <i>mov a, pa</i> ; Result: $a \leftarrow pa$; Flag Z is set when pa is zero. Affected flags: 『Y』 Z 『N』 C 『N』 AC 『N』 OV
<i>mov</i> <i>IO, a</i>	Move data from ACC into IO Example: <i>mov pb, a</i>; Result: $pb \leftarrow a$ Affected flags: 『N』 Z 『N』 C 『N』 AC 『N』 OV
<i>nmov</i> <i>M, a</i>	Take the negative logic (2's complement) of ACC to put on memory Example: <i>mov MEM, a</i>; Result: $MEM \leftarrow \overline{a}$ Affected flags: 『N』 Z 『N』 C 『N』 AC 『N』 OV Application Example: <pre> mov a, 0xf5 ; // ACC is 0xf5 nmov ram9, a; // ram9 is 0x0b, ACC is 0xf5 </pre>
<i>nmov</i> <i>a, M</i>	Take the negative logic (2's complement) of memory to put on ACC Example: <i>mov a, MEM</i> ; Result: $a \leftarrow \overline{MEM}$; Flag Z is set when $\overline{MEM}$ is zero. Affected flags: 『Y』 Z 『N』 C 『N』 AC 『N』 OV Application Example: <pre> mov a, 0xf5 ; mov ram9, a ; // ram9 is 0xf5 nmov a, ram9 ; // ram9 is 0xf5, ACC is 0x0b </pre>
<i>ldtabh</i> <i>index</i>	Load high byte data in OTP program memory to ACC by using index as OTP address. It needs 2T to execute this instruction. Example: <i>ldtabh index</i>; Result: $a \leftarrow \{ \text{bit } 15 \sim 8 \text{ of OTP } [index] \}$; Affected flags: 『N』 Z 『N』 C 『N』 AC 『N』 OV Application Example: <pre> word ROMptr ; // declare a pointer of ROM in RAM ... </pre>

	<pre> 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 ; </pre>
<i>ldtbl index</i>	Load low byte data in OTP to ACC by using index as OTP address. It needs 2T to execute this instruction. Example: <code>ldtbl index;</code> Result: $a \leftarrow \{\text{bit7} \sim 0 \text{ of OTP } [\text{index}]\};$ Affected flags: $\text{『N』 Z } \text{『N』 C } \text{『N』 AC } \text{『N』 OV}$ Application Example: <pre> 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) ... ldtbl ROMptr ; // load TableA LSB to ACC (ACC=0x34) ... TableA : dc 0x0234, 0x0042, 0x0024, 0x0018 ; </pre>
<i>ldt16 word</i>	Move 16-bit counting values in Timer16 to memory in word. Example: <code>ldt16 word;</code> Result: $\text{word} \leftarrow \text{16-bit timer}$ Affected flags: $\text{『N』 Z } \text{『N』 C } \text{『N』 AC } \text{『N』 OV}$ Application Example: <pre> word T16val ; // declare a RAM word ... clear lb@ T16val ; // clear T16val (LSB) clear hb@ T16val ; // clear T16val (MSB) stt16 T16val ; // initial T16 with 0 ... set1 t16m.5 ; // enable Timer16 ... set0 t16m.5 ; // disable Timer16 ldt16 T16val ; // save the T16 counting value to T16val ... </pre>
<i>stt16 word</i>	Store 16-bit data from memory in word to Timer16. Example: <code>stt16 word;</code> Result: $\text{16-bit timer} \leftarrow \text{word}$ Affected flags: $\text{『N』 Z } \text{『N』 C } \text{『N』 AC } \text{『N』 OV}$

	Application Example: <pre> word T16val ; // declare a RAM word ... mov a, 0x34 ; mov lb@ T16val , a ; // move 0x34 to T16val (LSB) mov a, 0x12 ; mov hb@ T16val , a ; // move 0x12 to T16val (MSB) stt16 T16val ; // initial T16 with 0x1234 ... </pre>
<i>xch M</i>	Exchange data between ACC and memory Example: <code>xch MEM ;</code> Result: <code>MEM ← a , a ← MEM</code> Affected flags: 『N』 Z 『N』 C 『N』 AC 『N』 OV
<i>idxm a, index</i>	Move data from specified memory to ACC by indirect method. It needs 2T to execute this instruction. Example: <code>idxm a, index;</code> Result: <code>a ← [index]</code>, where index is declared by word. Affected flags: 『N』 Z 『N』 C 『N』 AC 『N』 OV Application Example: <pre> 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 </pre>
<i>ldxm index, a</i>	Move data from ACC to specified memory by indirect method. It needs 2T to execute this instruction. Example: <code>ldxm index, a;</code> Result: <code>[index] ← a</code>; where index is declared by word. Affected flags: 『N』 Z 『N』 C 『N』 AC 『N』 OV Application Example: <pre> 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 ; ldxm RAMIndex, a ; // mov 0xA5 to memory in address 0x5B </pre>

<i>pushaf</i>	Move the ACC and flag register to memory that address specified in the stack pointer. Example: <code>pushaf;</code> Result: $[sp] \leftarrow \{\text{flag}, \text{ACC}\};$ $sp \leftarrow sp + 2;$ Affected flags: 『N』 Z 『N』 C 『N』 AC 『N』 OV Application Example: <pre>.romadr 0x10 ; // ISR entry address pushaf ; // put ACC and flag into stack memory ... // ISR program ... // ISR program popaf ; // restore ACC and flag from stack memory reti ;</pre>
<i>popaf</i>	Restore ACC and flag from the memory which address is specified in the stack pointer. Example: <code>popaf;</code> Result: $sp \leftarrow sp - 2;$ $\{\text{Flag}, \text{ACC}\} \leftarrow [sp];$ Affected flags: 『Y』 Z 『Y』 C 『Y』 AC 『Y』 OV

15.2. Arithmetic Operation Instructions

<i>add a, l</i>	Add immediate data with ACC, then put result into ACC Example: <code>add a, 0x0f;</code> Result: $a \leftarrow a + 0fh$ Affected flags: 『Y』 Z 『Y』 C 『Y』 AC 『Y』 OV
<i>add a, M</i>	Add data in memory with ACC, then put result into ACC Example: <code>add a, MEM;</code> Result: $a \leftarrow a + \text{MEM}$ Affected flags: 『Y』 Z 『Y』 C 『Y』 AC 『Y』 OV
<i>add M, a</i>	Add data in memory with ACC, then put result into memory Example: <code>add MEM, a;</code> Result: $\text{MEM} \leftarrow a + \text{MEM}$ Affected flags: 『Y』 Z 『Y』 C 『Y』 AC 『Y』 OV
<i>addc a, M</i>	Add data in memory with ACC and carry bit, then put result into ACC Example: <code>addc a, MEM;</code> Result: $a \leftarrow a + \text{MEM} + C$ Affected flags: 『Y』 Z 『Y』 C 『Y』 AC 『Y』 OV
<i>addc M, a</i>	Add data in memory with ACC and carry bit, then put result into memory Example: <code>addc MEM, a;</code> Result: $\text{MEM} \leftarrow a + \text{MEM} + C$ Affected flags: 『Y』 Z 『Y』 C 『Y』 AC 『Y』 OV
<i>addc a</i>	Add carry with ACC, then put result into ACC Example: <code>addc a;</code> Result: $a \leftarrow a + C$ Affected flags: 『Y』 Z 『Y』 C 『Y』 AC 『Y』 OV
<i>addc M</i>	Add carry with memory, then put result into memory Example: <code>addc MEM;</code> Result: $\text{MEM} \leftarrow \text{MEM} + C$ Affected flags: 『Y』 Z 『Y』 C 『Y』 AC 『Y』 OV
<i>nadd a, M</i>	Add negative logic (2's complement) of ACC with memory Example: <code>nadd a, MEM;</code> Result: $a \leftarrow \overline{a} + \text{MEM}$ Affected flags: 『Y』 Z 『Y』 C 『Y』 AC 『Y』 OV

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

15.3. Shift Operation Instructions

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

15.4. Logic Operation Instructions

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

	Example: <code>xor MEM, a ;</code> Result: $MEM \leftarrow a \wedge MEM$ Affected flags: 『Y』 Z 『N』 C 『N』 AC 『N』 OV
<i>not a</i>	Perform 1's complement (logical complement) of ACC Example: <code>not a ;</code> Result: $a \leftarrow \sim a$ Affected flags: 『Y』 Z 『N』 C 『N』 AC 『N』 OV Application Example: <pre> mov a, 0x38 ; // ACC=0X38 not a ; // ACC=0XC7 </pre>
<i>not M</i>	Perform 1's complement (logical complement) of memory Example: <code>not MEM ;</code> Result: $MEM \leftarrow \sim MEM$ Affected flags: 『Y』 Z 『N』 C 『N』 AC 『N』 OV Application Example: <pre> mov a, 0x38 ; mov mem, a ; // mem = 0x38 not mem ; // mem = 0xC7 </pre>
<i>neg a</i>	Perform 2's complement of ACC Example: <code>neg a ;</code> Result: $a \leftarrow \overline{a}$ Affected flags: 『Y』 Z 『N』 C 『N』 AC 『N』 OV Application Example: <pre> mov a, 0x38 ; // ACC=0X38 neg a ; // ACC=0XC8 </pre>
<i>neg M</i>	Perform 2's complement of memory Example: <code>neg MEM ;</code> Result: $MEM \leftarrow \overline{MEM}$ Affected flags: 『Y』 Z 『N』 C 『N』 AC 『N』 OV Application Example: <pre> mov a, 0x38 ; mov mem, a ; // mem = 0x38 neg mem ; // mem = 0xC8 </pre>

<i>comp</i> <i>a, I</i>	Compare ACC with immediate data Example: <code>comp a, 0x55;</code> Result: Flag will be changed by regarding as ($a - 0x55$) Affected flags: 『Y』 Z 『Y』 C 『Y』 AC 『Y』 OV Application Example: <pre> 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 </pre>
<i>comp</i> <i>a, M</i>	Compare ACC with the content of memory Example: <code>comp a, MEM;</code> Result: Flag will be changed by regarding as ($a - \text{MEM}$) Affected flags: 『Y』 Z 『Y』 C 『Y』 AC 『Y』 OV Application Example: <pre> 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 </pre>
<i>comp</i> <i>M, a</i>	Compare ACC with the content of memory Example: <code>comp MEM, a;</code> Result: Flag will be changed by regarding as ($\text{MEM} - a$) Affected flags: 『Y』 Z 『Y』 C 『Y』 AC 『Y』 OV

15.5. Bit Operation Instructions

<i>set0</i> <i>IO.n</i>	Set bit n of IO port to low Example: <code>set0 pa.5 ;</code> Result: set bit 5 of port A to low Affected flags: 『N』 Z 『N』 C 『N』 AC 『N』 OV
<i>set1</i> <i>IO.n</i>	Set bit n of IO port to high Example: <code>set1 pb.5 ;</code> Result: set bit 5 of port B to high Affected flags: 『N』 Z 『N』 C 『N』 AC 『N』 OV
<i>tog</i> <i>IO.n</i>	Toggle bit state of bit n of IO port Example: <code>tog pa.5 ;</code> Result: toggle bit 5 of port A Affected flags: 『N』 Z 『N』 C 『N』 AC 『N』 OV

<i>set0</i> M.n	Set bit n of memory to low Example: <i>set0 MEM.5</i> ; Result: set bit 5 of MEM to low Affected flags: 『N』 Z 『N』 C 『N』 AC 『N』 OV
<i>set1</i> M.n	Set bit n of memory to high Example: <i>set1 MEM.5</i> ; Result: set bit 5 of MEM to high Affected flags: 『N』 Z 『N』 C 『N』 AC 『N』 OV
<i>swapc</i> IO.n	Swap the nth bit of IO port with carry bit Example: <i>swapc IO.0</i> ; Result: $C \leftarrow IO.0$, $IO.0 \leftarrow C$ When IO.0 is a port to output pin, carry C will be sent to IO.0; When IO.0 is a port from input pin, IO.0 will be sent to carry C; Affected flags: 『N』 Z 『Y』 C 『N』 AC 『N』 OV Application Example1 (serial output) : <pre> ... set1 pac.0 ; // set PA.0 as output ... set0 flag.1 ; // C=0 swapc pa.0 ; // move C to PA.0 (bit operation), PA.0=0 set1 flag.1 ; // C=1 swapc pa.0 ; // move C to PA.0 (bit operation), PA.0=1 ... </pre>
	Application Example2 (serial input) : <pre> ... set0 pac.0 ; // set PA.0 as input ... swapc pa.0 ; // read PA.0 to C (bit operation) src a ; // shift C to bit 7 of ACC swapc pa.0 ; // read PA.0 to C (bit operation) src a ; // shift new C to bit 7, old C ... </pre>

15.6. Conditional Operation Instructions

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

	Affected flags: 『N』 Z 『N』 C 『N』 AC 『N』 OV
<i>izsn a</i>	Increment ACC and skip next instruction if ACC is zero Example: <i>izsn a</i> ; Result: $a \leftarrow a + 1$, skip next instruction if $a = 0$ Affected flags: 『Y』 Z 『Y』 C 『Y』 AC 『Y』 OV
<i>dzsn a</i>	Decrement ACC and skip next instruction if ACC is zero Example: <i>dzsn a</i> ; Result: $A \leftarrow A - 1$, skip next instruction if $a = 0$ Affected flags: 『Y』 Z 『Y』 C 『Y』 AC 『Y』 OV
<i>izsn M</i>	Increment memory and skip next instruction if memory is zero Example: <i>izsn MEM</i> ; Result: $MEM \leftarrow MEM + 1$, skip next instruction if $MEM = 0$ Affected flags: 『Y』 Z 『Y』 C 『Y』 AC 『Y』 OV
<i>dzsn M</i>	Decrement memory and skip next instruction if memory is zero Example: <i>dzsn MEM</i> ; Result: $MEM \leftarrow MEM - 1$, skip next instruction if $MEM = 0$ Affected flags: 『Y』 Z 『Y』 C 『Y』 AC 『Y』 OV
<i>wait0 IO.n</i>	Wait here until bit n of IO port is low. Example: <i>wait0 pa.5</i> ; Result: Wait bit 5 of port A low to execute next instruction; Affected flags: 『N』 Z 『N』 C 『N』 AC 『N』 OV
<i>wait1 IO.n</i>	Wait here until bit n of IO port is low. Example: <i>wait1 pa.5</i> ; Result: Wait bit 5 of port A high to execute next instruction; Affected flags: 『N』 Z 『N』 C 『N』 AC 『N』 OV

15.7. System control Instructions

<i>call label</i>	Function call, address can be full range address space Example: <i>call function1</i> ; Result: $[sp] \leftarrow pc + 1$ $pc \leftarrow function1$ $sp \leftarrow sp + 2$ Affected flags: 『N』 Z 『N』 C 『N』 AC 『N』 OV
<i>goto label</i>	Go to specific address which can be full range address space Example: <i>goto error</i> ; Result: Go to error and execute program. Affected flags: 『N』 Z 『N』 C 『N』 AC 『N』 OV
<i>delay a</i>	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 <i>delay</i> instruction is executed, the ACC will be zero. Example: <i>delay a</i> ; Result: Delay 16 cycles here if $ACC = 0fh$

	Affected flags: 『N』 Z 『N』 C 『N』 AC 『N』 OV 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.
<i>delay I</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 <i>delay</i> instruction is executed, the ACC will be zero. Example: <i>delay 0x05;</i> Result: Delay 6 cycles here Affected flags: 『N』 Z 『N』 C 『N』 AC 『N』 OV 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.
<i>delay M</i>	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 <i>delay</i> instruction is executed, the ACC will be zero. Example: <i>delay M;</i> Result: Delay 256 cycles here if M=ffh Affected flags: 『N』 Z 『N』 C 『N』 AC 『N』 OV 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.
<i>ret I</i>	Place immediate data to ACC, then return Example: <i>ret 0x55;</i> Result: $A \leftarrow 55h$ <i>ret ;</i> Affected flags: 『N』 Z 『N』 C 『N』 AC 『N』 OV
<i>ret</i>	Return to program which had function call Example: <i>ret;</i> Result: $sp \leftarrow sp - 2$ $pc \leftarrow [sp]$ Affected flags: 『N』 Z 『N』 C 『N』 AC 『N』 OV
<i>reti</i>	Return to program that is interrupt service routine. After this command is executed, global interrupt is enabled automatically. Example: <i>reti;</i> Affected flags: 『N』 Z 『N』 C 『N』 AC 『N』 OV
<i>nop</i>	No operation Example: <i>nop;</i> Result: nothing changed Affected flags: 『N』 Z 『N』 C 『N』 AC 『N』 OV
<i>pcadd a</i>	Next program counter is current program counter plus ACC. Example: <i>pcadd a;</i> Result: $pc \leftarrow pc + a$ Affected flags: 『N』 Z 『N』 C 『N』 AC 『N』 OV Application Example: <pre> ----- ... mov a, 0x02 ; pcadd a ; // PC <- PC+2 goto err1 ; goto correct ; // jump here </pre>

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

15.8. Summary of Instructions Execution Cycle

Instruction	Condition	1 FPPA	2 FPPA
<i>goto, call</i>		2T	1T
<i>ceqsn, cneqsn, t0sn, t1sn, dzsn, izsn</i>	Condition is fulfilled	2T	1T
	Condition is not fulfilled	1T	1T
<i>ldtabh, ldtabl, idxm, pcadd, ret, reti</i>		2T	2T
Others		1T	1T