PFC460

Industrial Grade -

24 Touch Keys 8-bit MTP MCU (FPPA™)

Datasheet

Version 0.08

January 23, 2026

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

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

IMPORTANT NOTICE

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

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

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

Table of Contents

Revision History

Revision	Data	Description
0.06	2024/09/20	1. Modify the value of DC/AC Characteristics 2. Updated 14.4 and 14.8 temperature drift curves
0.07	2025/12/10	Updated description of Section 15
0.08	2026/01/23	1. Section 1.2.: Added note: All PWMG source clock rates are limited to 32 MHz when VDD ≥ 3.0 V. 2. Section 14.2.: Added description for symbol fpwm.

Usage Warning

- There can be no overvoltage input (greater than the chip VDD voltage) at all IO pins of the chip, which will cause interference to the touch and cause abnormal touch.
- 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=159&kw=460

  (The following picture are for reference only.)

◆◆ PFC460 ◆◆

- High EFT series
- Operating temperature range: -40°C ~ 85°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		
APN015	Capacitive touch screen PCB design guide		
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

- 4KW MTP program memory for four FPPA units (programming cycle at least 1,000 times)
- 512 Bytes data SRAM for four FPPA units
- One hardware 16-bit timer
- Two hardware 8-bit timers with PWM generation (Timer2/Timer3), Timer2/Timer3 also configured with NILRC oscillator. Its frequency is slower than ILRC and suitable for a more power-saving wake-up clock.
- Timer2/Timer3 PWM resolution is 6/7/8 bit
- Three hardware 11-bit PWM generators (PWMG0/PWMG1/PWMG2)
  (Note: All the PWMG source clock rate can be 32MHz only when VDD≧3.0V.)
- One set triple 11bit SuLED (Super LED) PWM generators and timers(LPWMG0/LPWMG1/LPWMG2)
- Provide one PFG hardware circuit for precise frequency output
- Provide one hardware comparator
- Provide one OP Amplifier (OPA)
- Provide 1T 8x8 hardware multiplier
- 26 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
- Up to 24 IO pins could be selected as touch keys
- Built-in LDO hardware circuit provides 2V voltage reference for touch function
- Bandgap circuit to provide 1.20V reference voltage
- Up to 28-channel 12-bit resolution ADC with one channel comes from internal Bandgap reference voltage or 0.25*V_{DD}
- Provide ADC reference high voltage: external input, internal V_{DD}, Bandgap(1.20V), 4V, 3V, 2V
- Clock sources: internal high RC oscillator (IHRC), internal low RC oscillator (ILRC) and external crystal oscillator (EOSC)
- Provide four IO output capabilities to satisfy different application requirements
  (1) PB0 drive current could option 0/10mA, and it sink current could option 108/20mA
  (2) PB2~PB7 drive current could option 28/10mA, and it sink current could option 75/20mA
  (3) PA0~PA4 drive/sink current =10mA/20mA
  (4) PA5~PA7, PB1, PC0~PC7, PD0~PD1 drive/sink current =10mA/14mA
- Built-in VDD/2 bias voltage generator to provide 5COM×21SEG dots LCD display
- 14 selectable levels of LVR reset from 2.0V to 4.5V
- 8 selectable external interrupt pins

1.3. CPU Features

- Operating modes: Four processing units FPPA™ mode or Traditional one processing unit mode
- 104 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, SRAM memory space and MTP space are independent

1.4. Ordering/ Package Information

- PFC460-S08: SOP8 (150mil);
- PFC460-S14: SOP14 (150mil);
- PFC460-S16: SOP16 (150mil);
- PFC460-S20: SOP20 (300mil);
- PFC460-H20: HTSOP20 (150mil);
- PFC460-T20: TSSOP20 (173mil);
- PFC460-S24: SOP24 (300mil);
- PFC460-Y24: SSOP24 (150mil);
- PFC460-S28: SOP28 (300mil);
- PFC460-T28: TSSOP28 (173mil);
- PFC460-Y24: SSOP24 (150mil);

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

2. General Description and Block Diagram

The PFC460 family is an ADC-Type of PADAUK's parallel processing with touch function, fully static, MTP-based CMOS 8 bit*2 processor array that can execute four 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.

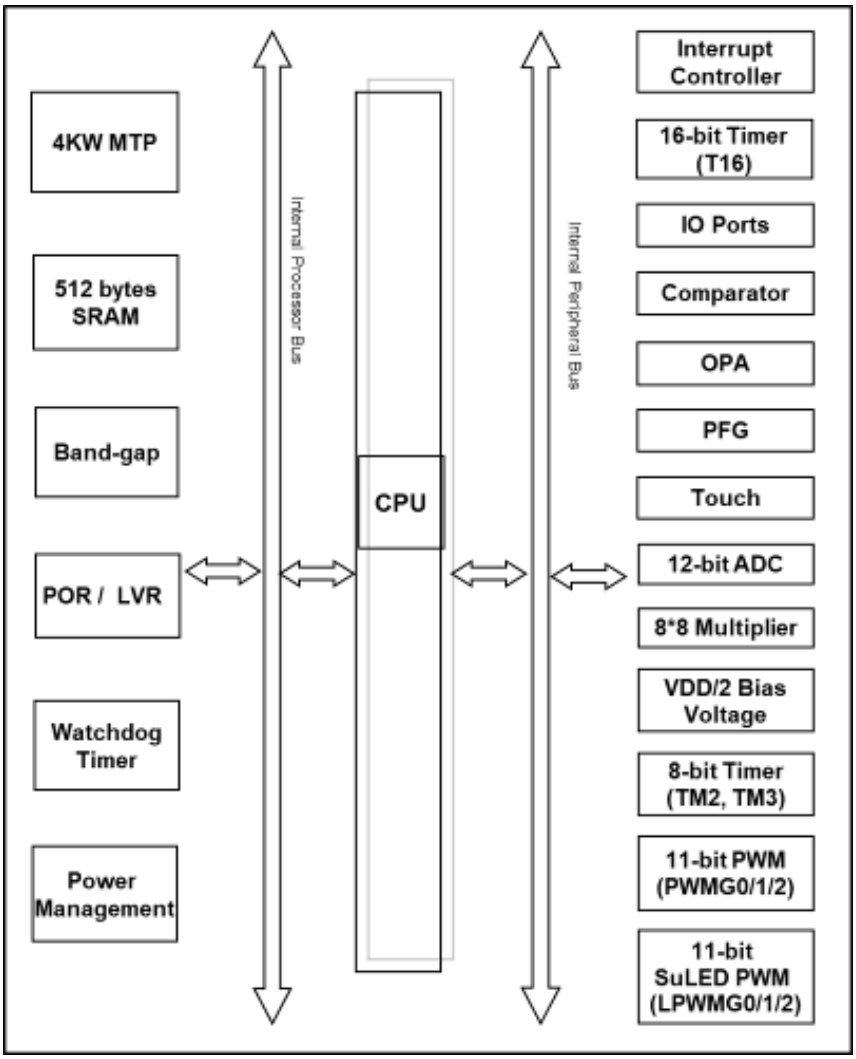
4KW MTP program memory and 512 bytes data SRAM are inside for PFC460.

One up to 28 channels (including GND) 12-bit ADC is built inside the chip with one channel for internal Bandgap reference voltage or 0.25*V_{DD}.

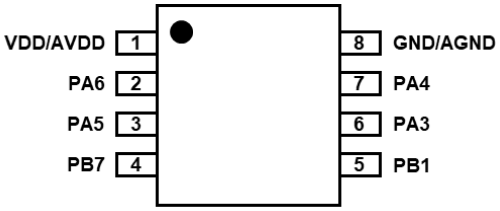
C-Touch hardware circuit is built inside the PFC460, which up to 24 IO pins could be selected as touch keys.

PFC460 provides a hardware 16-bit timer, two hardware 8-bit timers with PWM generation (Timer2, Timer3), three hardware 11-bit timers with PWM generation (PWMG0/1/2) and one new triple 11-bit timer with SuLED PWM generation (LPWMG0/1/2).

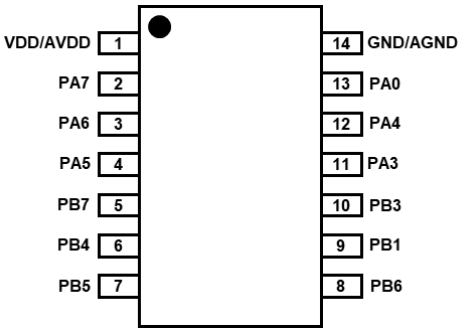
PFC460 also supports a PFG frequency modulation circuit, an Operational Amplifier (OPA) and a hardware comparator, plus one VDD/2 bias voltage generator for LCD display application and one 8x8 hardware multiplier to enhance hardware capability in arithmetic function.



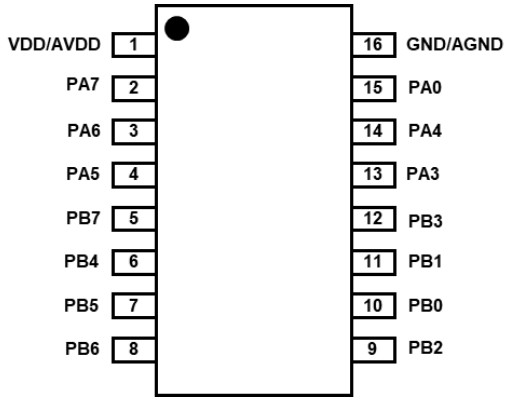
3. Pin Definition and Functional Description



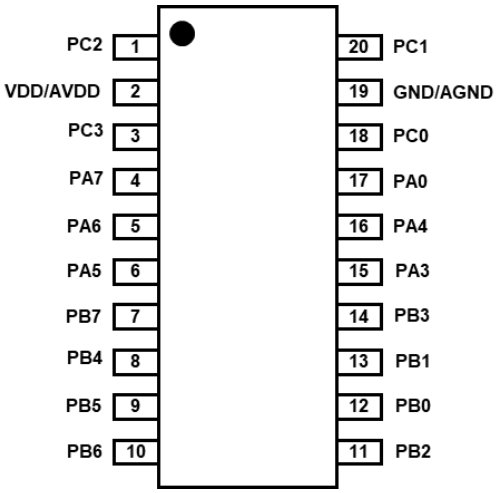
PFC460-S08: SOP8 (150mil)



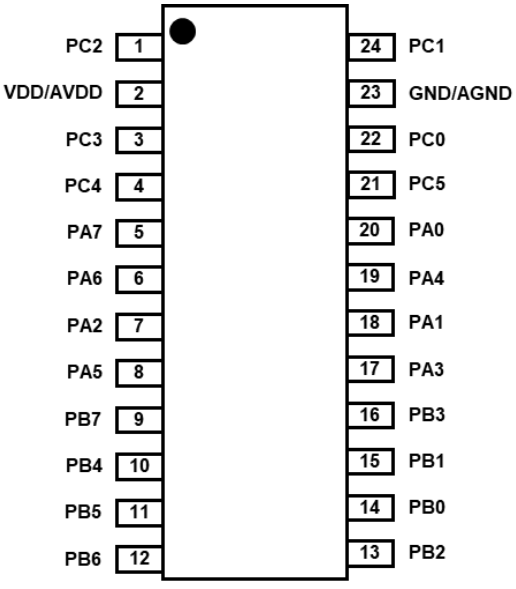
PFC460-S14: SOP14 (150mil)



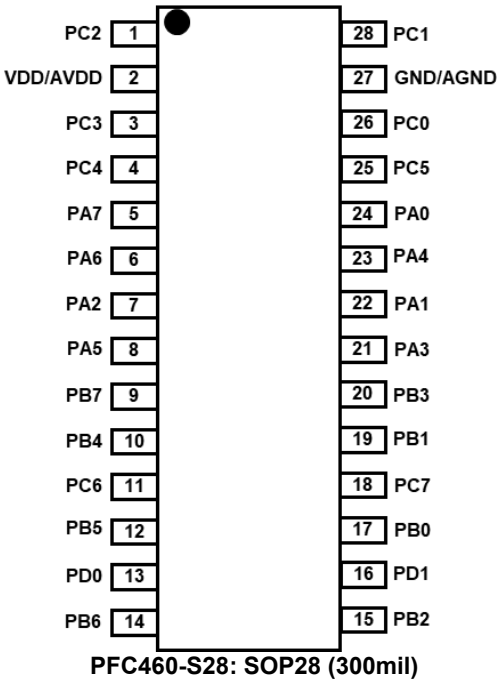
PFC460-S16: SOP16 (150mil)



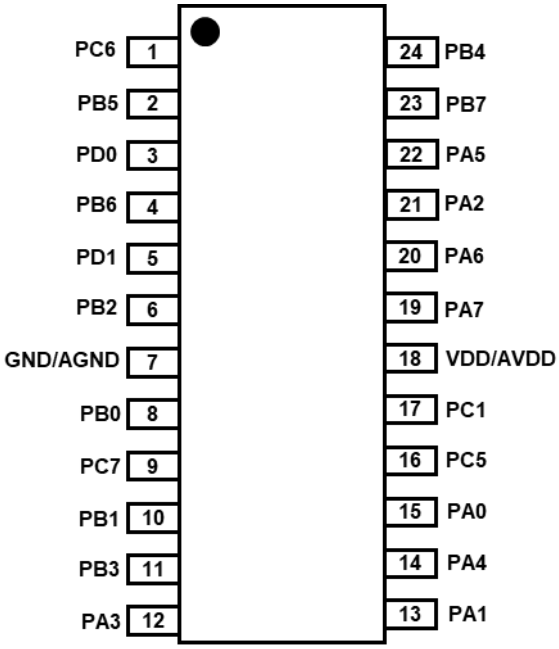
PFC460-S20: SOP20 (300mil)
PFC460-H20: HSOP20 (300 mil)
PFC460-T20: TSSOP20



PFC460-S24: SOP24 (300mil)



PFC460-S28: SOP28 (300mil)
PFC460-T28: TSSOP28



PFC460-Y24: SSOP24 (150mil)

Note: PFC460-Y24 pinout and S24 pinout are incompatible with each other.

Pin Name	I/O	Pull High/Low	Crystal	VDD/2	PWM	PFG	Touch	CS	ADC	Comp.	OPA	Ex. Int	Ex. Reset	Program
PA0	√	H/L		COM2	PWMG0/ LPWMG0		TK7		AD10	CO	OPO	INT0		
PA1	√	H/L					TK13		AD16					
PA2	√	H/L					TK14		AD17			INT0		
PA3	√	H/L		COM2	TM2PWM/ PWMG2/ LPWMG2		TK5		AD8	CIN-	OPIN-	INT1		√
PA4	√	H/L		COM2	PWMG1/ LPWMG1		TK6		AD9	CIN+/ CIN-	OPIN+/ OPIN-	INT1		
PA5	√	H/L			LPWMG2		TK17	CS1	AD18				√	√
PA6	√	H/L	Xout			√	TK8		AD19					√
PA7	√	H/L	Xin					CS0	AD20			INT0		
PB0	√	H/L		COM1/ COM2	TM2PWM	√	TK11		AD0			INT1		
PB1	√	H/L		COM1/ COM2			TK12		AD1/ Vref					
PB2	√	H/L		COM1	TM2PWM/ PWMG2/ LPWMG2		TK9		AD2					
PB3	√	H/L			PWMG2/ LPWMG2		TK10		AD3					
PB4	√	H/L			TM2PWM/ PWMG0/ LPWMG0	√	TK1		AD4					
PB5	√	H/L		COM1	TM3PWM/ PWMG0/ LPWMG0/ LPWMG2		TK2		AD5			INT0		

Pin Name	I/O	Pull High/Low	Cryst-al	VDD/2	PWM	PFG	Touch	CS	ADC	Comp.	OPA	Ex. Int	Ex. Reset	Pro-gram
PB6	√	H/L		COM1	TM3PWM/ PWMG1/ LPWMG1/ LPWMG0		TK3		AD6	CIN-	OPIN-	INT1		
PB7	√	H/L			TM3PWM/ PWMG1/ LPWMG1	√	TK4		AD7	CIN-	OPIN-			
PC0	√	H/L			PWMG2/ LPWMG2		TK15		AD11					
PC1	√	H/L				√	TK18		AD12					
PC2	√	H/L			PWMG0/ LPWMG0		TK16		AD21					
PC3	√	H/L			PWMG1/ LPWMG1		TK19		AD22					
PC4	√	H/L					TK20		AD23					
PC5	√	H/L					TK21		AD24					
PC6	√	H/L					TK22		AD25					
PC7	√	H/L					TK23		AD26					
PD0	√	H/L					TK0		AD27					
PD1	√	H/L							AD28					
VDD/ AVDD														√
GND/ AGND														√

Brief Description of Pin: (For other undescribed things, please refer to the relevant chapters of I/O)

1. All the I/O pins have: Schmitt Trigger input and CMOS voltage level.
2. PA5 is NOT open-drain output. Please put 33Ω resistor in series to have high noise immunity when PA5 is in input mode.
3. 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.
4. 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.
5. IO function is automatically deactivated when a pin is used as PWM output port.

4. Central Processing Unit (CPU)

4.1. Functional Description

There are four processing units (FPPA0 ~ FPPA3) inside the chip of PFC460. In each processing unit, it includes:

a. its own Program Counter to control the program execution sequence
b. its own Stack Pointer to store or restore the program counter for program execution
c. its own accumulator
d. 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, each FPPA unit can execute its own program independently, thus parallel processing can be expected.

4.1.1. Processing Units

These FPPA0 ~ FPPA3 units share the same 4KWx16bits MTP program memory, 512 bytes data SRAM and all the IO ports, these four 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 of FPPA units are illustrated in Fig. 1.

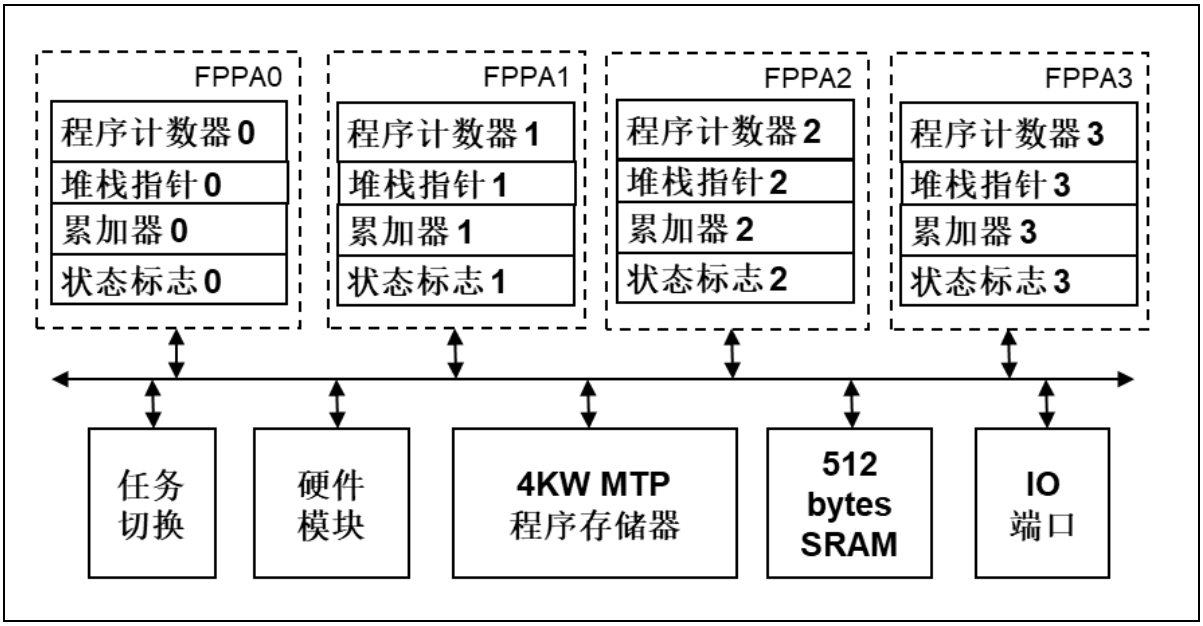


Fig. 1: Hardware Diagram of FPPA units

These four FPPA units are operated at mutual exclusive clock cycles and can be enabled independently.

The system performance is shared to the assigned FPPA units via *pmode* command; please refer to the description of *pomde* instruction. The bandwidth assignment is nothing to do with FPPA enable, which means that the bandwidth is also allocated to the assigned FPPA unit even though it is disabled.

Each FPPA units 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 ~FPPA3 units can be enabled by user's program if necessary. FPPA0 ~ FPPA3 can be enabled or disabled by using any one FPPA unit, including disabled the FPPA unit itself.

Four FPPA units mode

Fig.2 shows the timing sequence of FPPA units for *pmode=6* which will assign the bandwidth to four FPPA units (FPPA0, FPPA1, FPPA2, FPPA3). For *pmode*=6, FPPA0, FPPA1, FPPA2 and FPPA3 will be operated at 2MHz if system clock is 8MHz, which means that each FPPA unit has quarter computing power of whole system.

For FPPA0 unit, its program will be executed once in sequence every four system clock, shown as (M-1)th, Mth, ... (M+4)th instruction. For FPPA1 unit, its program will be also executed once in sequence every four system clock, shown as (N-1)th, Nth, ... (N+3)th instructions. For FPPA2 unit, its program will be also executed once in sequence every four system clock, shown as (O-1)th, Oth, ... (O+3)th instructions. For FPPA3 unit, its program will be also executed once in sequence every four system clock, shown as (P-1)th, Pth, ... (P+3)th instructions.

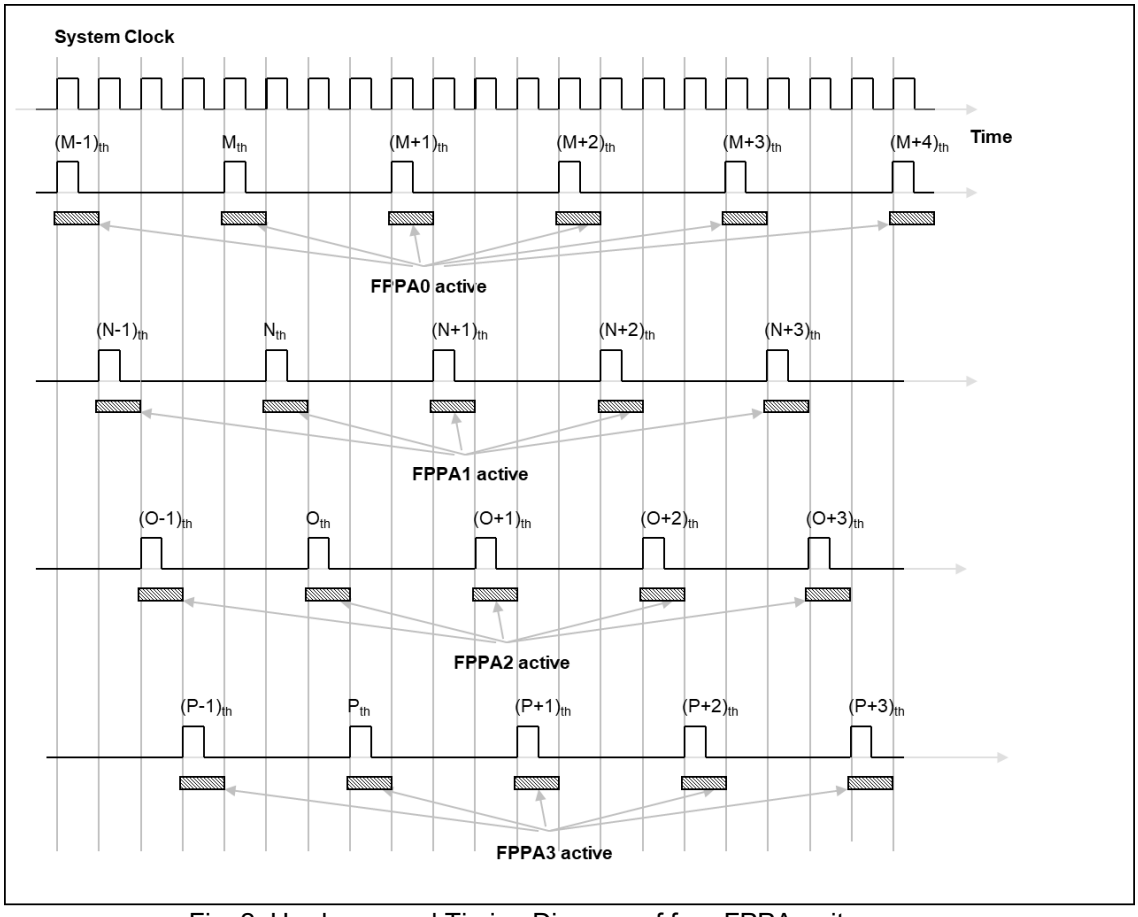


Fig. 2: Hardware and Timing Diagram of four FPPA units

Two FPPA units mode

Fig.3 shows the timing sequence of FPPA units for *pmode*=0 which will assign the bandwidth to two FPPA units only. FPPA0 and FPPA1 each have half computing power of whole system; for *pmode*=0, FPPA0 and FPPA1 will be operated at 4MHz if system clock is 8MHz.

For FPPA0 unit, its program will be executed in sequence every other system clock, shown as (M-1)th, Mth, ... (M+4)th instruction. For FPPA1 unit, its program will be also executed in sequence every other system clock, shown as (N-1)th, Nth, ... (N+3)th instructions.

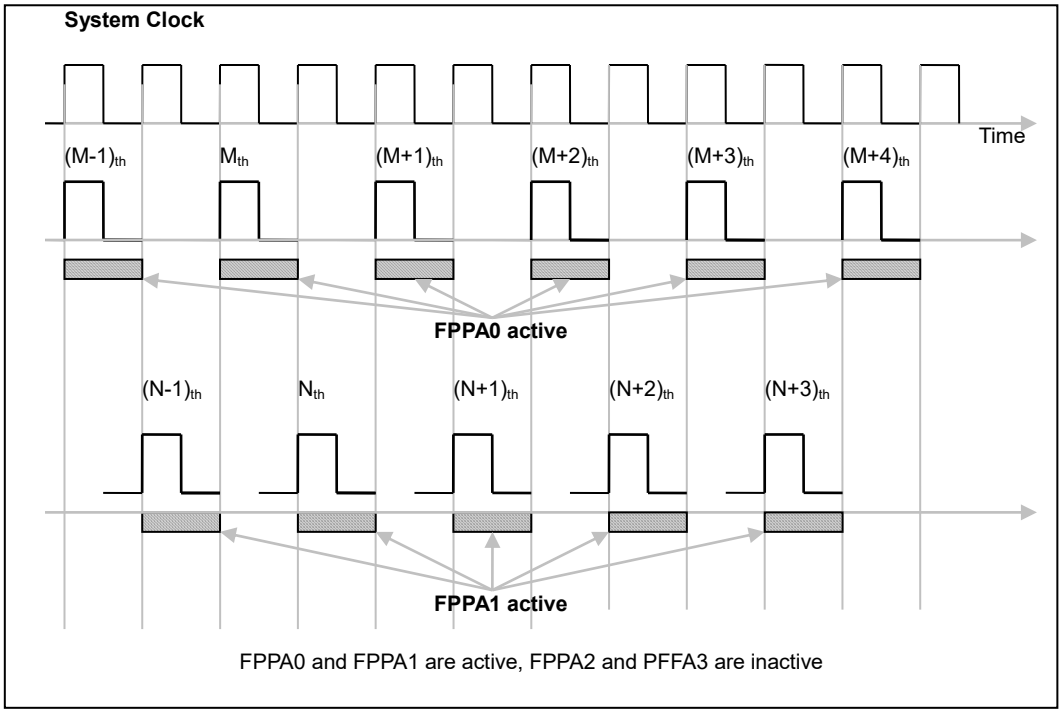


Fig. 3: Hardware and Timing Diagram of two FPPA units

Single FPPA unit mode

For traditional MCU user who does not need parallel processing capability, PFC460 provides single FPPA mode optional to behave as traditional MCU. After single FPPA mode is selected, FPPA1 ~ FPPA3 are always disabled and only FPPA0 is active. Fig.4 shows the timing diagram for each FPPA units, FPPA1/2/3 are always disabled and only FPPA0 active.

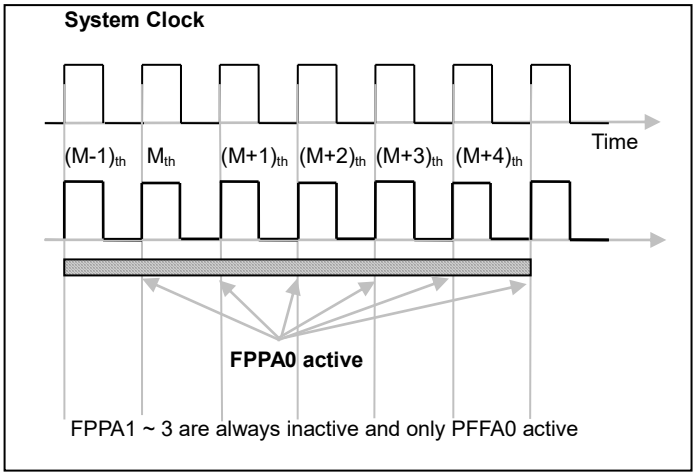


Fig. 4: Hardware and Timing Diagram of single FPPA unit

4.1.2. 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 width of the program counter is 16 for PFC460. After hardware reset, the program counters of FPPA0 ~ FPPA3 are 0, 1, 2, 3. 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.

4.1.3. Program Structure

Program structure of four FPPA units mode

After power-up, the program starting addresses of FPPA0 ~ FPPA3 are 0x000, 0x001, 0x002, 0x003. The 0x010 is the entry address of interrupt service routine, which belongs to FPPA0 only. The basic firmware structure for PFC460 is shown as Fig. 5, the program codes of four 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.

```
// Page 1
.romadr    0x00
// Program Begin
goto      FPPA0Boot;
goto      FPPA1Boot;
goto      FPPA2Boot;
goto      FPPA3Boot;
//-----Interrupt Service Routine--------
.romadr     0x010
pushaf ;
t0sn        intrq.0;       //PA.0 ISR
goto        ISR_PA0;
t0sn        intrq.1;       //PB.0 ISR
//------End of ISR ---------
//------Begin of FPPA0 ----------
FPPA0Boot :
  //--- Initialize FPPA0 SP and so on...
…
FPPA0Loop:
…
goto FPPA0Loop:
//------ End of FPPA0 --------
```

```
// Page 2
//------ Begin of FPPA1 ----------
FPPA1Boot :
//--- Initialize FPPA1 SP and so on…
FPPA1Loop:
…
goto FPPA1Loop:
//--------- End of FPPA1 --------
//------ Begin of FPPA2----------
FPPA2Boot :
//--- Initialize FPPA2 SP and so on…
FPPA2Loop:
…
goto FPPA2Loop:
//--------- End of FPPA2 --------
//------ Begin of FPPA3----------
FPPA3Boot :
//--- Initialize FPPA3 SP and so on…
FPPA3Loop:
…
goto FPPA3Loop:
//--------- End of FPPA3 --------
```

Fig. 5: Program Structure

Program structure of single FPPA unit 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.

4.1.4. 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 ~ FPPA1 will share ALU for its corresponding operation.

4.2. Storage Memory

4.2.1. Program Memory – ROM

The PFC460 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 all FPPA units are stored in this MTP memory. The MTP program memory for PFC460 is 4K words that is partitioned as Table 1.

After reset, the initial address for FPPA0 is 0x000, 0x001 for FPPA1, 0x002 for FPPA2, 0x003 for FPPA3. The interrupt entry is 0x010 if used and interrupt function is for FPPA0 only.

The MTP memory from address 0xFE0 to 0xFFF are for system using, address space from 0x004 to 0x00F and from 0x011 to 0xFDF are user program spaces.

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

Address	Function
0x000	FPPA0 reset – *goto* instruction
0x001	FPPA1reset – *goto* instruction
0x002	FPPA2 reset – *goto* instruction
0x003	FPPA3 reset – *goto* instruction
0x004	User program
•	•
0x00F	User program
0x010	Interrupt entry address
0x011	User program
•	•
0xFDF	User program
0xFE0	System Using
•	•
0xFFF	System Using

Table 1: Program Memory Organization

Example of Using Program Memory for Four FPPA mode

In order to have maximum flexibility for user program using, the user program memory is shared for all FPPA units, and the program space allocation is done by program compiler automatically, user does not need to specify the address if not necessary.

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

Address	Function
0x000	FPPA0 reset – *goto* instruction (*goto* 0x020)
0x001	Begin of FPPA1 program – *goto* instruction (*goto* 0x400)
0x002	Begin of FPPA2 program – *goto* instruction (*goto* 0x5A8)
0x003	Begin of FPPA3 program
•	•
0x00F	*goto* 0xC00 to continue FPPA3 program
0x010	Interrupt entry address (FPPA0 only)
•	•
0x01F	End of ISR (depend on user code size)
0x020	Begin of FPPA0 user program
•	•
0x3FF	End of FPPA0 user program
0x400	Begin of FPPA1 user program
•	•
0x5A7	End of FPPA1 user program
0x5A8	Begin of FPPA2 user program
•	•
0xBFF	End of FPPA2 user program
0xC00	Continuing FPPA3 program
•	•
0xFDF	End of user program
0xFE0	System Using
•	•
0xFFF	System Using

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

Example of Using Program Memory for Two FPPA mode

Address	Function
0x000	FPPA0 reset – goto instruction (*goto* 0x020)
0x001	FPPA1reset – goto instruction (*goto* 0x7A1)
0x002	Reserved
0x003	Reserved
0x004	Not used
•	•
0x00F	Not used
0x010	Interrupt entry address(FPPA0 only)
•	•
0x01F	End of ISR (depend on user code size)
0x020	Begin of FPPA0 user program
•	•
0x7A0	End of FPPA0 user program
0x7A1	Begin of FPPA1 user program
•	•
0xF37	End of FPPA1 user program
0xF38	Not used
•	•
0xFDF	Not used
0xFE0	System Using
•	•
0xFFF	System Using

Table 3: 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 4 shows the example of program memory using for single FPPA unit mode.

Address	Function
0x000	FPPA0 reset
0x001	Begin of FPPA0 user program
•	•
0x00F	*goto* instruction (*goto* 0x020)
0x010	Interrupt entry address
•	• ISR
0x01F	End of ISR (depend on user code size)
0x020	user program
•	•
0xFDF	user program
0xFE0	System Using
•	•
0xFFF	System Using

Table 4: Example of using Program Memory for single FPPA mode

4.2.2. Data Memory – SRAM

Fig. 6 shows the SRAM data memory organization of PFC460, all the SRAM data memory could be accessed by FPPA0 ~ FPPA3 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 512 bytes data memory of PFC460 can be accessed by indirect access mechanism.

Bit defined in RAM: Addressed at 0x00 ~ 0x1FF. However, bit defined in register space: Only addressed at 0x00 ~ 0x3F.

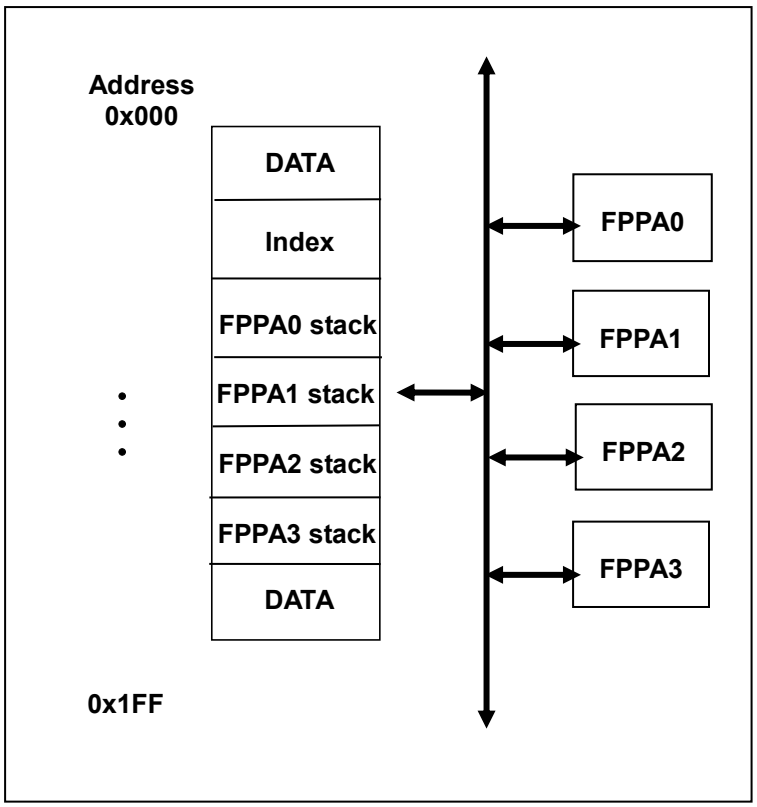


Fig. 6: Data Memory Organization

4.2.3. System Register

The register space of PFC460 is independent of SRAM space and MTP space.

The following is the PFC460 register address. For usage and description, please refer to their relevant chapters.

	+0	+1	+2	+3	+4	+5	+6	+7
0x00	*FLAG*	*FPPEN*	*SP*	*CLKMD*	*INTEN*	*INTRQ*	*INTEN2*	*INTRQ2*
0x08	*T16M*	-	-	-	*LPWMG0C*	*LPWMG1C*	*LPWMG2C*	*EOSCR*
0x10	*PA*	*PAC*	*PAPH*	*PAPL*	*PB*	*PBC*	*PBPH*	*PBPL*
0x18	*PC*	*PCC*	*PCPH*	*PCPL*	*PD*	*PDC*	*PDPH*	*PDPL*
0x20	*ADCC*	*ADCM*	*PWMG0C*	*PWMG0S*	*PWMG1C*	*PWMG1S*	*PWMG2C*	*PWMG2S*
0x28	*TM2C*	*TM2CT*	*TM3C*	*TM3CT*	*MULOP*	*MULRH*	-	-
0x30	*PFGRH*	*PFGRL*	*PFGC*	*OPAC*	*OPAOFS*	*GPCC*	*GPCS*	*TS*
0x38	*TCC*	*TKE3*	*TKE2*	*TKE1*	*TPS*	*TPS2*	-	-
0x40	-	-	*ADCRGC*	*GPC2PWMG*	-	-	-	-
0x48	-	*MISC*	*ADCRH*	*ADCRL*	*PADIER*	*PBDIER*	*PCDIER*	*PDDIER*
0x50	*PWMG0DTH*	*PWMG0DTL*	*PWMG0CUBH*	*PWMG0CUBL*	*PWMG1DTH*	*PWMG1DTL*	*PWMG1CUBH*	*PWMG1CUBL*
0x58	*PWMG2DTH*	*PWMG2DTL*	*PWMG2CUBH*	*PWMG2CUBL*	-	*INTEGS*	-	*IOHD*
0x60	*TM2S*	*TM2B*	*TM3S*	*TM3B*	-	-	-	*LPWMGCLK*
0x68	*LPWMCUBH*	*LPWMCUBL*	*LPWMG0DTH*	*LPWMG0DTL*	*LPWMG1DTH*	*LPWMG1DTL*	*LPWMG2DTH*	*LPWMG2DTL*
0x70	-	-	-	-	-	-	-	-
0x78	-	-	*TKCH*	*TKCL*	-	-	-	-

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 – 4	-	-	Reserved.
3	0	R/W	FPPA3 enable. This bit is used to enable FPPA3. 0/1: disable/enable
2	0	R/W	FPPA2 enable. This bit is used to enable FPPA2. 0/1: disable/enable
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 = 0x49

Bit	Reset	R/W	Description
7	-	-	Reserved. (keep 0 for future compatibility)
6	-	WO	Reserved. Please set to 1 by manual.
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 2048 ILRC clocks (Not for fast boot-up) 1: Fast wake up. The wake-up time is 32 ILRC clocks.
4	0	WO	Enable VDD/2 bias voltage generator 0 / 1: Disable / Enable
3	0	WO	VDD/2bias voltage output pin selection: 0: LCD_B01256, including PB0/PB1/PB2/PB5/PB6 1: LCD_A034_B01, including PA0/PA3/PA4/PB0/PB1
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 256 bytes and should be defined in even positions within 256 bytes from 0x00 address. The stack memory of PFC460 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. 7 shows that the status of stack before FPPA0 execution, system has calculated the required stack space and has reserved for the program.

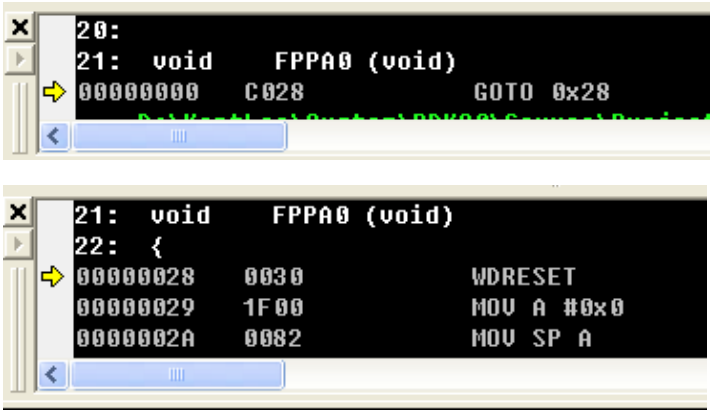


Fig. 7: Stack Assignment in Mini-C project

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	14 levels	4.5V, 4.0V, 3.75V, 3.5V, 3.3V, 3.15V, 3.0V, 2.7V, 2.5V, 2.4V, 2.3V, 2.2V, 2.1V, 2.0V
Interrupt Src0	**PA.0(default)**	*INTEN/ INTRQ*.Bit0 is form PA.0
	PB.5	*INTEN/ INTRQ*.Bit0 is form PB.5
	PA.2	*INTEN/ INTRQ*.Bit0 is form PA.2
	PA.7	*INTEN/ INTRQ*.Bit0 is form PA.7
Interrupt Src1	**PA.4(default)**	*INTEN/ INTRQ*.Bit1 is form PA.4
	PA.3	*INTEN/ INTRQ*.Bit1 is form PA.3
	PB.0	*INTEN/ INTRQ*.Bit1 is form PB.0
	PB.6	*INTEN/ INTRQ*.Bit1 is form PB.6
TMx_Source	**16MHz(default)**	When *TMxC*[7:4]=0010, TMx clock source= 16 MHz (x=2/3)
	32MHz	When *TMxC*[7:4]=0010, TMx clock source = 32 MHz (x=2/3)
TMx_Bit	**6 Bit(default)**	When *TMxS*.7 = 1, TMx PWM resolution is 6 Bit (x=2/3)
	7 Bit	When *TMxS*.7 = 1, TMx PWM resolution is 7 Bit (x=2/3)
TM2C[3:2]=1	**PB2(default)**	When *TM2C[3:2]*=01, output pin is PB2
	PB0	When *TM2C[3:2]*=01, output pin is PB0
OPA_PWM	**Disable(default)**	OPA / PWM are independent

Option	Selection	Description
	Enable	OPA output control PWM output
PWM_Source	**16MHz(default)**	When *PWMGxC*.0 = 1, PWMGx clock source = 16 MHz (x=0/1/2)
	32MHz	When *PWMGxC*.0 = 1, PWMGx clock source = 32 MHz (x=0/1/2)
LPWM_Source	**16MHz(default)**	When *LPWMGCLK*.0 = 1, LPWMG clock source = 16 MHz
	32MHz	When *LPWMGCLK*.0 = 1, LPWMG clock source = 32 MHz
GPC_P_In	**PA.4(default)**	Comparator plus input is from PA.4
	PA.0	Comparator plus input is from PA.0
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
CS_Sel	**PA7(default)**	Set PA7 as touch CS pin
	PA5	Set PA5 as touch CS pin
	Disable	Disable touch CS pin
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
	4-FPPA	Four FPPA units mode
Burning_Enhancement	**Newer（default）**	Select the method of burning enhancement (recommended to use)
	Older	Compatible with older versions (before 0.97C5)

5. Oscillator and System Clock

There are three oscillator circuits provided by PFC460: 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	*EOSCR*.7
IHRC	*CLKMD*.4
ILRC	*CLKMD*.2

Table 5: Three Oscillator Circuits provided by PFC460

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 PFC460 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 MHz to 4MHz, depending on the crystal placed on; higher frequency oscillator than 4MHz is NOT supported. Fig.8 shows the hardware connection under this application.

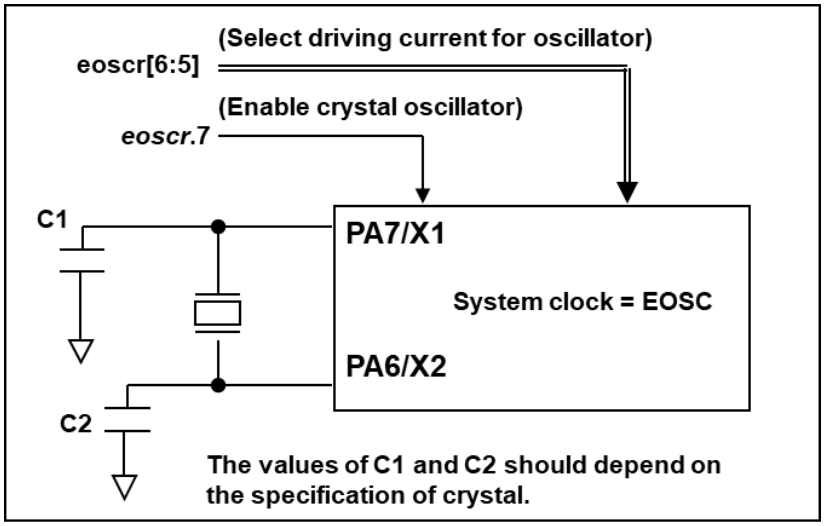


Fig. 8: Connection of crystal oscillator

5.2.1. External Oscillator Setting Register (*EOSCR*), address = 0x0F

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	-	-	Reserved.
3	-	WO	LDO output voltage option。0/1：2.4V/2V
2	-	WO	Touch module power option。0/1：VDD/LDO
1 – 0	-	-	Reserved.

5.2.2. Usages and Precautions of External Oscillator

Besides crystal, external capacitor and options of PFC460 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 6 shows the recommended values of C1 and C2 for different crystal oscillator; the measured start-up time under its corresponding conditions is also shown. Since the crystal or resonator had its own characteristic, the capacitors and start-up time may be slightly different for different type of crystal or resonator, please refer to its specification for proper values of C1 and C2.

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

Table 6: 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 PFC460 is shown as Fig. 9.

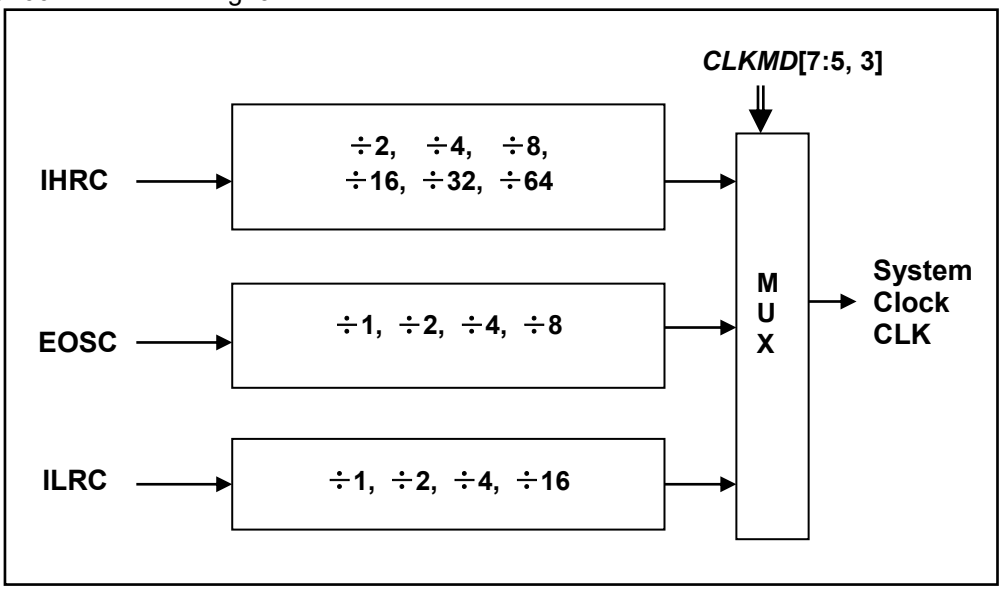


Fig. 9: Options of System Clock

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/16 011: IHRC/32 100: IHRC/64 101: EOSC/8 110: ILRC/2 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, PFC460 provide the IHRC frequency calibration to eliminate this variation, and this function can be selected when compiling user's program and the command will be inserted into user's program automatically. The calibration command is shown as below:

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

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

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

p3=3.9 ~ 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 stating 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 7:

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 change, bandgap OFF

Table 7: Options for IHRC Frequency Calibration

The following shows the different states of PFC460 under different options:

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

After boot up, *CLKMD* = 0x14:

a. IHRC frequency is calibrated to 16MHz@V_{DD}=5V and IHRC module is enabled
b. System CLK = IHRC/4 = 4MHz
c. 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 up, *CLKMD* = 0x3C：

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

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

After boot up, *CLKMD* = 0xE4：

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

(4) .ADJUST_IC DISABLE

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

a. IHRC is not calibrated.
b. System CLK = ILRC or IHRC/64
c. 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 PFC460 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

```
...
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

```
...
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

```
...
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

There are many causes to reset the PFC460, once reset is asserted, most of all the registers in PFC460 will be set to default values, system should be restarted once abnormal cases happen, or by jumping program counter to address 0x00.

After a power-on reset or LVR reset occurs, if VDD is greater than VDR (data storage voltage), the value of the data memory will be retained, but if the SRAM is cleared after re-power, the data cannot be retained; if VDD is less than VDR, the data The value of the memory will be turned into an unknown state that is in an indeterminate state.

If a reset occurs, and there is an instruction or syntax to clear SRAM in the program, the previous data will be cleared during program initialization and cannot be retained.

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

6.1. Power On Reset - POR

POR (Power-On-Reset) is used to reset PFC460 when power up. Time for boot-up is about 2048 ILRC clock cycles. The power up sequence is shown in the Fig. 10. Customer must ensure the stability of supply voltage after power up no matter which option is chosen.

After power-up, 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}.

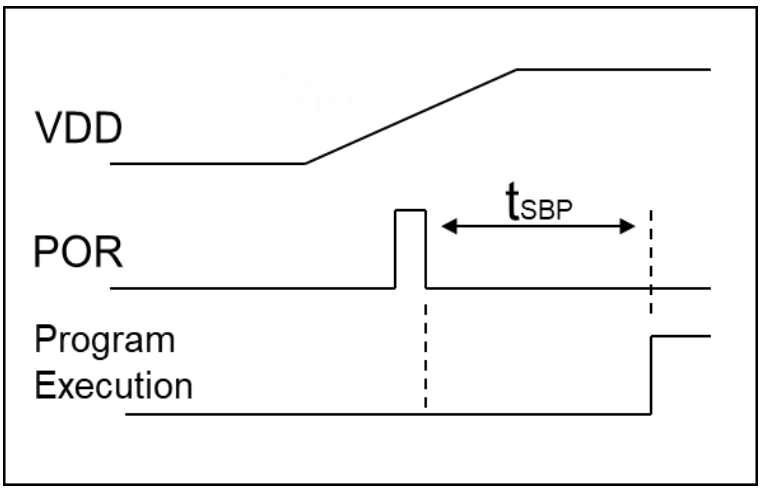


Fig. 10: Power Up Sequence

Fig. 11 shows the typical program flow after boot up. Please notice that the FPPA1 - FPPA3 are disabled after reset and recommend NOT enabling other FPPA units before system and FPPA0 initialization.

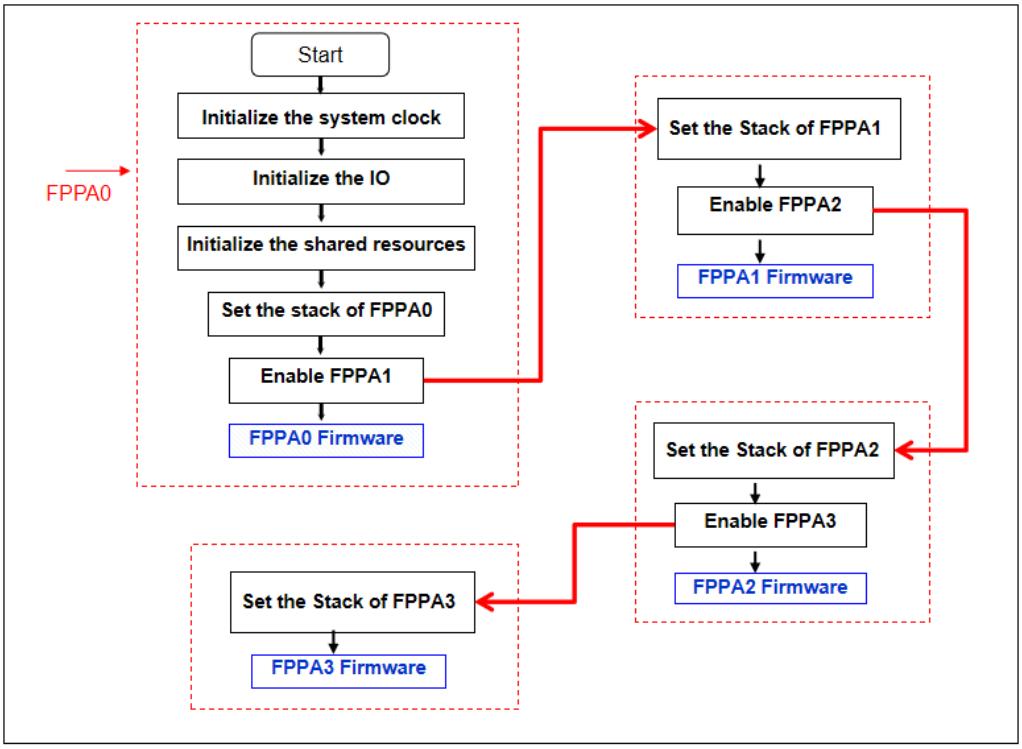


Fig. 11: Boot Procedure

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 12.

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<VDR.

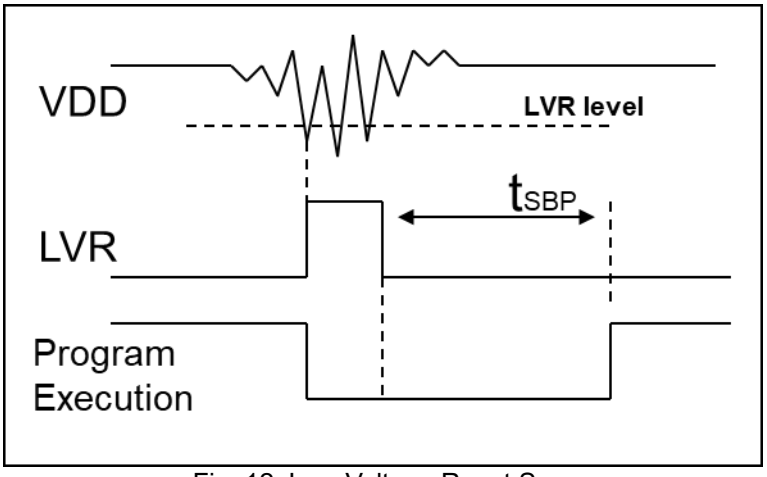


Fig. 12: Low Voltage Reset Sequence

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	≧ 4.5V	4.5V
4MHz	≧ 2.7V	2.7V
2MHz	≧ 2.2V	2.2V

Table 8: LVR setting for reference

(1) The setting of LVR (2.0V ~ 4.5V) will be valid just after successful power-on process.

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

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

MISC Register (*MISC*), address = 0x49			
Bit	**Reset**	**R/W**	**Description**
7	-	-	Reserved. (keep 0 for future compatibility)
6	0	WO	The crystal oscillator driving current selection.
5	0	WO	Enable fast Wake-up.
4	0	WO	Enable VDD/2 bias voltage generator
3	0	WO	VDD/2 bias voltage output pin selection.
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, watchdog is open by default, but when the program executes ADJUST_IC, the watchdog will be closed. To use the watchdog, you need to reconfigure the open. Watchdog will be inactive once ILRC is disabled. 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, PFC460 will be reset to restart the program execution. The relative timing diagram of watchdog timer is shown as Fig. 13.

The PFC460 data memory will be reserved when the WDT reset occurs.

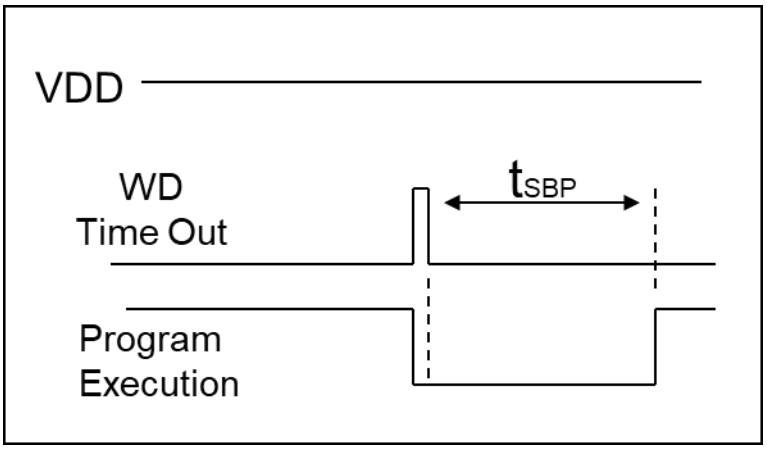


Fig. 13: Sequence of Watch Dog Timeout reset

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(*CLKMD*), 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 PFC460 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 14.

The PFC460 data memory will be reserved when the PRSTB reset occurs.

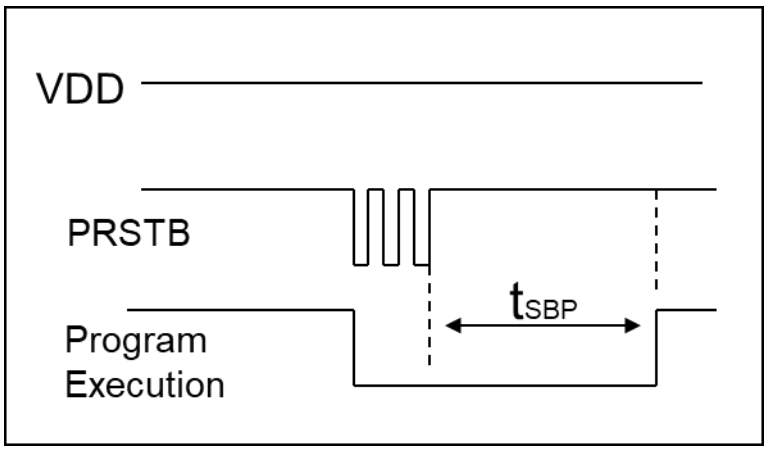


Fig. 14: Sequence of PRSTB reset

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, TM2/3, PWMG0/1/2, LPWMG.)
(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 PFC460 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:
    a. IO toggle in digital mode (*PxDIER* bit is 1)
    b. *TM3C*.NILRC wake-up: The clock source of Timer3 selects NILRC.

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

```
CLKMD   =   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 PFC460 can be resumed to normal operation by toggling IO pins or *TM3C*.NILRC wake-up. Other Timer wake-up are available for Power-Save mode ONLY. Table 9 shows the differences in wake-up sources between STOPSYS and STOPEXE.

Differences in wake-up sources between STOPSYS and STOPEXE			
	IO Toggle	TM3C.NILRC wake-up	Others Timer wake-up
STOPSYS	Yes	Yes	No
STOPEXE	Yes	Yes	Yes

Table 9: Differences in wake-up sources between Power-Save mode and Power-Down mode

When using the IO pins to wake-up the PFC460, registers *PxDIER* should be properly set to enable the wake-up function for every corresponding pin. The time for normal wake-up is about 2048 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 32 ILRC clocks from IO toggling.

Suspend mode	Wake-up mode	Wake-up time (t_{WUP}) from IO toggle
stopexe suspend or *stopsys* suspend	Fast wake-up	32 * T_{ILRC}, Where T_{ILRC} is the time period of ILRC
stopexe suspend or *stopsys* suspend	Normal wake-up	2048* T_{ILRC}, Where T_{ILRC} is the clock period of ILRC

Table 10: Differences in wake-up time between fast/normal wake-up

8. Interrupt

There are eleven interrupt sources for PFC460, including two external IO interrupts. The two external interrupt sources have a total of eight IO for users to choose. The details are as follows:

- External interrupt PA0/PB5/PA2/PA7 (by Code Option Interrupt Src0)
- External interrupt PB0/PA4/PA3/PB6 (by Code Option Interrupt Src1)
- ADC interrupt
- GPC interrupt
- Timer16 interrupt
- Timer2 interrupt
- Timer3 interrupt
- LPWM (SuLED PWM) interrupt
- PWMG0 interrupt
- TK_OV interrupt
- TK_END 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. 15. All the interrupt request flags are set by hardware and cleared by writing *INTRQ/INTRQ2* 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/INTRQ2* register.

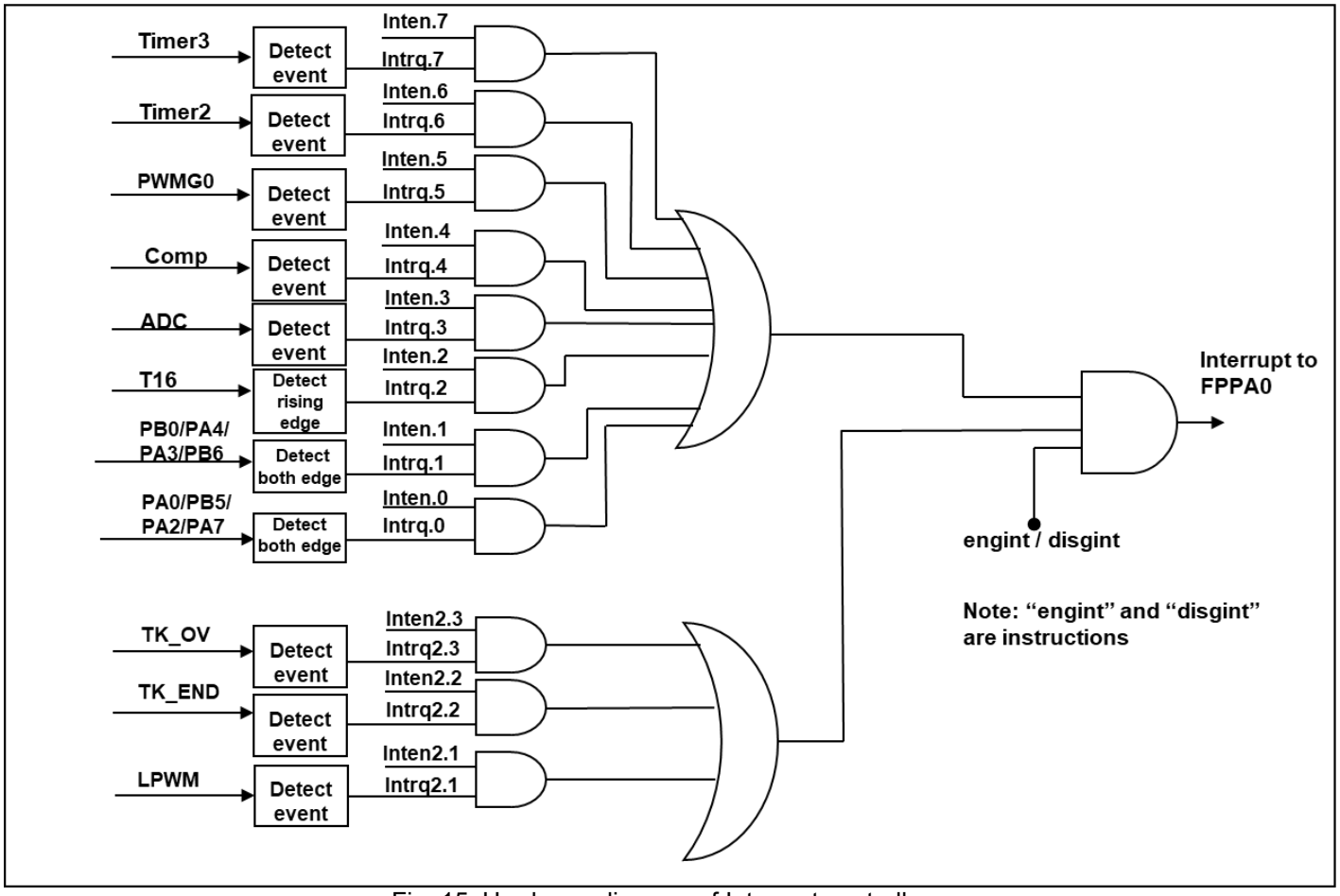


Fig. 15: Hardware diagram of Interrupt controller

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

Bit	Reset	R/W	Description
7	-	R/W	Enable interrupt from Timer3. 0 / 1: disable / enable.
6	-	R/W	Enable interrupt from Timer2. 0 / 1: disable / enable.
5	-	R/W	Enable interrupt from PWMG0. 0 / 1: disable / enable.
4	-	R/W	Enable interrupt from comparator. 0 / 1: disable / enable.
3	-	R/W	Enable interrupt from ADC. 0 / 1: disable / enable.
2	-	R/W	Enable interrupt from Timer16 overflow. 0 / 1: disable / enable.
1	-	R/W	Enable interrupt from PB0/PA4/PA3/PB6. 0 / 1: disable / enable. (by code option Interrupt Src1)
0	-	R/W	Enable interrupt from PA0/PB5/PA2/PA7. 0 / 1: disable / enable. (by code option Interrupt Src0)

8.2. Interrupt Enable Register2 (*INTEN2*), address = 0x06

Bit	Reset	R/W	Description
7 – 4	-	-	Reserved
3	-	R/W	Enable interrupt from TK_OV. 0/1: disable / enable.
2	-	R/W	Enable interrupt from TK_END. 0/1: disable / enable.
1	-	R/W	Enable interrupt from LPWM. 0/1: disable / enable.
0	-	-	Reserved

8.3. 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/PA3/PB6, this bit is set by hardware and cleared by software. 0 / 1: No request / Request
0	-	R/W	Interrupt Request from pin PA0/PB5/PA2/PA7, this bit is set by hardware and cleared by software. 0 / 1: No Request / request

8.4. Interrupt Request Register2 (*INTRQ2*), address = 0x07

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

8.5. Interrupt Edge Select Register (*INTEGS*), address = 0x5D

Bit	Reset	R/W	Description
7 – 5	-	WO	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/PA3/PB6 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/PA2/PA7 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:

INTEN/INTEN2 and *INTRQ/INTRQ2* have no initial values. Please set required value before enabling interrupt function.

Even if *INTEN/INTEN2*=0, *INTRQ/INTRQ2* will be still triggered by the interrupt source.

8.6. 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.

8.7. General Steps to Interrupt

When using the interrupt function, the procedure should be:

Step1: Set *INTEN/INTEN2* register, enable the interrupt control bit.
Step2: Clear *INTRQ/INTRQ2* register.
Step3: In the main program, using *engint* to enable CPU interrupt function.
Step4: Wait for interrupt. When interrupt occurs, enter to Interrupt Service Routine.
Step5: After the Interrupt Service Routine being executed, return to the main program.

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

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

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

8.8. 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 = 0x4C

Bit	Reset	R/W	Description
7	1	WO	Enable PA7 digital input and wake-up event and interrupt request. 1 /0: enable / disable. This bit should be set to low to prevent leakage current when external crystal oscillator is used.
6	1	WO	Enable PA6 digital input and wake-up event. 1 /0: enable / disable. This bit 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 – 2	111	WO	Enable PA4/PA3/PA2 digital input and wake-up event and interrupt request. 1 /0: enable / disable.
1	1	WO	Enable PA1 digital input and wake-up event. 1 /0: enable / disable.
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 = 0x4D

Bit	Reset	R/W	Description
7	1	WO	Enable PB7 digital input and wake-up event. 1 /0: enable / disable.
6 – 5	11	WO	Enable PB6~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 C Digital Input Enable Register (*PCDIER*), address = 0x4E

Bit	Reset	R/W	Description
7 – 0	0xFF	WO	Enable PC7~PC0 digital input and wake-up event. 1 /0: enable / disable

9.1.4. Port D Digital Input Enable Register (*PDDIER*), address = 0x4F

Bit	Reset	R/W	Description
7 – 2	-	-	Reserved
1 – 0	11	WO	Enable PD1~PD0 digital input and wake-up event. 1 /0: enable / disable

9.1.5. Port A/B/C Data Registers(*PA/PB/PC*), address = 0x10/0x14/0x18

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

9.1.6. Port A/B/C Control Registers (*PAC/PBC/PCC*), address = 0x11/0x15//0x19

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

9.1.7. Port A/B/C Pull-High Registers (*PAPH/PBPH/PCPH*), address = 0x12/0x16/0x1A

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

9.1.8. Port A/B/C Pull-Low Register (*PAPL/PBPL/PCPL*), address = 0x13/0x17/0x1B

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

9.1.9. Port D Data Registers (*PD*), address = 0x1C

Bit	Reset	R/W	Description
7 – 2	-	-	Reserved.
1 – 0	00	R/W	Data registers bit[1:0] for Port D.

9.1.10. Port D Control Registers (*PDC*), address = 0x1D

Bit	Reset	R/W	Description
7 – 2	-	-	Reserved.
1 – 0	00	R/W	Port D control registers. This register is used to define input mode or output mode for PD0 ~ PD1. 0 / 1: input / output.

9.1.11. Port D Pull-High Registers (*PDPH*), address = 0x1E

Bit	Reset	R/W	Description
7 – 2	-	-	Reserved.
1 – 0	00	R/W	Port D pull-high registers. This register is used to enable the internal pull-high device on PD0 ~ PD1. 0 / 1 : disable / enable

9.1.12. Port D Pull-Low Register (*PDPL*), address = 0x1F

Bit	Reset	R/W	Description
7 – 2	-	-	Reserved.
1 – 0	00	R/W	Port B pull-low register. This register is used to enable the internal pull-low device on PD0 ~ PD1. 0 / 1 : disable / enable

9.2.IO Structure and Functions

9.2.1. IO Pin Structure

Almost all the IO pins of PFC460 have the same structure. The hardware diagram of IO buffer is shown as Fig. 16.

PA5 of PFC460 is similar to others IO and can be used as a general input/output pin, rather than open-drain output.

For the description of the drive capability, please refer to the specification of the *IOHD* register below.

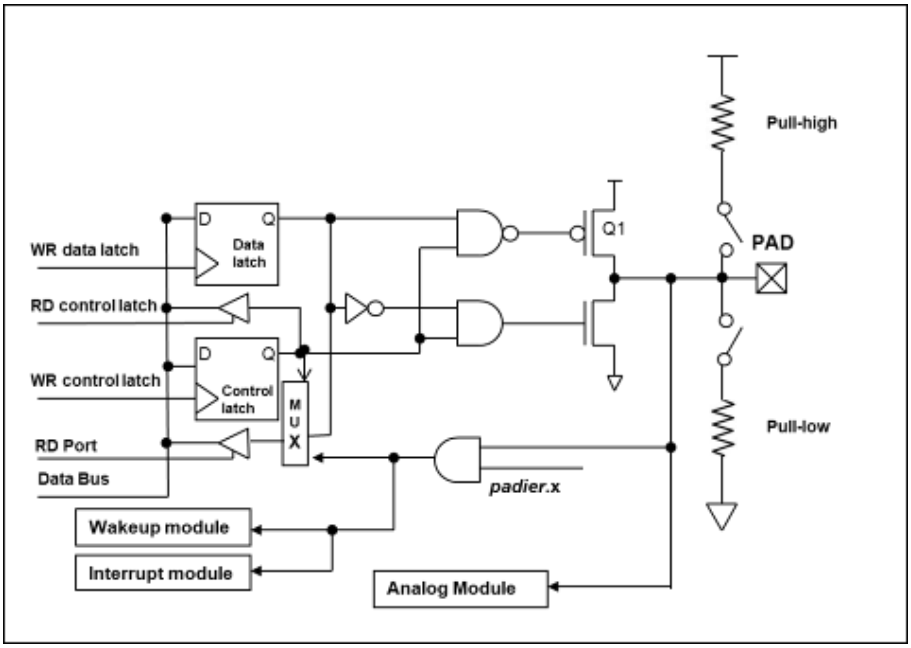


Fig. 16: Hardware diagram of IO buffer

9.2.2. IO Pin Functions

(1) Input / Output function:

PFC460 all the pins can be independently set into digital input, analog input, output low, output high.

Each IO pin can be independently configured for different state by configuring the data registers (*PA/PB/PC/PD*), control registers (*PAC/PBC/PCC/PDC*), pull-high registers (*PAPH/PBPH/PCPH/PDPH)* and pull-low registers (*PAPL/PBPL/PCPL/PDPL).*

The corresponding bits in registers *PxDIER* should be set to low to prevent leakage current for those pins are selected to be analog function. When it is set to output mode, the pull-high / pull-low resistor is turned off automatically.

If user wants to read the pin state, please notice that it should be set to input mode before reading the data port. If user reads the data port when it is set to output mode, the reading data comes from data register, NOT from IO pad.

As an example, Table 11 shows the configuration table of bit 0 of port A.

PA.0	*PAC*.0	*PAPH*.0	*PAPL*.0	Description
X	0	0	0	Input without pull-high / pull-low resistor
X	0	1	0	Input with pull-high resistor
X	0	0	1	Input with pull-low resistor
X	0	1	1	Input with pull-low / pull-high resistor
0	1	X	X	Output low without pull-high / pull-low resistor
1	1	X	X	Output high without pull-high / pull-low resistor

Table 11: PA0 Configuration Table

(2) Wake-up function:

When PFC460 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 digital 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:

PB0 and PB2~PB7 of PFC460 have two kinds of strong/normal drive capability for output, and users can flexibly adjust capability through the *IOHD* register.

9.2.2.1. IO Control Output Drive Capability Register(*IOHD*), address = 0x5F

Bit	Reset	R/W	Description
7	0	WO	PB7 IOH/IOL drive capability selection. 0/1: Normal/Strong
6	0	WO	PB6 IOH/IOL drive capability selection. 0/1: Normal/Strong
5	0	WO	PB5 IOH/IOL drive capability selection. 0/1: Normal/Strong
4	0	WO	PB4 IOH/IOL drive capability selection. 0/1: Normal/Strong
3	0	WO	PB3 IOH/IOL drive capability selection. 0/1: Normal/Strong
2	0	WO	PB2 IOH/IOL drive capability selection. 0/1: Normal/Strong
1	-	-	Reserved
0	0	WO	PB0 IOH/IOL drive capability selection. 0/1: Normal/Strong

9.2.3. IO Pin Usage and Setting

(1) IO pin as digital input

◆ When IO is set as digital input, the level of Vih and Vil would changes with the voltage and temperature. Please follow the minimum value of Vih and the maximum value of Vil.

◆ The value of internal pull high resistor would also change with the voltage, temperature and pin voltage. It is not the fixed value.

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

◆ Configure IO pin as input mode by PxC register.

◆ Set corresponding bit to “1” in PxDIER.

◆ For those IO pins of PD that are not used, PDDIER[7:2] 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 >33Ω resistor in between PA5 and the long wire.

◆ Avoid using PA5 as input in such application.

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

10. Timer / PWM Counter

10.1. 16-bit Timer (Timer16)

10.1.1. Timer16 Introduction

PFC460 provide a 16-bit hardware timer (Timer16/T16) and its block diagram is shown in Fig. 17. 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.

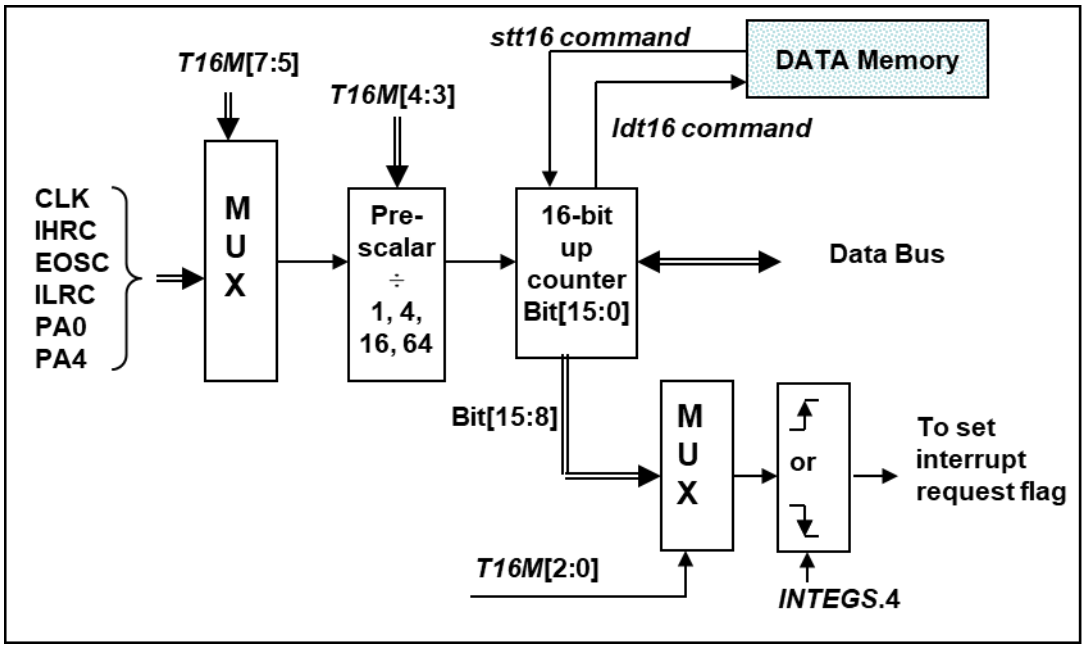


Fig. 17: Hardware diagram of Timer16

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

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

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

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

$    T16M       PA0_F, /1, BIT8;
     // choose PA0 as clock source, every 2^9 to generate INTRQ.2 =1
     // receiving every 512 times PA0 to generate INTRQ.2 =1

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

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

$$F_{INTRQ\_T16M} = F_{clock\ source} \div P \div 2^{n+1}$$

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

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

N is the nth bit selected to request interrupt service, for example: n=10 if bit 10 is selected.

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. Timer16Mode Register (*T16M*), address = 0x08

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 PFC460. Timer2 is used as an example to describe its function due to these principles of two 8-bit timers are similar. Fig. 18 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.

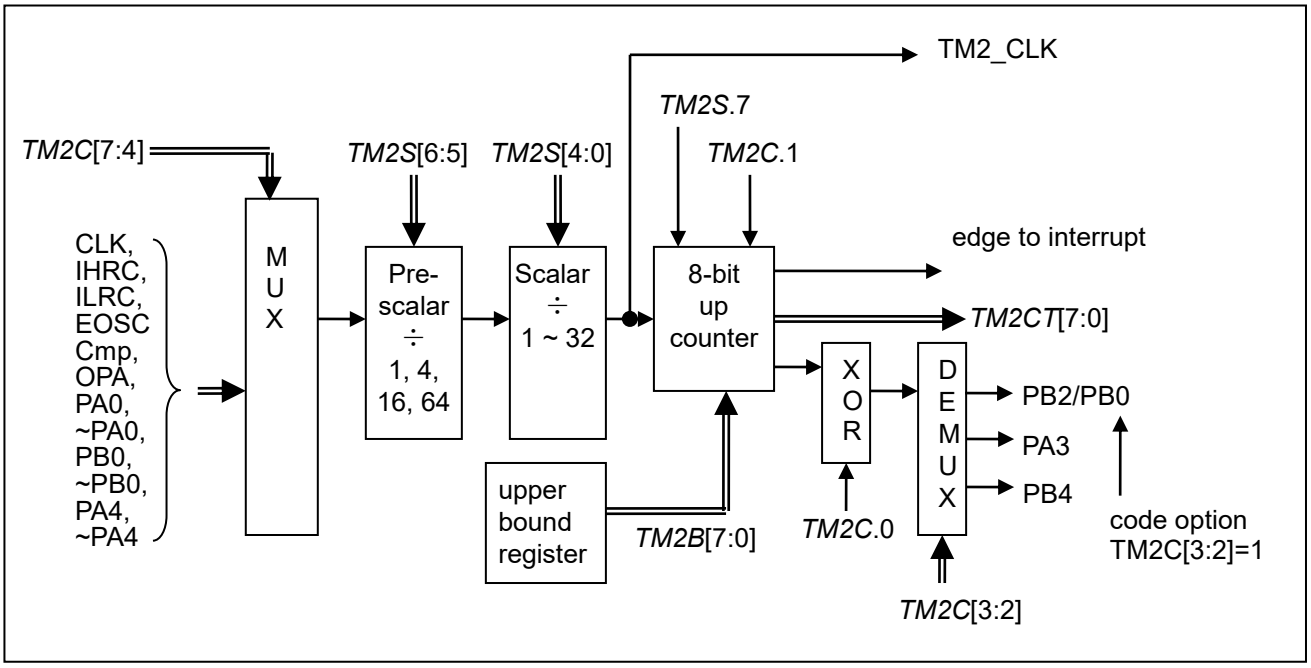


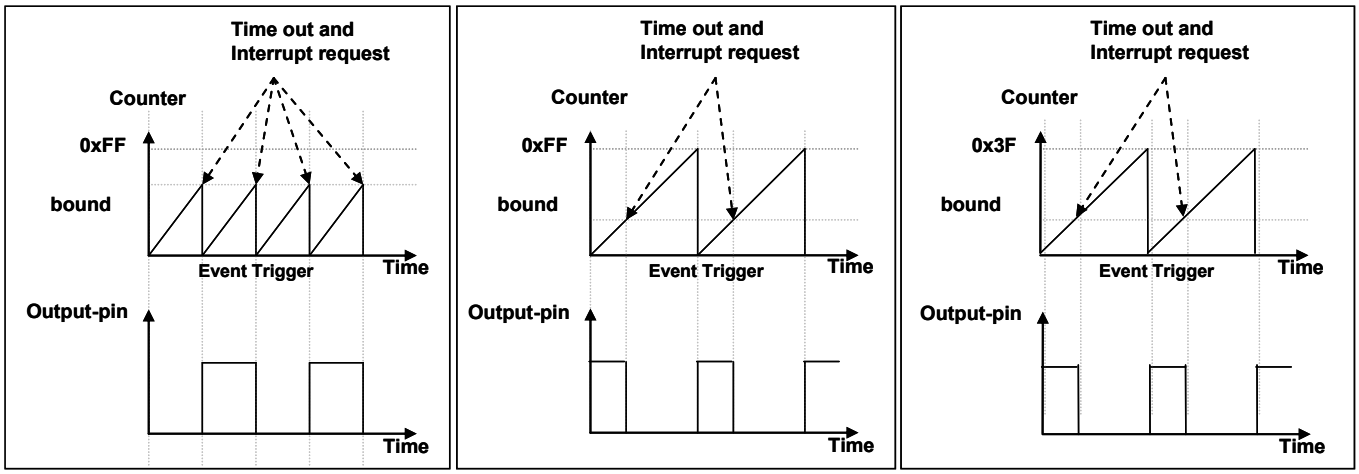
Fig. 18: Timer2 hardware diagram

The hardware diagram of Timer3 is similar to Timer2, but its counter clock source has one more NILRC oscillator than Timer2, and its PWM output pin is PB5, PB6 or PB7.

Bit [7:4] of register *TM3C* selects NILRC as clock source, which can support lower-power wake-up “stopexe” and “stopsys”. NILRC is a slower clock than ILRC, and it is used to make a wake-up clock source with lower power consumption. It can be used by Timer3 ONLY. NILRC and ILRC estimate frequency through IHRC, however NILRC’s frequency drifts a lot. It needs to estimate frequency before it can be used. If users need related demo, please contact FAE.

Selecting code option OPA_PWM can control the generated PWM waveform by the comparator result. Please refer the section of OPA for details.

Fig. 19 shows the timing diagram of Timer2 for both period mode and PWM mode.



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

Fig. 19: Timing diagram of Timer2 in period mode and PWM mode (*TM2C*.1=1)

10.2.1. Timer2, Timer3 Related Registers

10.2.1.1. Timer2/Timer3 Bound Register (*TM2B/TM3B*), address = 0x61/0x63

Bit	Reset	R/W	Description
7 – 0	0x00	WO	Timer2/Timer3 bound register.

10.2.1.2. Timer2/Timer3 Counter Register (*TM2CT/TM3CT*), address = 0x29/0x2B

Bit	Reset	R/W	Description
7 – 0	0x00	R/W	Bit [7:0] of Timer2/Timer3 counter register.

10.2.1.3. Timer2/Timer3 Scalar Register (*TM2S/TM3S*), address = 0x60/0x62

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/Timer3 clock pre-scalar. 00 : ÷ 1 01 : ÷ 4 10 : ÷ 16 11 : ÷ 64
4 – 0	00000	WO	Timer2/Timer3 clock scalar.

10.2.1.4. Timer2/Timer3 Control Register(*TM2C/TM3C*), address = 0x28/0x2A

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	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 0111: NILRC 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
3 – 2	00	R/W	Timer2 output selection. 00: disable 01: PB2 or PB0 (by code option TM2C[3:2]=1) 10: PA3 11: PB4	Timer3 output selection. 00: disable 01: PB5 10: PB6 11: PB7
1	0	R/W	Timer2/Timer3 mode selection. 0 / 1: period mode / PWM mode	
0	0	R/W	Enable to inverse the polarity of Timer2/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:

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

Where,

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

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

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

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

Example 1:

TM2C = 0b0001_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:

```
void    FPPA0 (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:

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

Duty of Output =[(K＋1) ÷ 256] × 100%

Where,

Y = *TM2C*[7:4] : frequency of selected clock source
K = *TM2B*[7:0] : bound register in decimal
S1= *TM2S*[6:5] : pre-scalar (S1 = 1, 4, 16, 64)
S2 = *TM2S*[4:0] : scalar register in decimal (S2 = 0 ~ 31)

Example 1:

TM2C = 0b0001_1010, Y=4MHz
TM2B = 0b0111_1111, K=127
TM2S = 0b0_00_00000, S1=1, S2=0

frequency of output = 4MHz ÷ (256 × 1 × (0+1)) = 15.625KHz
duty of output = [(127+1) ÷ 256] × 100% = 50%

Example 2:

TM2C = 0b0001_1010, Y=4MHz
TM2B = 0b0000_1001, K = 9
TM2S = 0b0_00_00000, S1=1, S2=0

frequency of output = 4MHz ÷ (256 × 1 × (0+1)) = 15.625KHz
duty of output = [(9+1) ÷ 256] × 100% = 3.9%

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

```
void    FPPA0 (void)
{
    .ADJUST_IC    SYSCLK=IHRC/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:

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

Duty of Output = [(K＋1) ÷ 64] × 100%

Where,

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

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

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

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

Example 1:

TM2C = 0b0001_1010, Y=4MHz

TM2B = 0b0011_1111, K=63

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

frequency of output = 4MHz ÷ (64 × 1 × (0+1)) = 62.5KHz

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

Example 2:

TM2C = 0b0001_1010, Y=4MHz

TM2B = 0b0000_0000, K=0

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

Frequency = 4MHz ÷ (64 × 1 × (0+1)) = 62.5KHz

Duty = [(0+1) ÷ 64] × 100% =1.5%

10.3. 11-bit PWM Generation (PWMG0/1/2)

Three 11-bit hardware PWM generators (PWMG0, PWMG1 & PWMG2) are implemented in the PFC460. 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, PC2
(2) PWMG1 – PA4, PB6, PB7, PC3
(3) PWMG2 – PA3, PB2, PB3, PC0

10.3.1. PWM Waveform

A PWM output waveform (Fig. 20) has a time-base (T_{Period} = Time of Period) and a time with output high level (Duty Cycle). The frequency of the PWM output is the inverse of the period ($f_{PWM} = 1/T_{Period}$), the resolution of the PWM is the clock count numbers for one period (N bits resolution, $2^N \times T_{clock} = T_{Period}$).

Besides, selecting code option OPA_PWM can control the generated PWM waveform by the comparator result. Please refer the section of OPA for details.

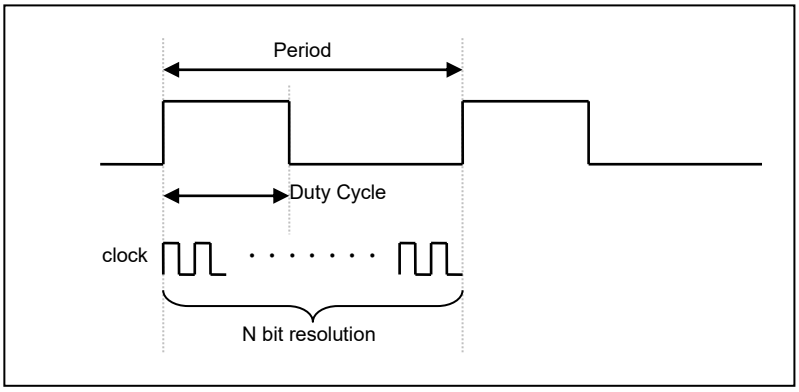


Fig. 20: PWM Output Waveform

10.3.2. Hardware and Timing Diagram

Fig. 21 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*).

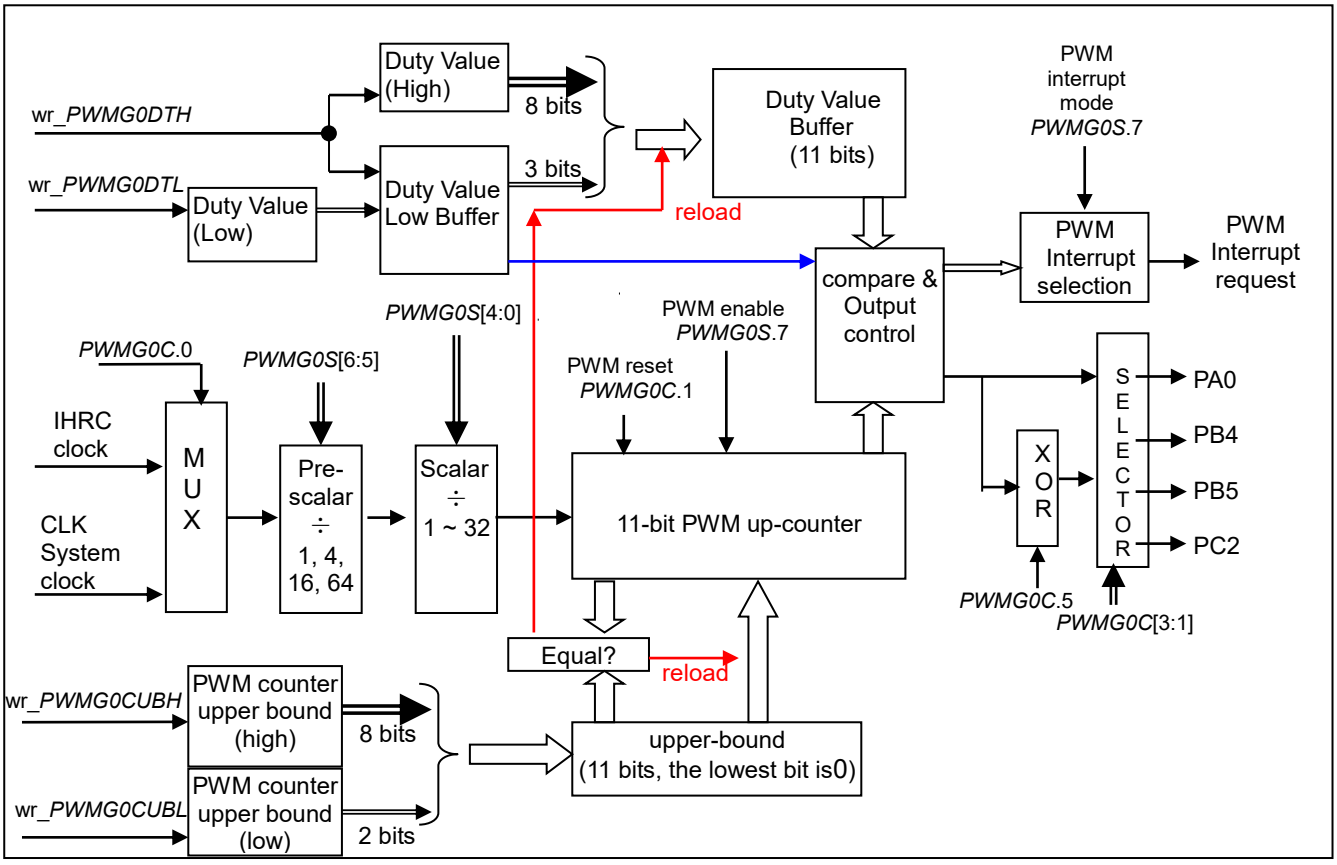


Fig. 21: Hardware Diagram of 11-bit PWM Generator 0 (PWMG0)

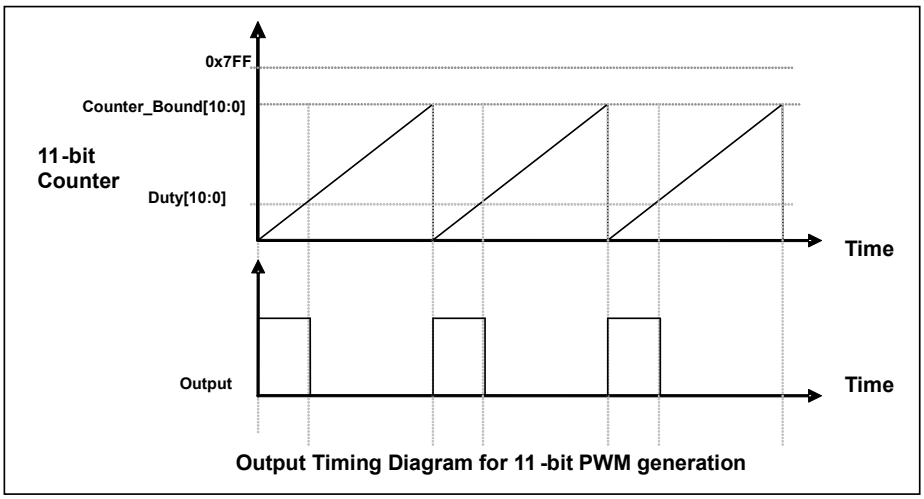


Fig. 22: Output Timing Diagram of 11-bit PWM Generator

10.3.3. Equations for 11-bit PWM Generator

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

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

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

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

Where,

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

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

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

DB0 = Duty_Bound[0] = *PWMGxDTL*[5] （x=0/1/2）

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 = 0x22

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	000	R/W	Select PWM output pin for PWMG0. 000: none 001: PB5 010: PC2 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 = 0x23

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	00	WO	PWMG0 clock pre-scalar 00: ÷1 01: ÷4 10: ÷16 11: ÷64
4 – 0	00000	WO	PWMG0 clock divider

10.3.4.3. PWMG0 Duty Value High Register (*PWMG0DTH*), address = 0x50

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 = 0x51

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 = 0x52

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 = 0x53

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 = 0x24

Bit	Reset	R/W	Description
7	0	R/W	Enable PWMG1. 0 / 1: disable / enable
6	-	RO	Output status of PWMG1 generator
5	0	R/W	Enable to inverse the polarity of PWMG1 generator 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	000	R/W	Select PWMG1 output pin. 000: none 001: PB6 010: PC3 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. PWMG2 Control Register (*PWMG2C*), address = 0x26

Bit	Reset	R/W	Description
7	0	R/W	Enable PWMG2. 0 / 1: disable / enable
6	-	RO	Output status of PWMG2 generator.
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: none 001: PB3 010: PC0 011: PA3 100: PB2 Others: reserved
0	0	R/W	Clock source of PWMG2. 0: CLK 1: IHRC or IHRC*2 (by code option PWM_source)

10.3.4.9. PWMG1/*PWMG2* Scalar Register (*PWMG1S/PWMG2S*), address = 0x25/0x27

Bit	Reset	R/W	Description
7	0	WO	PWMG1/PWMG2 interrupt mode. 0: Generate interrupt when counter matches the duty value 1: Generate interrupt when counter is 0
6 – 5	00	WO	PWMG1/PWMG2 clock pre-scalar. 00 : ÷1 01 : ÷4 10 : ÷16 11 : ÷64
4 – 0	00000	WO	PWMG1/PWMG2 clock divider.

10.3.4.10. PWMG1/*PWMG2* Duty Value High Register (*PWMG1DTH/PWMG2DTH*), address = 0x54/0x58

Bit	Reset	R/W	Description
7 – 0	0x00	WO	Duty values bit[10:3] of PWMG1/PWMG2.

10.3.4.11. PWMG1/*PWMG2* Duty Value Low Register (*PWMG1DTL/PWMG2DTL*), address = 0x55/0x59

Bit	Reset	R/W	Description
7 – 5	000	WO	Duty values bit[2:0] of PWMG1/PWMG2.
4 – 0	-	-	Reserved.

Note: It’s necessary to write *PWMG1DTL* Register before writing *PWMG1DTH* Register. PWMG2 should be set in the same method.

10.3.4.12. PWMG1/*PWMG2* Counter Upper Bound High Register(*PWMG1CUBH/PWMG2CUBH*), address = 0x56/0x5A

Bit	Reset	R/W	Description
7 - 0	0x00	WO	Bit[10:3] of PWMG1/PWMG2 counter upper bound.

10.3.4.13. PWMG1/*PWMG2* Counter Upper Bound Low Register (*PWMG1CUBL/PWMG2CUBL*), address = 0x57/0x5B

Bit	Reset	R/W	Description
7 – 6	00	WO	Bit[2:1] of PWMG1/PWMG2 counter upper bound.
5	0	WO	Bit[0] of PWMG1/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,SYSCLK;       //    PWMG0 output the PWM0 waveform to PA0
  $ PWMG0S INTR_AT_DUTY,/1,/1;
```

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

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

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

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

The PWM0 / PWM1 waveform obtained by the above program is shown in Fig. 23.

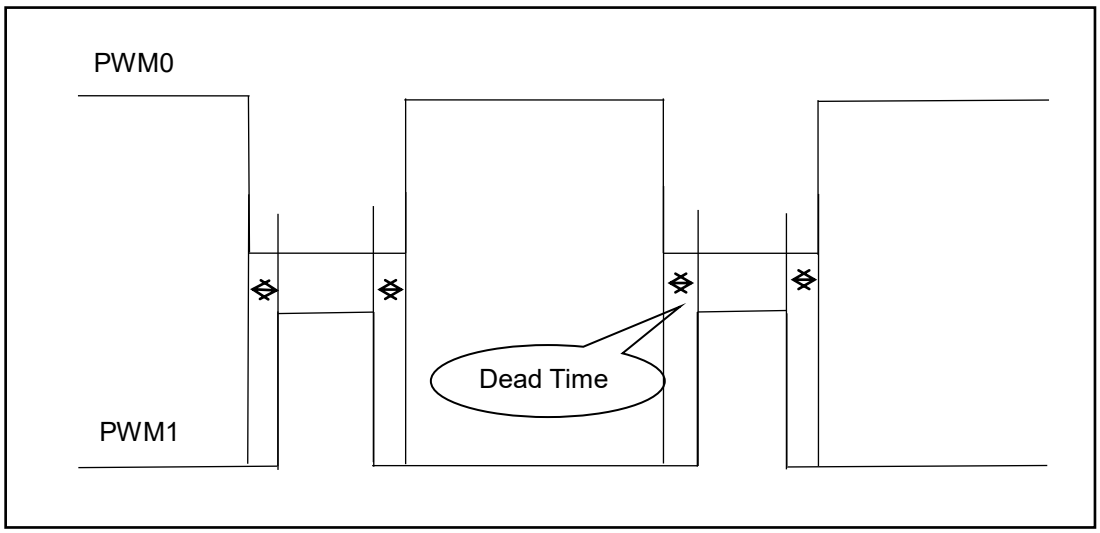


Fig. 23: Two complementary PWM waveforms with dead zones

Users can modify the **dead_zone_R** and **dead_zone_F** values in the program to adjust the dead-time. Table 12 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 12: 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
...	...

Table 13: The parameter regulation of dead-time

10.4. 11-bit SuLED LPWM Generation (LPWMG0/1/2)

Three 11-bit SuLED(Super LED) hardware LPWM generators are built in the PFC460. Their individual outputs are listed as below:

- LPWMG0 – PA0, PB4, PB5, PB6, PC2
- LPWMG1 – PA4, PB6, PB7, PC3
- LPWMG2 – PA3, PA5, PB2, PB3, PB5, PC0

LPWMG0 and LPWMG2 share output pin PB5, but they cannot output LPWM to PB5 at the same time;

At the same way, LPWMG0 and LPWMG1 share output pin PB6, but they cannot output LPWM to PB6 at the same time.

10.4.1. LPWM Waveform

A LPWM output waveform (Fig. 24) has a time-base (T_{Period} = Time of Period) and a time with output high level (Duty Cycle). The frequency of the LPWM output is the inverse of the period ($f_{LPWM} = 1/T_{Period}$),

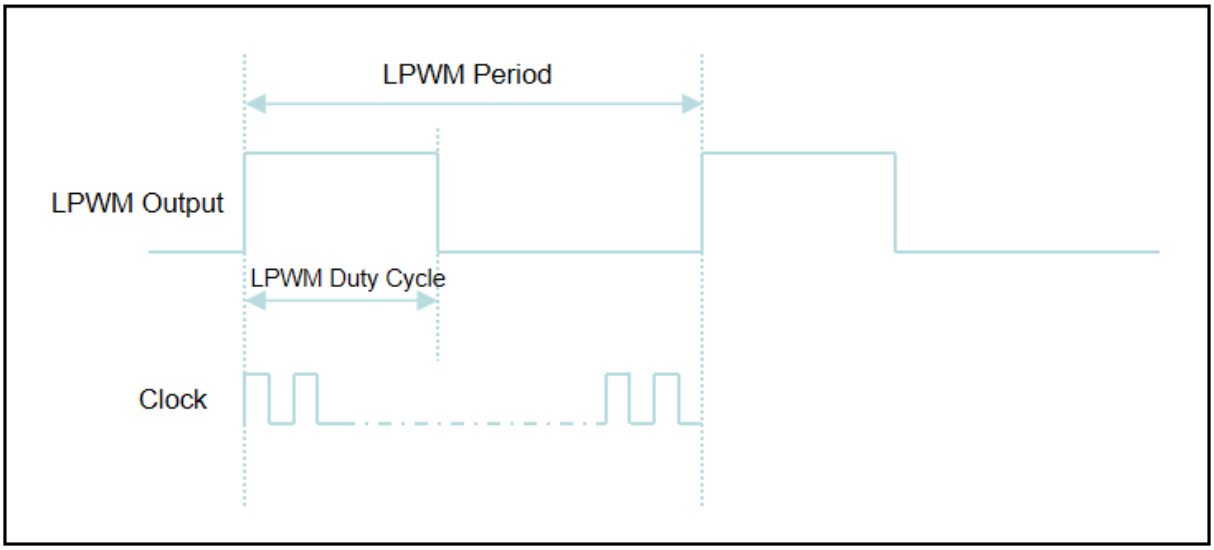


Fig. 24: PWM Output Waveform

10.4.2. Hardware Diagram

Fig. 25 shows the hardware diagram of 11-bit LPWM generation. These triple of LPWM generators use UP-Counter together and clock source selection switch to generate the time-base. So, the starting point(rising) of the LPWM cycle is synchronized, and the clock source can be IHRC or system clock. The LPWM output pin is selected by register *LPWMGxC.* The period of LPWM waveform is defined in the LPWM upper bond high and low registers, the duty cycle of LPWM waveform is defined in the LPWM duty high and low registers.

The two attached logic gates OR and XOR in the LPWMG0 channel are used to generate complementary non-overlapping switches with dead zones to control waveforms.

Besides, selecting code option OPA_PWM can control the generated LPWM waveform by the comparator result. Please refer the section of OPA for details.

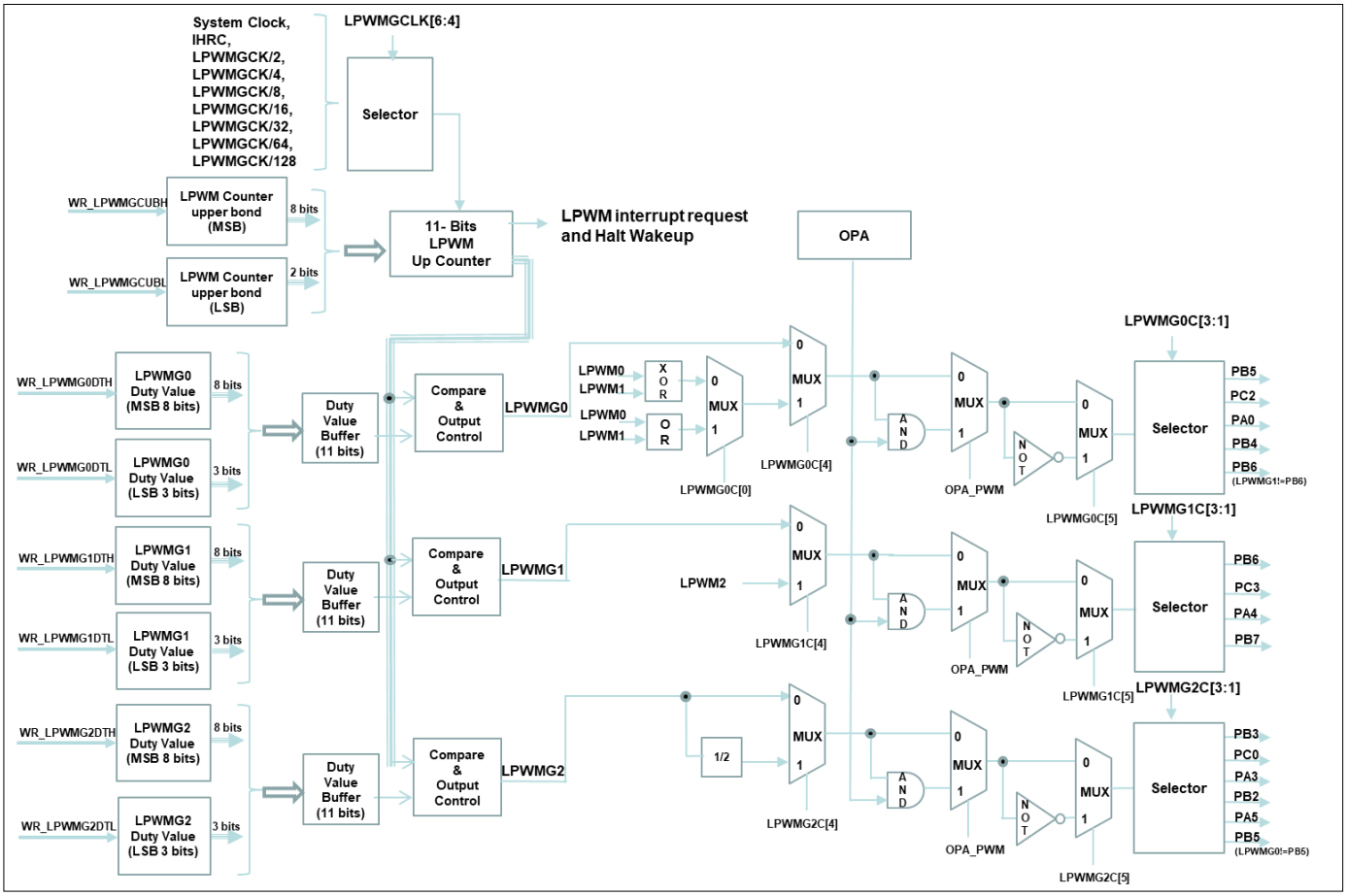


Fig. 25: Hardware Diagram of three 11-bit LPWM Generators

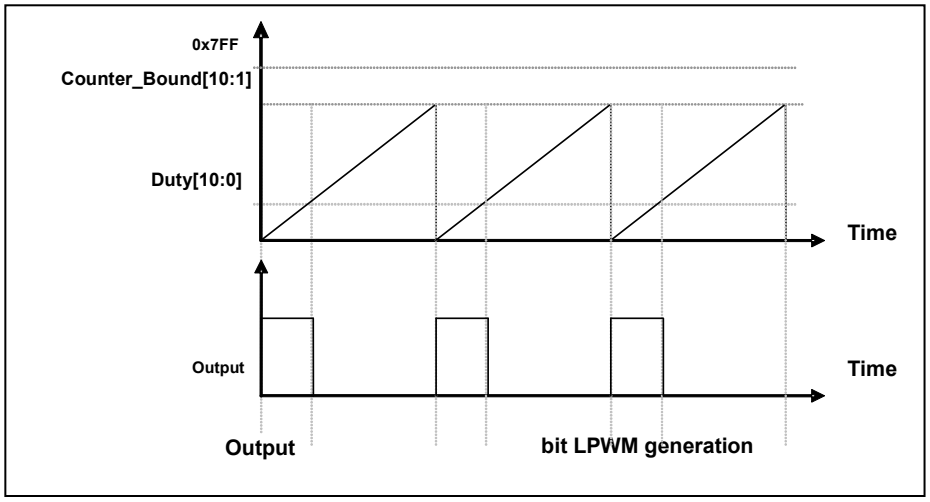


Fig. 26: Output Timing Diagram of 11-bit LPWM Generator

10.4.3. Equations for 11-bit LPWM Generator

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

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

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

Where,

P=*LPWMGCLK*[6:4]; pre-scalar **P**=1,2,4,8,16,32,64,128

DB10_1 = Duty_Bound[10:1] = {*LPWMGxDTH* [7:0],*LPWMGxDTL*[7:6]},（x=0/1/2） duty bound

DB0 = Duty_Bound[0] = *LPWMGxDTL*[5] （x=0/1/2）

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

10.4.4. 11bit LPWM Related Registers

10.4.4.1. LPWMG0 Control Register (*LPWMG0C*), address= 0x0C

Bit	Reset	R/W	Description
7	-	-	Reserved.
6	-	RO	Output status of LPWMG0 generator.
5	0	WO	Enable to inverse the polarity of LPWMG0 generator output. 0 / 1: disable / enable.
4	0	WO	LPWMG0 output selection. 0: LPWMG0 output 1: LPWMG0 XOR LPWMG1 or LPWMG0 OR LPWMG1 (by bit 0 of LPWMG0C)
3 – 1	000	R/W	LPWMG0 output pin selection. 000: none 001: PB5 010: PC2 011: PA0 100: PB4 101: PB6 (Only enabled when PB6 is not used as LPWMG1 output) Others: reserved
0	0	R/W	LPWMG0 output pre-selection. 0: LPWMG0 XOR LPWMG1 1: LPWMG0 OR LPWMG1

10.4.4.2. LPWMG1 Control Register (*LPWMG1C*), address = 0x0D

Bit	Reset	R/W	Description
7	-	-	Reserved.
6	-	RO	Output status of LPWMG1 generator.
5	0	R/W	Enable to inverse the polarity of LPWMG1 generator output. 0 / 1: disable / enable.
4	0	R/W	LPWMG1 output selection. 0: LPWMG1 1: LPWMG2
3 – 1	000	R/W	LPWMG1 output pin selection. 000: none 001: PB6 010: PC3 011: PA4 100: PB7 1xx: reserved
0	-	R/W	Reserved

10.4.4.3. LPWMG2 Control Register (*LPWMG2C*), address = 0x0E

Bit	Reset	R/W	Description
7	-	-	Reserved.
6	-	RO	Output status of LPWMG2 generator.
5	0	R/W	Enable to inverse the polarity of LPWMG2 generator output. 0 / 1: disable / enable.
4	0	R/W	LPWMG2 output selection. 0: LPWMG2 1: LPWMG2 ÷2
3 – 1	000	R/W	LPWMG2 output pin selection. 000: none 001: PB3 010: PC0 011: PA3 100: PB2 101: PA5 110: PB5 (Only enabled when PB5 is not used as LPWMG0 output) 111: reserved
0	-	R/W	Reserved

10.4.4.4. LPWMG Clock Register (*LPWMGCLK*), address = 0x67

Bit	Reset	R/W	Description
7	0	RO	LPWMG disable/enable. 0: LPWMG disable 1: LPWMG enable
6 – 4	000	RO	LPWMG clock pre-scalar. 000: ÷1 001: ÷2 010: ÷4 011: ÷8 100: ÷16 101: ÷32 110: ÷64 111: ÷128
3 – 1	-	-	Reserved
0	0	RO	LPWMG clock source selection. 0: system clock 1: IHRC or IHRC*2 (by code option LPWM_Source)

10.4.4.5. LPWMG Counter Upper Bound High Register (*LPWMGCUBH*), address = 0x68

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

10.4.4.6. LPWMG Counter Upper Bound Low Register (LPWMGCUBL), address = 0x69

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

10.4.4.7. LPWMG0/1/2 Duty Value High Register (*LPWMGxDTH, x=0/1/2*), address = 0x6A/0x6C/0x6E

Bit	Reset	R/W	Description
7 – 0	-	RO	Duty values bit[10:3] of LPWMG0/LPWMG1/LPWMG2.

10.4.4.8. LPWMG0/1/2 Duty Value Low Register (*LPWMGxDTL, x=0/1/2*), address = 0x6B/0x6D/0x6F

Bit	Reset	R/W	Description
7 – 5	-	RO	Duty values bit [2:0] of LPWMG0/LPWMG1/LPWMG2.
4 – 0	-	-	Reserved.

Note: It's necessary to write *LPWMGxDTL* Register before writing *LPWMGxDTH* Register.(x=0/1/2)

10.4.5. Examples of LPWM Waveforms with Complementary Dead Zones

Based on the specific 11 bit SuLED LPWM architecture of PFC460, here we employ LPWM2 output and LPWM0 inverse output after LPWM0 xor LPWM1 to generate two LPWM waveforms with complementary dead zones. Example program is as follows:

```
#define   dead_zone     10          //   dead time = 10% * (1/LPWM_Frequency) us
#define   LPWM_Pulse    50          //   set 50% as LPWM duty cycle

#define   LPWM_Pulse_1  35          //   set 35% as LPWM duty cycle
#define   LPWM_Pulse_2  60          //   set 60% as LPWM duty cycle
#define   switch_time   400*2       //   adjusting switch time
//Note：To avoid noise, switch_time must be a multiple of LPWM period. In this example LPWM period = 400us,
// so switch_time = 400*2 us.

void    FPPA0 (void)
{
.ADJUST_IC SYSCLK=IHRC/16, IHRC=16MHz, VDD=5V;
//******* Generating fixed duty cycle waveform ************************
//------- Set the counter upper bound and duty cycle ---------------------
LPWMG0DTL     =     0x00;
LPWMG0DTH     =     LPWM_Pulse + dead_zone;
LPWMG1DTL     =     0x00;
```

```
LPWMG1DTH    =    dead_zone;        // After LPWMG0 xor LPWMG, LPWM duty cycle=LPWM_Pulse%

LPWMG2DTL    =    0x00;
LPWMG2DTH    =    LPWM_Pulse + dead_zone*2;

LPWMGCUBL    =    0x00;
LPWMGCUBH    =    100;
//---- Configure clock and pre-scalar ----------
$ LPWMGCLK      Enable, /1, sysclk;
//------- Output control   ----------------------------
$ LPWMG0C  Enable,Inverse,LPWM_Gen,PA0,gen_xor;     //     After LPWMG0 xor LPWMG,
                                                    //    output the inversed waveform through PA0
$ LPWMG1C  Enable, LPWMG1,disable;                  //     disable LPWMG1 output
$ LPWMG2C      Enable, PA3;                         //     output LPWMG2 waveform through PA3

while(1)
{
   //******** Switching duty cycle *********************************
      // To avoid the possible instant disappearance of dead zone, user should comply with the following
      // instruction sequence.
      // When increase the duty cycle: 50%/60%   →   35%
   LPWMG0DTL   =    0x00;
   LPWMG0DTH   =    LPWM_Pulse_1 + dead_zone;
   LPWMG2DTL   =    0x00;
   LPWMG2DTH   =    LPWM_Pulse_1 + dead_zone*2;
   .delay      switch_time

   //When decrease the duty cycle: 35% → 60%
     LPWMG2DTL =    0x00;
     LPWMG2DTH=     LPWM_Pulse_2 + dead_zone*2;
     LPWMG0DTL =    0x00;
     LPWMG0DTH=     LPWM_Pulse_2 + dead_zone;
     .delay      switch_time
   }
   }
```

The following figures show the waveforms at different condition.

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

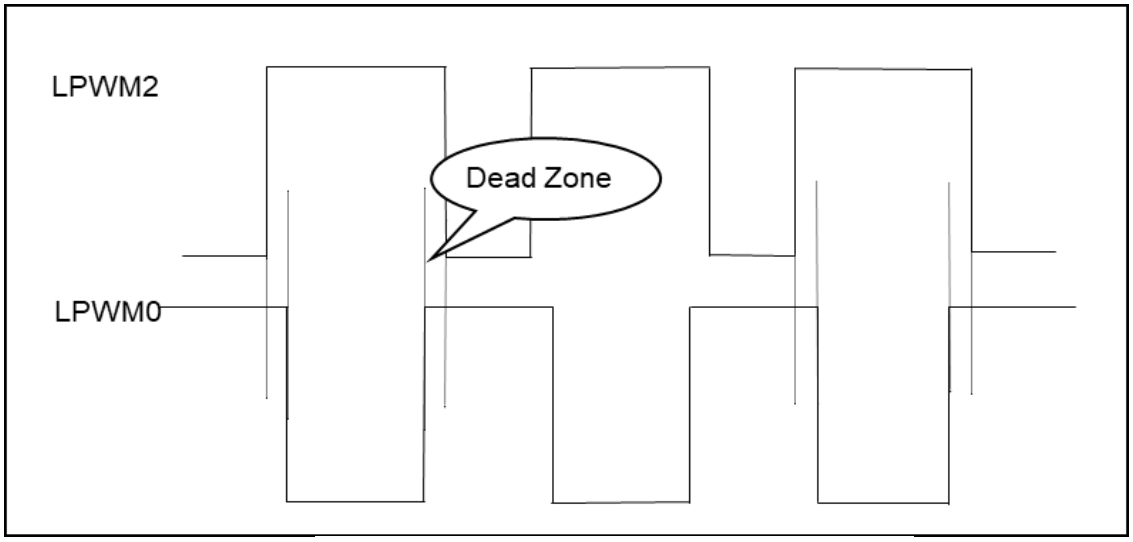


Fig.27: Complementary LPWM waveform with dead zones

2. PWM waveform when switching two duty cycles:

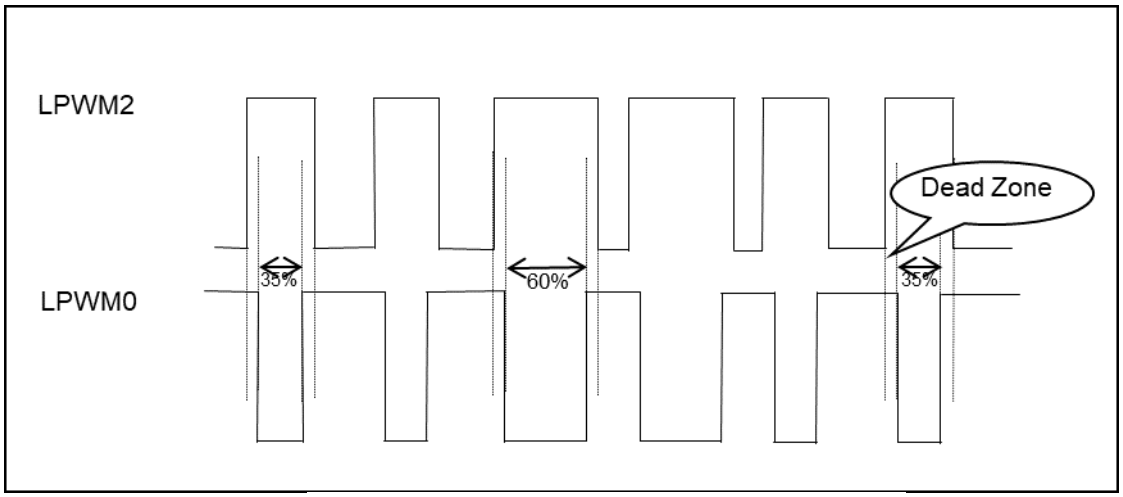


Fig.28: Complementary LPWM waveform with dead zones

User can find that above example only provides dead zone where LPWM are both in high. If need dead zone where LPWM are both in low, you can realize it by resetting each control register's Inverse like:

```
$ LPWMG0C  Enable,LPWM_Gen,PA0,gen_xor;
$ LPWMG2C   Enable, Inverse, PA3;
```

11. Special Functions

11.1. Comparator

One hardware comparator is built inside the PFC460; Fig. 29 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.

(6) control various PWM outputs, please refer to *GPC2PWM* register for detailed usage.

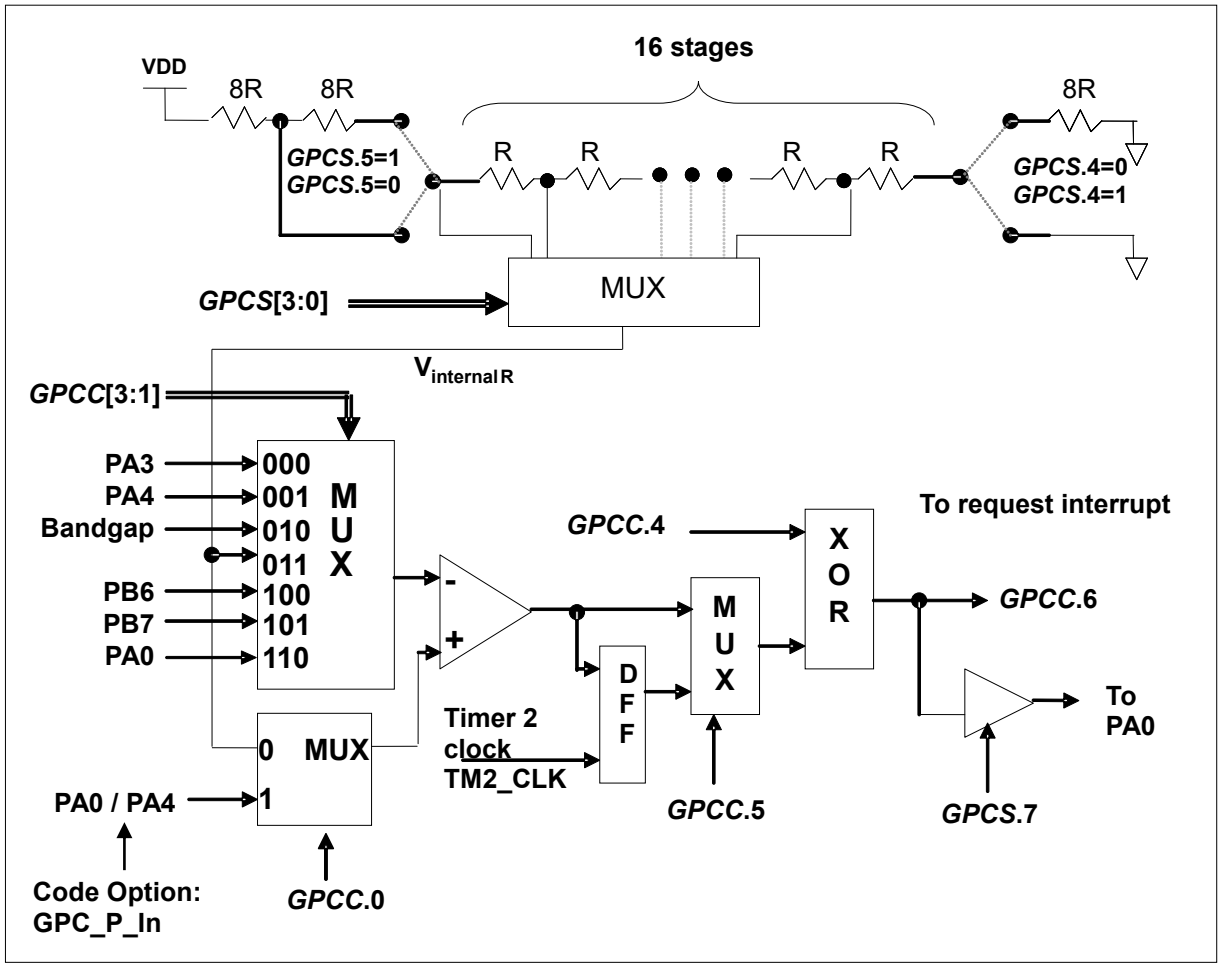


Fig. 29: Hardware diagram of comparator

11.1.1. Comparator Control Register (*GPCC*), address = 0x35

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 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 = 0x36

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. Comparator Results Triggle PWM Control Register(*GPC2PWM*), address = 0x43

Bit	Reset	R/W	Description
7	0	WO	Comparator results control Timer2 PWM output. 0/1: disable/enable
6	0	WO	Comparator results control Timer3 PWM output. 0/1: disable/enable
5	0	WO	Comparator results control PWMG2 PWM output. 0/1: disable/enable
4	0	WO	Comparator results control PWMG1 PWM output. 0/1: disable/enable
3	0	WO	Comparator results control PWMG0 PWM output. 0/1: disable/enable
2	0	WO	Comparator results control LPWMG2 PWM output. 0/1: disable/enable
1	0	WO	Comparator results control LPWMG1 PWM output. 0/1: disable/enable
0	0	WO	Comparator results control LPWMG0 PWM output. 0/1: disable/enable

When the output result of the comparator is 1, disable PWM output; when the comparator result is 0, enable PWM output. Please refer to Fig. 30 for the principle of comparator control PWM output.

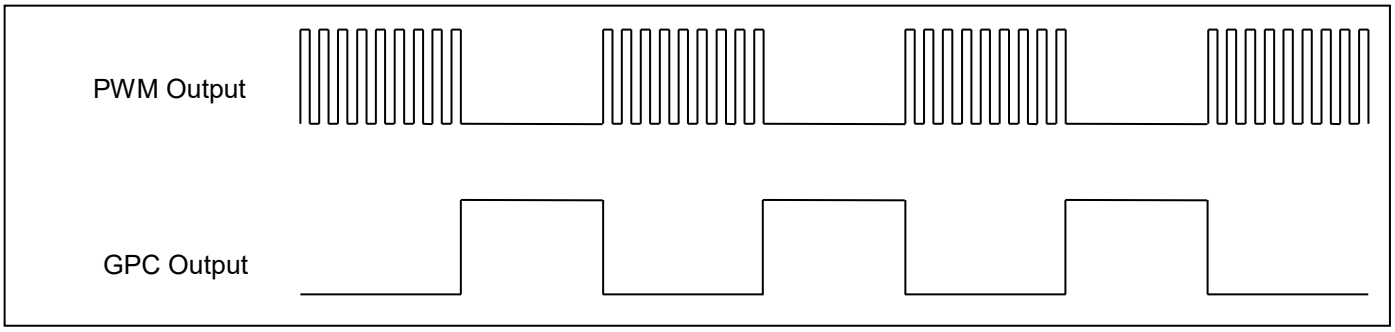


Fig .30: Comparator results control PWM output

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

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

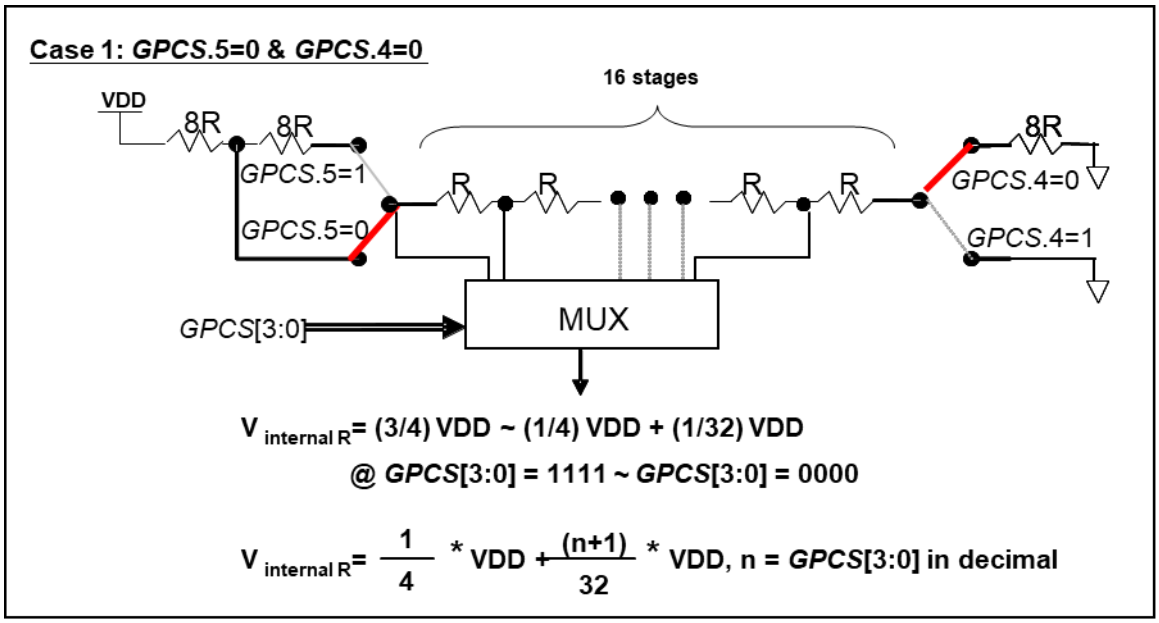


Fig. 31: $V_{internal\ R}$ hardware connection if *GPCS*.5=0 and *GPCS*.4=0

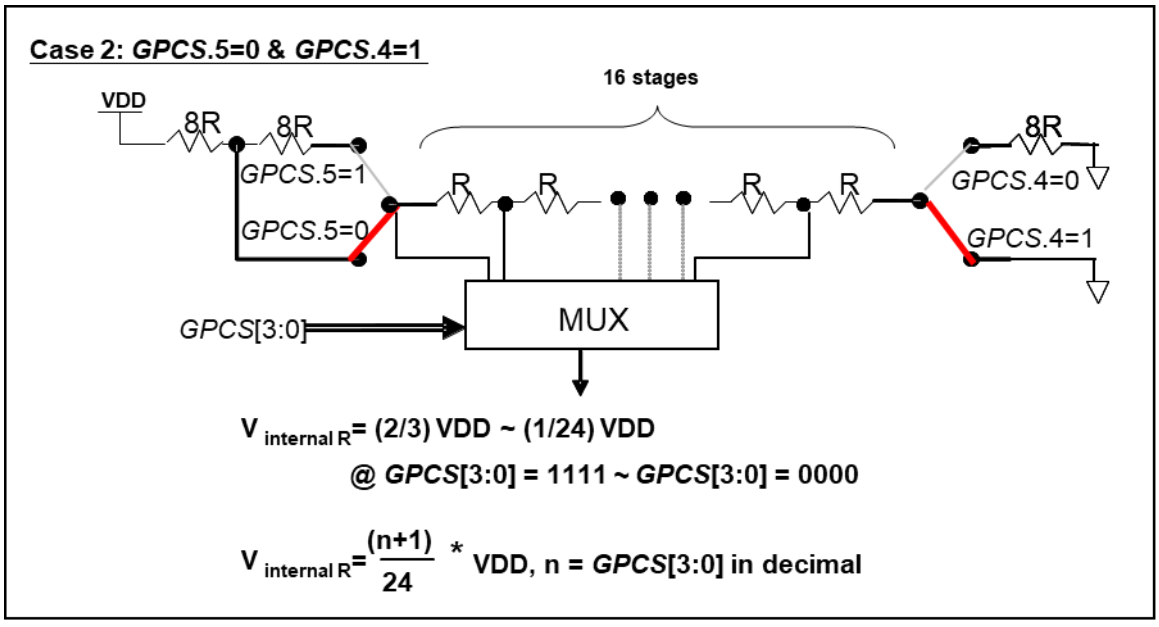


Fig. 32: $V_{internal\ R}$ hardware connection if *GPCS*.5=0 and *GPCS*.4=1

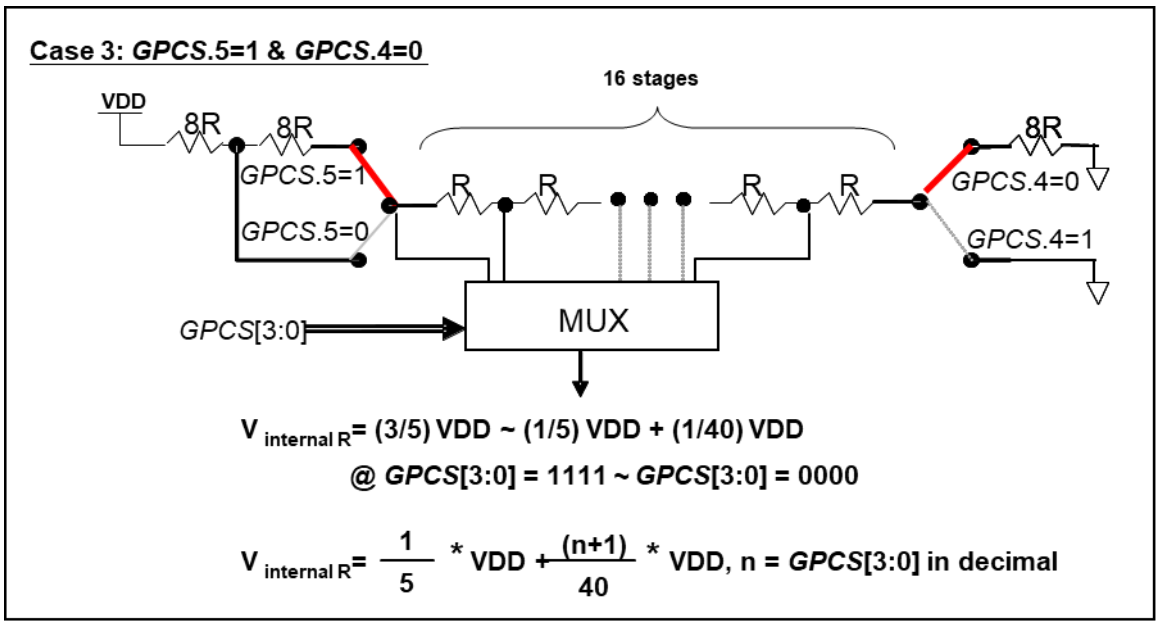


Fig. 33: $V_{internal\ R}$ hardware connection if *GPCS*.5=1 and *GPCS*.4=0

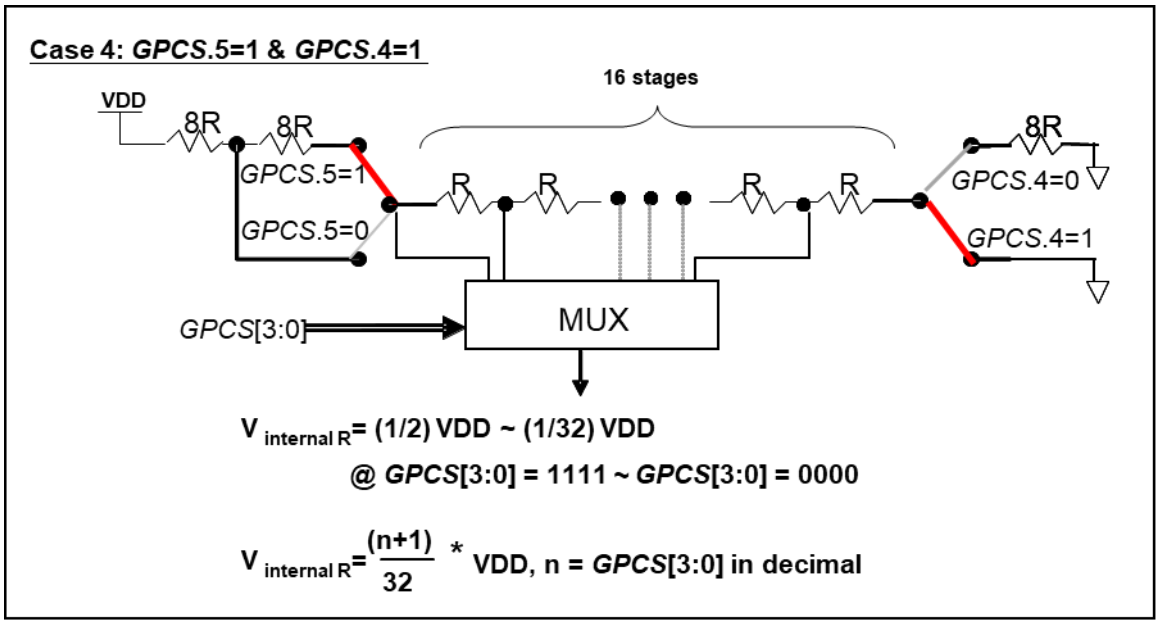


Fig. 34: $V_{internal\ R}$ hardware connection if *GPCS*.5=1 and *GPCS*.4=1

11.1.5. Using the Comparator

Case 1:

Choosing PA3 as minus input and Vinternal R with (18/32)*VDD voltage level as plus input. Vinternal R is configured as the above Figure "GPCS[5:4] = 2b'00" and GPCS[3:0] = 4b'1001 (n=9) to have Vinternal R = (1/4)*VDD + [(9+1)/32]*VDD = [(9+9)/32]*VDD = (18/32)*VDD.

```
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.

11.1.6. 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 $V_{internal\ R}$ is selected as plus input, the supply voltage of $V_{internal\ R}$ is VDD, the VDD voltage level can be detected by adjusting the voltage level of $V_{internal\ R}$ to compare with Bandgap.

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

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

Case 1:

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

11.2. VDD/2 Bias Voltage Generator

PFC460 provides a VDD/2 bias voltage generator, which can be used as COM function for LCD application. It also has two sets of five optional VDD/2 output pins: LCD_B01256 means that the VDD/2 bias voltage output through PB0/PB1/PB2/PB5/PB6. At the same time, LCD_A034_B01 output VDD/2 bias voltage through PA0/PA3/PA4/PB0/PB1. This function can be enabled or disable through the register *MISC*.4, and the output pin is selected through *MISC*.3.

MISC Register (*MISC*), address = 0x49			
Bit	**Reset**	**R/W**	**Description**
7 – 5	-	-	Reserved. (keep 0 for future compatibility)
6	-	WO	Reserved, please set to 1 by manually.
5	0	WO	Fast wake-up function.
4	**0**	**WO**	**Enable VDD/2 bias voltage generator 0 / 1 : Disable / Enable**
3	**0**	**WO**	**Select VDD/2 output pin** **0: LCD_B01256, including PB0/PB1/PB2/PB5/PB6** **1: LCD_A034_B01, including PA0/PA3/PA4/PB0/PB1**
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.35 shows how to use this function.

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

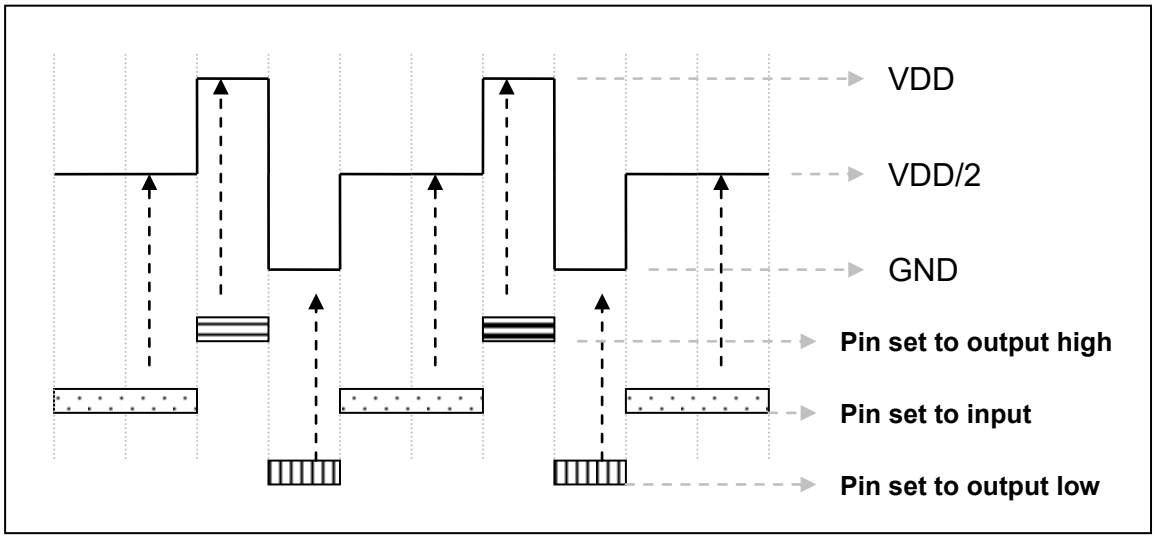


Fig. 35: Using VDD/2 bias voltage generator

11.3. Operational Amplifier (OPA) module

An Operational Amplifier (OPA) is built-in in the PFC460, 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.

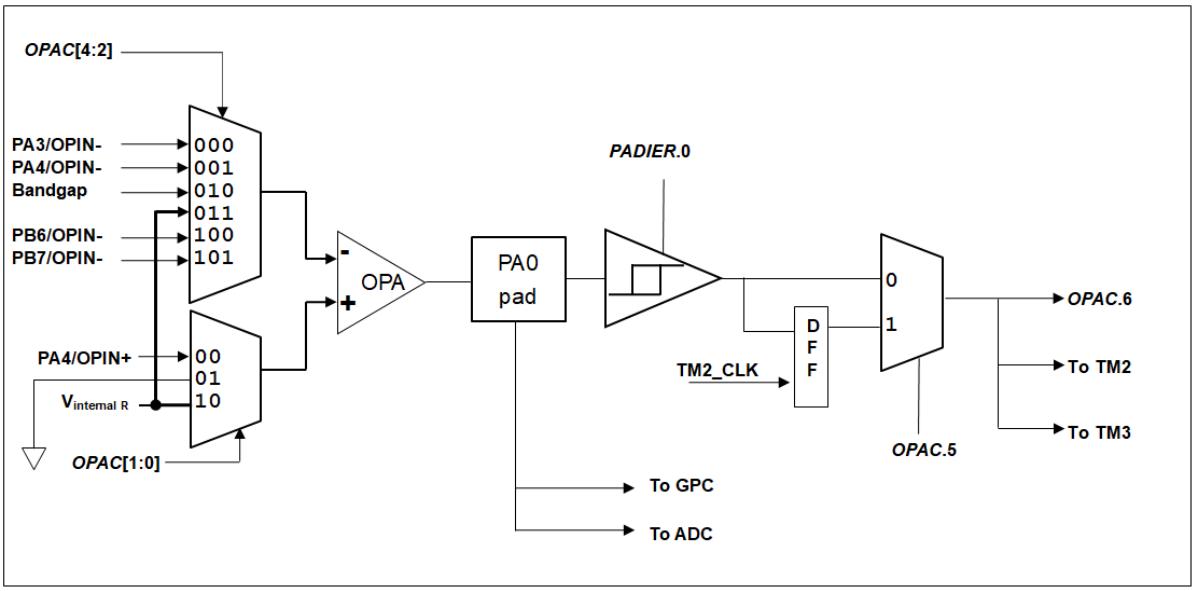


Fig. 36: hardware diagram of OPA

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.

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.

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

The Code Option OPA_PWM means that the PWM waveform is controlled by the output result of the OPA comparator mode. After OPA comparator mode is enabled, disable PWM output when the OPA output result is 1; enable PWM output when the OPA output result is 0. As shown in Fig. 37. This function is valid for 8bit/11bit PWM.

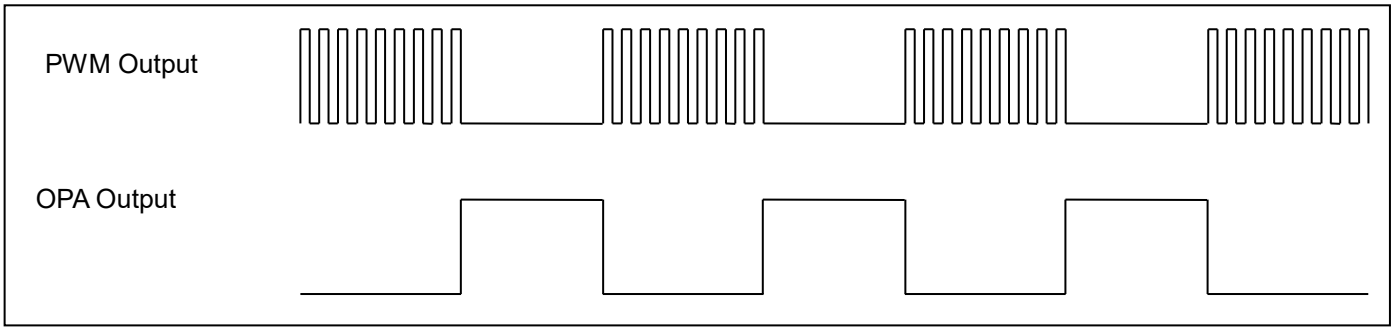


Fig. 37: OPA comparator mode control PWM output

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 = 0x33

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 = 0x34

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

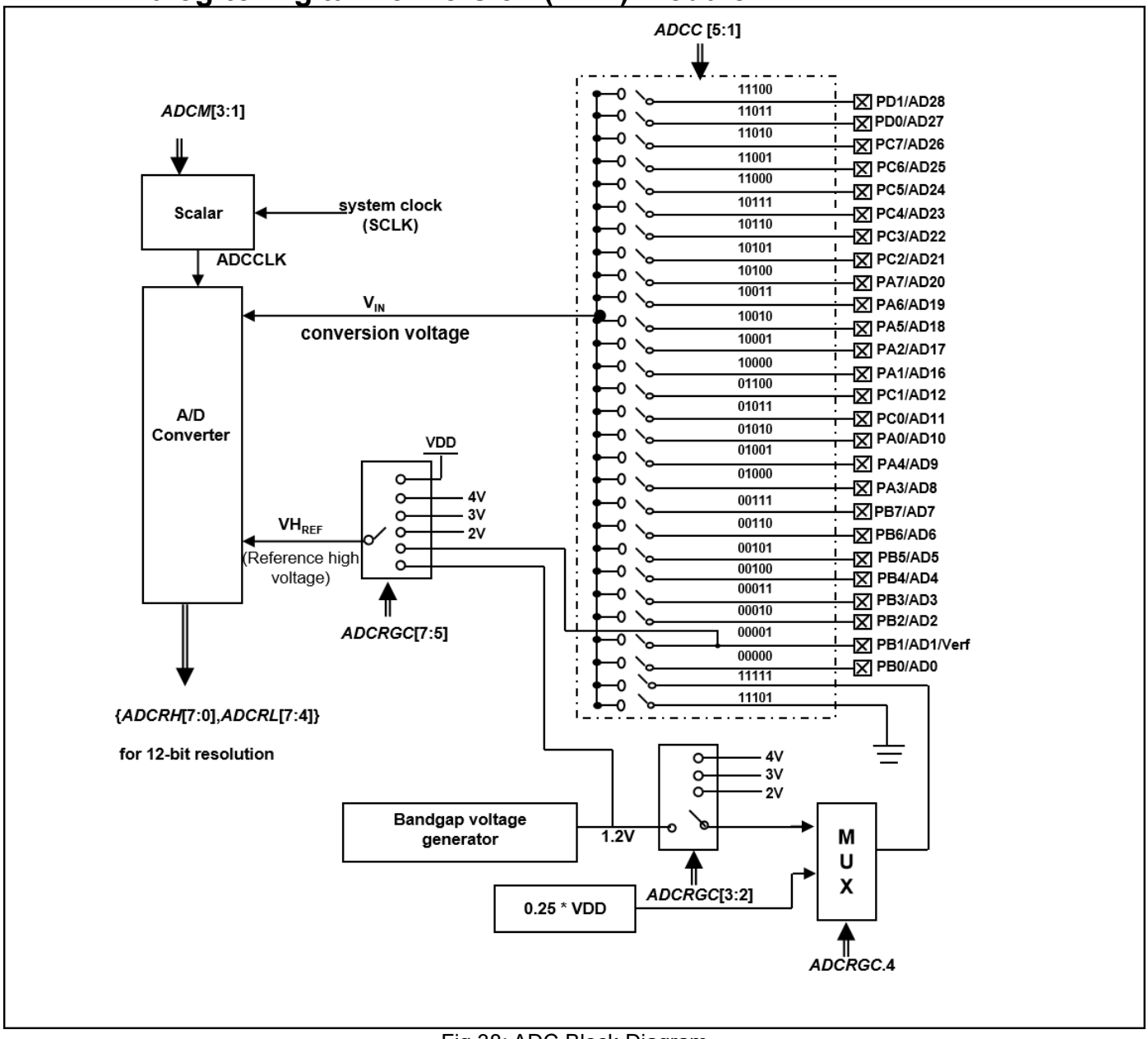


Fig.38: ADC Block Diagram

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 ***PxDIER*** 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.

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

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

Set '1' to addc.6 to start the conversion and check whether addc.6 is '1'

(8) Read the ADC result registers:

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

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

11.4.1. The input requirement for AD conversion

For the AD conversion to meet its specified accuracy, the charge holding capacitor (CHOLD) must be allowed to fully charge to the voltage reference high level and discharge to the voltage reference low level. The analog input model is shown as Fig.39, the signal driving source impedance (Rs) and the internal sampling switch impedance (Rss) will affect the required time to charge the capacitor CHOLD directly. The internal sampling switch impedance may vary with ADC supply voltage; the signal driving source impedance will affect accuracy of analog input signal. User must ensure the measured signal is stable before sampling; therefore, the maximum signal driving source impedance is highly dependent on the frequency of signal to be measured. The recommended maximum impedance for analog driving source is about 10KΩ under 500KHz input frequency.

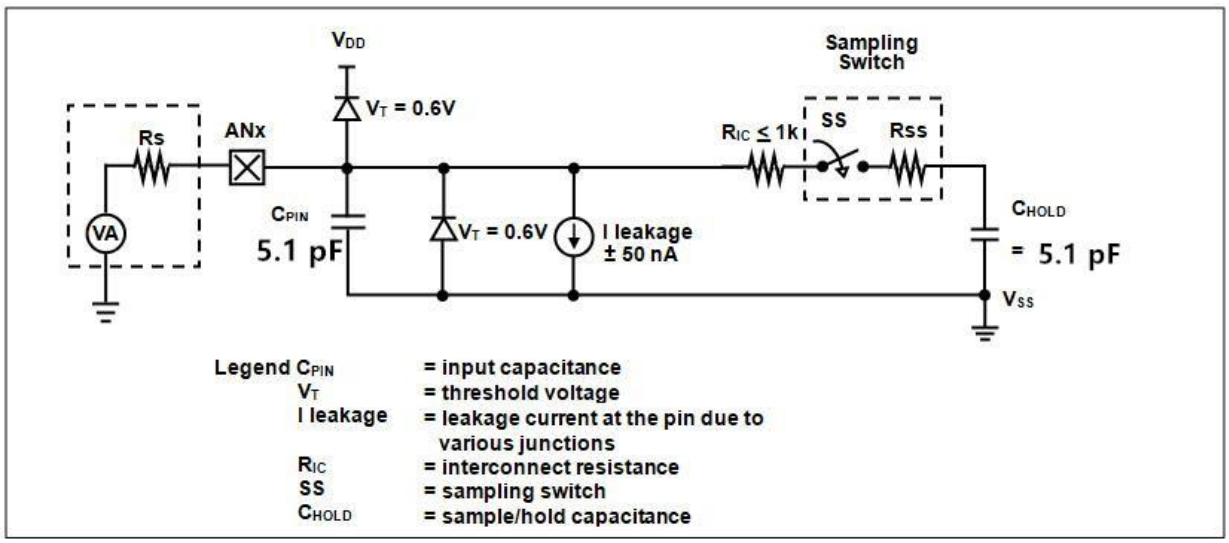


Fig.39: Analog Input Model

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

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, 1.20V bandgap 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÷1 to CLK÷128 (CLK is the system clock). Due to the signal acquisition time T_{ACQ} is one clock period of ADCLK, the ADCLK must meet that requirement. The recommended ADC clock is to operate at 2us.

11.4.4. Configure the analog pins

There are 28 analog signals can be selected for AD conversion, 26 analog input signals come from external pins and one is from internal bandgap reference voltage or 0.25*V_{DD}, the other uses GND as the ADC input channel. There are 4 voltage levels selectable for the internal bandgap reference, they are 1.2V, 2V, 3V and 4V. For external pins, those pins defined for analog input should disable the digital input function to avoid leakage current at the digital circuit (set the corresponding bit of ***PxDIER*** register to be 0).

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

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

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

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
```

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 = 0x20

Bit	Reset	R/W	Description	
7	0	R/W	Enable ADC function. 0/1: Disable/Enable.	
6	0	R/W	ADC process control bit. Read "1" to indicate the ADC is ready or end of conversion.	
5 – 1	00000	R/W	Channel selector. These five bits are used to select input signal for AD conversion.	
			00000: PB0/AD0, 00001: PB1/AD1, 00010: PB2/AD2, 00011: PB3/AD3, 00100: PB4/AD4, 00101: PB5/AD5, 00110: PB6/AD6, 00111: PB7/AD7, 01000: PA3/AD8, 01001: PA4/AD9, 01010: PA0/AD10, 01011: PC0/AD11 01100: PC1/AD12 01101: reserved 01110: reserved 01111: reserved	10000: PA1/AD16 10001: PA2/AD17 10010: PA5/AD18 10011: PA6/AD19 10100: PA7/AD20 10101: PC2/AD21 10110: PC3/AD22 10111: PC4/AD23 11000: PC5/AD24 11001: PC6/AD25 11010: PC7/AD26 11011: PD0/AD27 11100: PD1/AD28 11101: GND 11110: reserved 11111: (Channel F) Bandgap reference voltage or 0.25*V_{DD}
0	-	-	Reserved. (keep 0 for future compatibility)	

11.4.6.2. ADC Mode Register (*ADCM*), address = 0x21

Bit	Reset	R/W	Description
7 – 4	-	-	Reserved
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 = 0x42

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*V_{DD}. The deviation is within ±0.01*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 = 0x4A

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 = 0x4B

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 multiplicator must be put on *ACC* and register *MULOP* (0x08); After *mul* command, the high byte result will be put on register *MULRH* (0x09) and low byte result on *ACC*. The hardware diagram of this multiplier is shown as Fig.40.

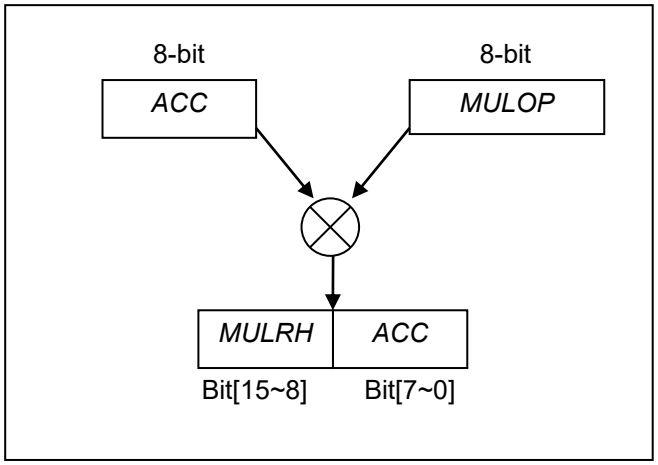


Fig. 40: Block diagram of hardware multiplier

11.5.1. Multiplier Operand Register (*MULOP*), address = 0x2C

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

11.5.2. Multiplier Result High Byte Register (*MULRH*), address = 0x2D

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

11.6. Touch Function

A touch detecting circuit is included in PFC460. Its functional block diagram is shown as Fig. 41.

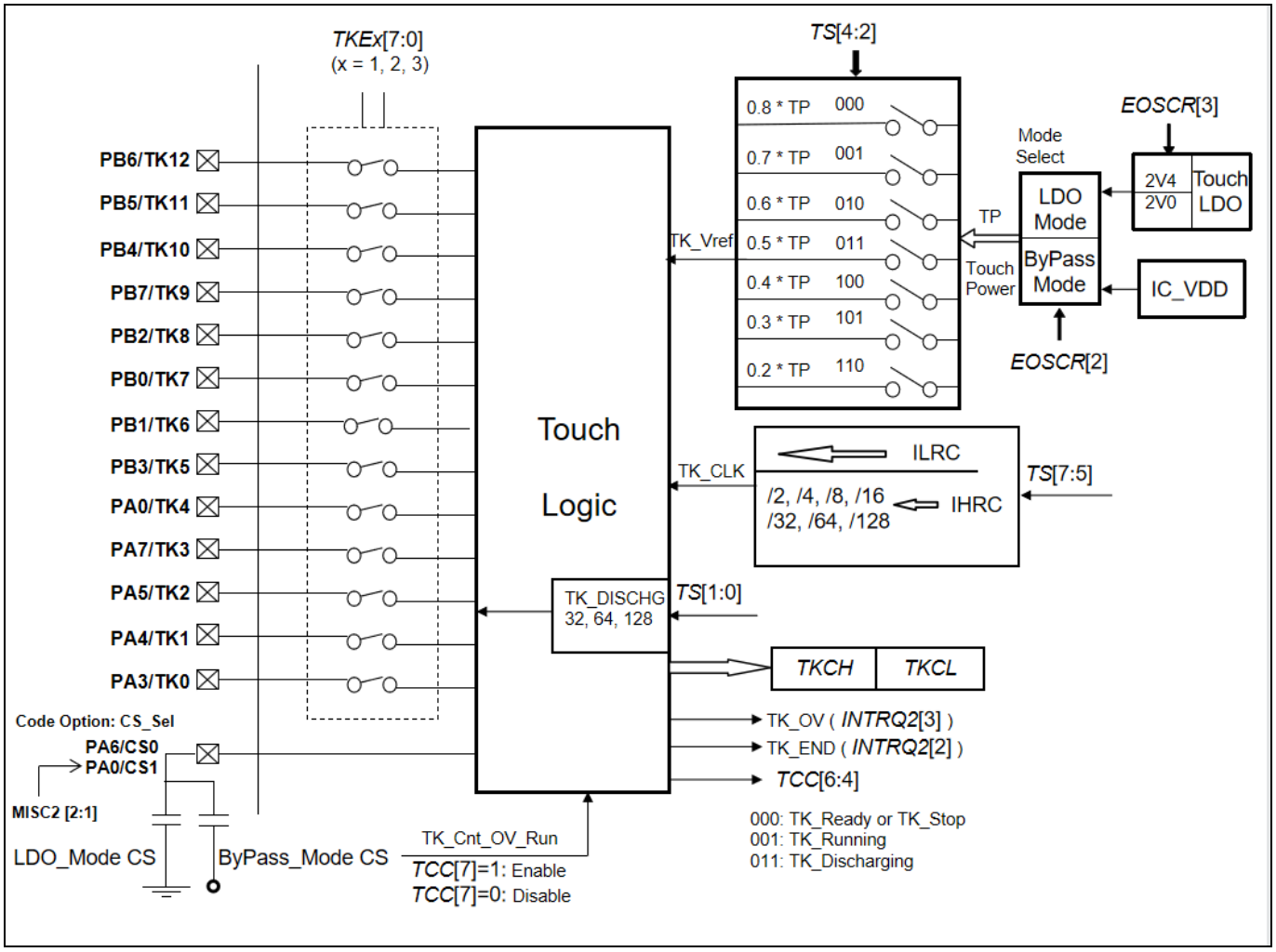


Fig. 41: Functional block diagram of the touch detecting circuit

The Touch detecting circuit in PFC460 applies the method of capacitive sensing, detecting the capacitive virtual ground effect of a finger, or the capacitance between sensors.

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

1. Set *ESOCR*[2] to select ByPass/LDO mode.
2. when ByPass mode is selected, the touch module power supply is chip VDD, an accurate and low-leakage external capacitor CS is required to be connect between CS pin and VDD.
3. when the LDO mode is selected, the touch module power supply can be provided by 2.4V/2V LDO through the *ESCR*[3] selection, an accurate and low-leakage external capacitor CS is required to be connect between CS pin and GND.
4. In the meantime, users should set PA5/PA7 as CS pin through the Code_OPTION CS_Sel. Please note, the CS1/TK17 function of PA5 cannot be used at the same time.

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

1. Selecting the touch pad to be measure by setting *TKE1/TKE2/TKE3* registers. Only one pad should be selected a time.
2. Issuing a Touch START command by writing “0x10” into TCC register. The capacitor CS will be automatically discharged to VSS firstly. The discharging time is selectable from 32, 64 and 128 Touch clocks by ***TS[1:0]***.
3. The larger the CS capacitance value, the longer the discharge time is needed to fully discharge the capacitor to VSS. However, in some cases, 128 Touch clocks may still be not long enough to fully discharge the CS capacitor. At this time, user should do it manually by writing “0x30” into TCC register instead of “0x10”. After a certain discharge time decided by the user, user can issue a Touch START (0x10) command to continue this touch conversion progress. Or user can also abort the conversion progress by writing “0x00” into TCC register.
4. After discharging, the CS will be charged toward VDD per Touch clock (TK_CLK). The charging speed is determined by the capacitance value of the selected Touch pad.
5. The charging progress will be stopped automatically when its voltage reaches the internal generated threshold voltage (VREF). The program determines whether the charging process is stopped by reading INTRQ[3].The VREF voltage is selectable from 0.2*TP ~ 0.8*TP by ***TS[4:2]***.
6. By reading the Touch Counter value from TKCH & TKCL registers, user can monitor the capacitance value change of the Touch pad. The value reads from Touch Counter is related to the ratio of CS and CP (Capacitive Touch Sensor Pin), while CP represents the total capacitance that is the combination of PCB, wire and touch pad whose capacitance can be varied by human finger’s touch. Once the CP value is altered, the periods required to charge the CS to VREF shorten. The user can judge whether the touch pad is pressed or released by reading the difference of the touch counter.
7. The user can change the sensitivity of the touch by adjusting the value of the CS capacitor. If a CS capacitor with an excessively large value is used, the touch counter value may overflow. At this time, the INTRQ2.TK_OV flag will be automatically set by the hardware, and The touch count value will continue to count from 0 again.

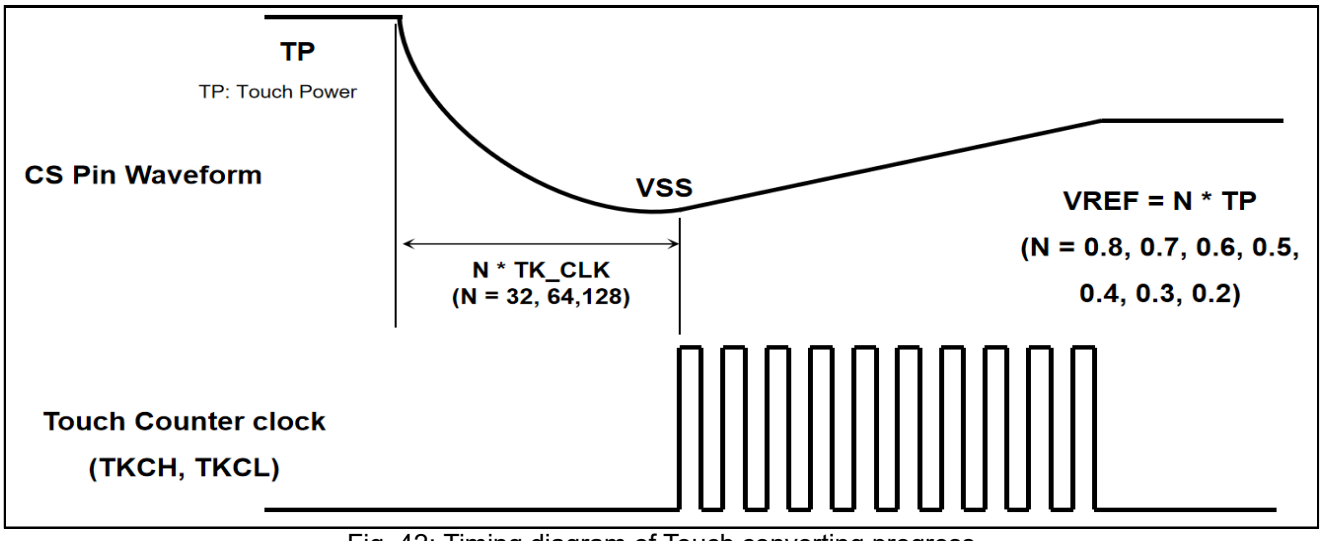


Fig. 42: Timing diagram of Touch converting progress

Note:

1. When the VREF voltage is first set or the reference voltage option is switched midway, please discard the first *TKCH* and *TKCL* data read after that.
2. The touch key channel and ADC channel cannot be enabled by the same IO pin currently. If enabled at the same time, the touch key count will decrease. The default ADC conversion channel is PB0/TK11. When PB0/TK11 is used, the default ADC channel must be set to other pins.
3. Under the same conditions, the touch key count value of different pins will be slightly different due to the capacitive effect of individual IO pins (drive current, package...and other factors). Touch key channel TK7/PA0 and TK17/PA5/CS1 will have touch key count values slightly smaller than the other pins.
4. In ByPass mode, the Touch START (write "0x10" into TCC register) command must be executed at a system frequency of 250KHz (IHRC/64). The LDO mode does not have this limitation.

11.6.1. Touch Selection Register (*TS*), address = 0x37

Bit	Reset	R/W	Description
7 – 5	000	R/W	Touch clock selection (TK_CLK) 000: ILRC 001: IHRC/2 010: IHRC/4 011: IHRC/8 100: IHRC/16 101: IHRC/32 110: IHRC/64 111: IHRC/128
4 – 2	011	R/W	Touch VREF selection (TP: Touch Power, the default is LDO 2V) 000: 0.8 * TP 001: 0.7 * TP 010: 0.6 * TP 011: 0.5 * TP 100: 0.4 * TP 101: 0.3 * TP 110: 0.2 * TP 111: reserved
1 – 0	00	R/W	Select the discharge time before starting the touch function (TK_DISCHG) 00: reserved 01: 32 * CLK 10: 64 * CLK 11: 128 * CLK

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

11.6.2. External Oscillator Setting Register (*EOSCR*), address = 0x0F

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	-	-	Reserved.
3	**-**	**WO**	**LDO output voltage option。0/1：2.4V/2V**
2	**-**	**WO**	**Touch module power option。0/1：VDD/LDO**
1 – 0	-	-	Reserved.

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

11.6.3. Touch Charge Control Register (*TCC*), address = 0x38

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

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

11.6.4. Touch Key Enable 3 Register (*TKE3*), address = 0x39

Bit	Reset	R/W	Description
7	0	R/W	Enable PC7/TK23. 0/1: disable/enable
6	0	R/W	Enable PC6/TK22. 0/1: disable/enable
5	0	R/W	Enable PC5/TK21. 0/1: disable/enable
4	0	R/W	Enable PC4/TK20. 0/1: disable/enable
3	0	R/W	Enable PC3/TK19. 0/1: disable/enable
2	0	R/W	Enable PC1/TK18. 0/1: disable/enable
1	0	R/W	Enable PA5/TK17. 0/1: disable/enable
0	0	R/W	Enable PC2/TK16. 0/1: disable/enable

Note: The touch key channel and ADC channel cannot be enabled by the same IO pin currently.

11.6.5. Touch Key Enable 2 Register (*TKE2*), address = 0x3A

Bit	Reset	R/W	Description
7	0	R/W	Enable PC0/TK15. 0/1: disable/enable
6	0	R/W	Enable PA2/TK14. 0/1: disable/enable
5	0	R/W	Enable PA1/TK13. 0/1: disable/enable
4	0	R/W	Enable PB1/TK12. 0/1: disable/enable
3	0	R/W	Enable PB0/TK11.0/1: disable/enable
2	0	R/W	Enable PB3/TK10. 0/1: disable/enable
1	0	R/W	Enable PB2/TK9. 0/1: disable/enable
0	0	R/W	Enable PA6/TK8. 0/1: disable/enable

Note: 1. The touch key channel and ADC channel cannot be enabled by the same IO pin currently.
2. When PB0/TK7 is used as the touch key channel, it is necessary to change the default ADC channel to another channel.

11.6.6. Touch Key Enable 1 Register (*TKE1*), address = 0x3B

Bit	Reset	R/W	Description
7	0	R/W	Enable PA0/TK7. 0/1: disable/enable
6	0	R/W	Enable PA4/TK6. 0/1: disable/enable
5	0	R/W	Enable PA3/TK5. 0/1: disable/enable
4	0	R/W	Enable PB7/TK4. 0/1: disable/enable
3	0	R/W	Enable PB6/TK3. 0/1: disable/enable
2	0	R/W	Enable PB5/TK2. 0/1: disable/enable
1	0	R/W	Enable PB4/TK1. 0/1: disable/enable
0	0	R/W	Enable PD0/TK0. 0/1: disable/enable

Note: The touch key channel and ADC channel cannot be enabled by the same IO pin currently.

11.6.7. Touch Key Charge Counter High Register (*TKCH*), address = 0x7A

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

11.6.8. Touch Key Charge Counter Low Register (*TKCL*), address = 0x7B

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

11.6.9. Touch parameter setting register (*TPS*), IO address = 0x3C

Bit	Reset	R/W	Description
7 – 0	0x00	R/W	Reserved, keep 0x00

Note：TPS = 0x00

11.6.10. Touch parameter setting register 2 (*TPS2*), IO address = 0x3D

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

Case:

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

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

11.7. Programmable Frequency Generator (PFG)

PFC460 provides a programmable frequency generator (PFG) for precise frequency output. It's hardware diagram is shown in Fig. 43.

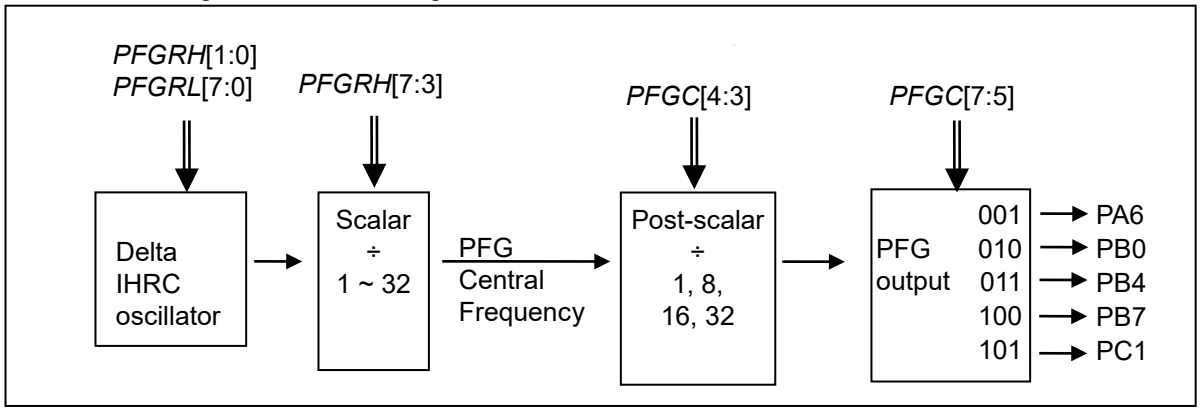


Fig. 43: Hardware Diagram of PFG

The PFG oscillator circuit is only provided by the Delta IHRC oscillator and is independent of the IHRC. The Delta IHRC frequency can be trimming to 36MHz by setting the registers *PFGRH[1:0]* and *PFGRL*. And there are 1024 trimming steps separated to ±512 steps, and about 0.12% per step. The system default frequency of Delta IHRC is 36MHz (*PFGRH*[1:0]=0x2, *PFGRL=0*).
The Delta IHRC oscillator can control scalar to generate the PFG central frequency through the register *PFGRH*[7:3], and then the PFG central frequency can be post-scalar to support more system requirements.

The Delta IHRC oscillator may be a non-precision 36MHz oscillator. It will be affected by the manufacturing process, operating voltage, operating temperature... Due to the influence of other factors, the oscillation frequency is drifted around 36MHz. It is recommended that the Delta IHRC oscillator must be calibrated for frequency before using PFG, and the frequency correction can use the internal IHRC or system runtime as a standard clock calibration source. After the correction, refill the appropriate 10-Bits correction value (PFGRH [1:0] + PFGRL [7:0]).

11.7.1. PFG Central Frequency

Set the required PFG central frequency by the register *PFGRH*[7:3]. Frequently-used central frequencies are shown in the following table, which are 3.3MHz/3MHz/2.4MHz/1.7MHz. For other frequencies, users can still configure them according to their needs.

Delta_IHRC (MHz)	*PFGRH*[7:3] (Decimal)	Result (MHz)
36	11	3.27
36	12	3.00
36	15	2.40
36	21	1.71

Table 14: PFG Central Frequency Set Parameter

PFG center frequency calculation formula:

PFG frequency=Y ÷ S1

Y = PFGRH[1:0] + PFGRL[7:0] : Delta IHRC oscillator frequency

S1 = PFGRH[7:3] : Pre-partition register set value (decimal)

11.7.2. PFG Output Pins

The central frequency can be divided by 1,8,16,32 through setting the post-scalar register *PFGC*[4:3]. And after setting the register *PFGC*[7:5], the system will automatically change the PFG signal output by the corresponding port.

The Delta IHRC oscillator is enabled after setting the PFG output channel. When *PFGC*[7:5]=000, the PFG module is disable.

PFG output frequency = Y ÷ S1 ÷ S2

Y = PFGRH[1:0]+PFGRL[7:0] : delta IHRC oscillator frequency

S1 = PFGRH[7:3] : pre-partition register set value (decimal)

S2 = PFGC[4:3] : post-division frequencies, which are divided by /1, /8, /16, /32

11.7.3. PFG Related Registers

11.7.3.1. Delta IHRC Trimming High Register (*PFGRH*), address = 0x30

Bit	Reset	R/W	Description
7 – 3	0x0b	R/W	Scalar Register. Central frequency = Delta IHRC / *PFGRH*[7:3]
2	-	-	Reserved
1 – 0	0x2	R/W	Delta IHRC trimming high register

11.7.3.2. Delta IHRC Trimming Low Register (*PFGRL*), address = 0x31

Bit	Reset	R/W	Description
7 – 0	0x00	R/W	Delta IHRC trimming low register

Note: the configuration of *PFGRL* should be written after *PFGRH*.

11.7.3.3. PFG Control Register(*PFGC*), address = 0x32

Bit	Reset	R/W	Description
7 – 5	000	R/W	PFG output pin selection 000: PFG Disable（Delta IHRC oscillator Disable） 001: PA6 010: PB0 011: PB4 100: PB7 101:: PC1 Others: reserved
4 – 3	00	WO	PFG post-scalar 00：÷ 1 01：÷ 8 10：÷ 16 11：÷ 32

12. Notes for Emulation

It is recommended to use 6S-M-001 for emulation of PFC460. Please notice the following items while simulating:

(1) 6S-M-001 can support the single-core (1 FPPA) simulation debugging mode and Single/multi-core operation mode at full speed.

(2) 6S-M-001 will occupy part of the memory space: ROM is occupied from 0xD00, RAM is occupied from 0x1F8 ~ 0x1FF.

(3) 6S-M-001 will occupy PD0/PD1.

(4) For other emulation matters needing attention, please refer to the 6S-M-001user manual on the PADAUK official website.

13. Program Writing

There are 5 pins for using the writer to program: PA3, PA5, PA6, VDD and GND.

Please use 5S-P-003x to program. 3S-P-002 or older versions do not support programming PFC460.

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

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 PFC460. 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 5.8V.

(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. On-Board Writing Mode

PFC460 can support On-board writing. On-Board Writing is known as the situation that the IC has to be programmed when the IC itself and other peripheral circuits and devices have already been mounted on the PCB. Five wires of 5S-P-003x are used for On-Board Writing: ICPCK(PA3), ICPDA(PA6), VDD, GND and ICVPP(PA5). They are used to connect PA3, PA6, VDD, GND and PA5 of the IC correspondingly.

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

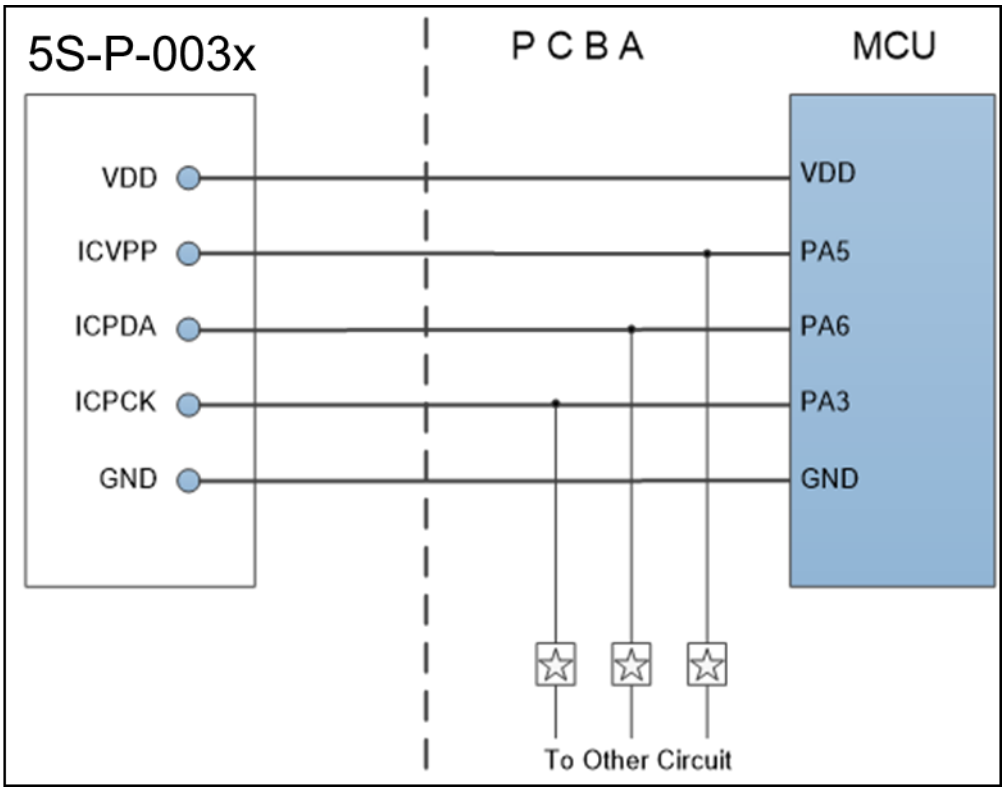


Fig. 44: Schematic Diagram of On-Board Writing

The above figure shows the connection for PFC460 on-board writing. In this figure, ☆can be either resistors or capacitors. They are used to isolate the programming signal wires from the peripheral circuit. it should be≧ 10KΩ for resistance while ≦220pF for capacitance.

Notice:

- The electrical characteristics of the On-board Writing are the same as the Normal Programming Mode, and the maximum of VDD is up to 7.5V. Please refers to the Normal Programming Mode for more details.
- Any zener diode ≦7.5V, or any circuitry which clamp the 7.5V to be created SHOULD NOT be connected between VDD and GND of the PCB.
- Any capacitor ≧500uF SHOULD NOT be connected between VDD and GND of the PCB.
- In general, the writing signal pins PA3, PA5 and PA6 SHOULD NOT be considered as strong output pins.

14. Device Characteristics

14.1. Absolute Maximum Ratings

Name	Min	Typ.	Max	Unit	Notes
Supply Voltage (VDD)	2.2# / 2.1		5.5	V	**Exceed the maximum rating may cause permanent damage !!**
Input Voltage	-0.3		V_{DD} + 0.3	V	
Operating Temperature	-40		105	°C	
Storage Temperature	-50		125	°C	
Junction Temperature		150		°C	

#The first batch of wafer characteristic parameters

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# / 2.1		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	 59K	 8M 4M 2M	Hz	 V_{DD}≧4.5V#; V_{DD}≧4.0V V_{DD}≧2.7V#; V_{DD}≧2.4V V_{DD}≧2.2V#; V_{DD}≧2.1V V_{DD} = 5V
P_{cycle}	Program cycle	1000			cycles	
f_{pwm}	The frequency of the clock source to PWMG0/1/2			32	MHz	32MHz is available only when VDD≧3.0V
I_{OP}	Operating Current		0.89 96		mA uA	f_{SYS}=IHRC/16=1MIPS@5V f_{SYS}=ILRC=59KHz@5V
I_{PD}	Power Down Current (by *stopsys* command)		0.8 1.2		uA uA	f_{SYS}= 0Hz,V_{DD}=5V f_{SYS}= 0Hz,V_{DD}=5V, NILRC Enalbe
I_{PS}	Power Save Current (by *stopexe* command)		3.5 4.0		uA uA	V_{DD} =5V; f_{SYS}= ILRC V_{DD} =5V; f_{SYS}= ILRC, NILRC Enable Only ILRC module is enabled.
V_{IL}	Input low voltage for IO lines	0		0.1V_{DD}	V	
V_{IH}	Input high voltage for IO lines	0.7 V_{DD}		V_{DD}	V	
I_{OH}	IO lines drive current PB2~PB7 (Strong) PB2~PB7 (Normal) Others IO		 -28 -10 -10		mA	V_{DD}=5V, V_{OH}=4.5V

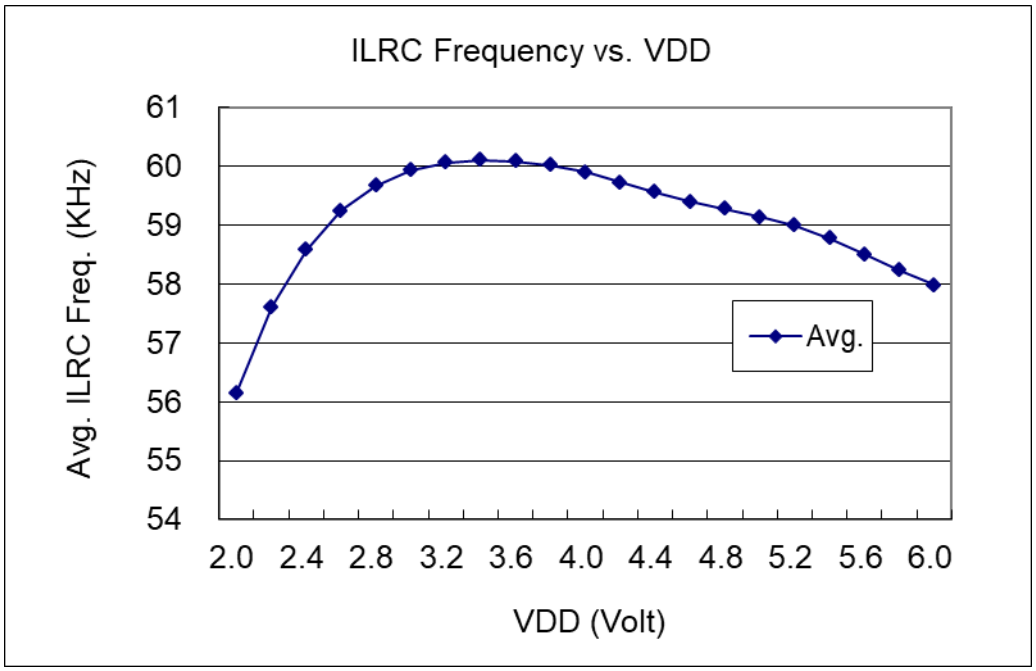
Symbol	Description	Min	Typ	Max	Unit	Conditions (Ta=25°C)
I_{OL}	IO lines sink current PA0~PA4 PB0 (Strong) PB0 (Normal) PB2~PB7 (Strong) PB2~PB7 (Normal) Others IO		 20 102 20 78 20 14		mA	V_{DD}=5V, V_{OL}=0.5V
V_{IN}	Input voltage	-0.3		V_{DD}+0.3	V	
$I_{INJ\ (PIN)}$	Injected current on pin			1	mA	V_{DD}+0.3≧V_{IN}≧ -0.3
R_{PH}	Pull-high Resistance		80		KΩ	V_{DD}=5.0V
R_{PL}	Pull-low Resistance		81		KΩ	V_{DD}=5.0V
V_{BG}	Bandgap Reference Voltage	1.145*	1.20*	1.255*	V	V_{DD} =3.9V ~ 5.5V -40°C < Ta < 105°C*
f_{IHRC}	Frequency of IHRC after calibration *	15.84*	16*	16.16*	MHz	V_{DD} =5V, Ta=25°C
		15.20*	16*	16.80*		V_{DD} =3.9V~5.5V, -40°C <Ta<105°C*
f_{DIHRC}	Frequency of Delta IHRC after calibration *		36		MHz	
f_{ILRC}	Frequency of ILRC *		59		KHz	V_{DD} = 5.0V
f_{NILRC}	Frequency of NILRC *		15		KHz	V_{DD} = 5.0V
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		0.93 0.83		mA	@5V @3V
ADclk	ADC clock period		2		us	3.9V ~ 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 4V 3V 2V	 3.90 2.93 1.95	 4 3 2	 4.10 3.07 2.05		V_{DD} =5V, 25 °C
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

Symbol	Description	Min	Typ	Max	Unit	Conditions (Ta=25°C)
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		50		ms	V_{DD}=5V
t_{RST}	External reset pulse width	120			us	V_{DD}=5V
CPos	Comparator offset*	-	±10	±20	mV	
CPcm	Comparator input common mode*	0		V_{DD} -1.5	V	
CPspt	Comparator response time**		100	500	ns	Both Rising and Falling
CPmc	Stable time to change comparator mode		2.5	7.5	us	
CPcs	Comparator current consumption		25		uA	V_{DD} = 5V
OPAcm	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	

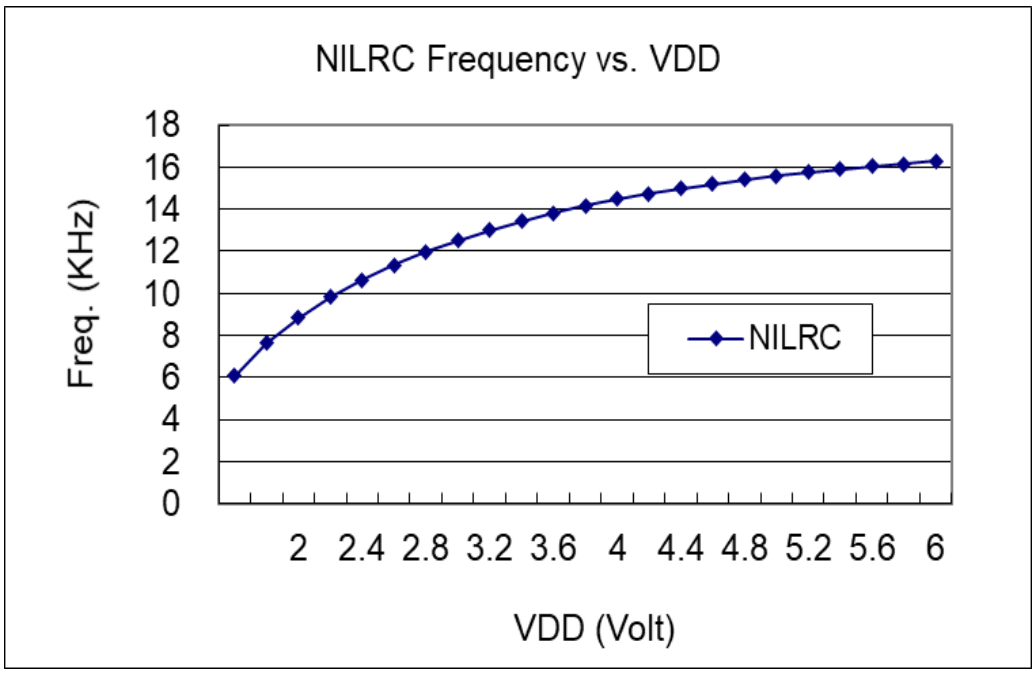
#The first batch of wafer characteristic parameters
*These parameters are for design reference, not tested for every chip.

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.

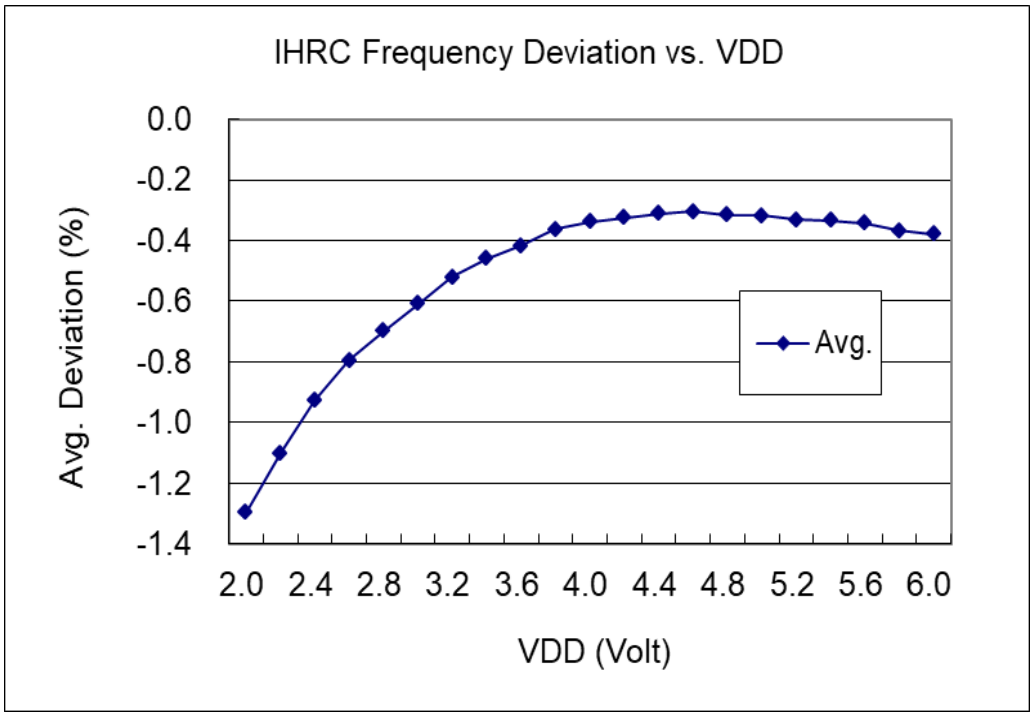
14.3. Typical ILRC frequency vs. VDD



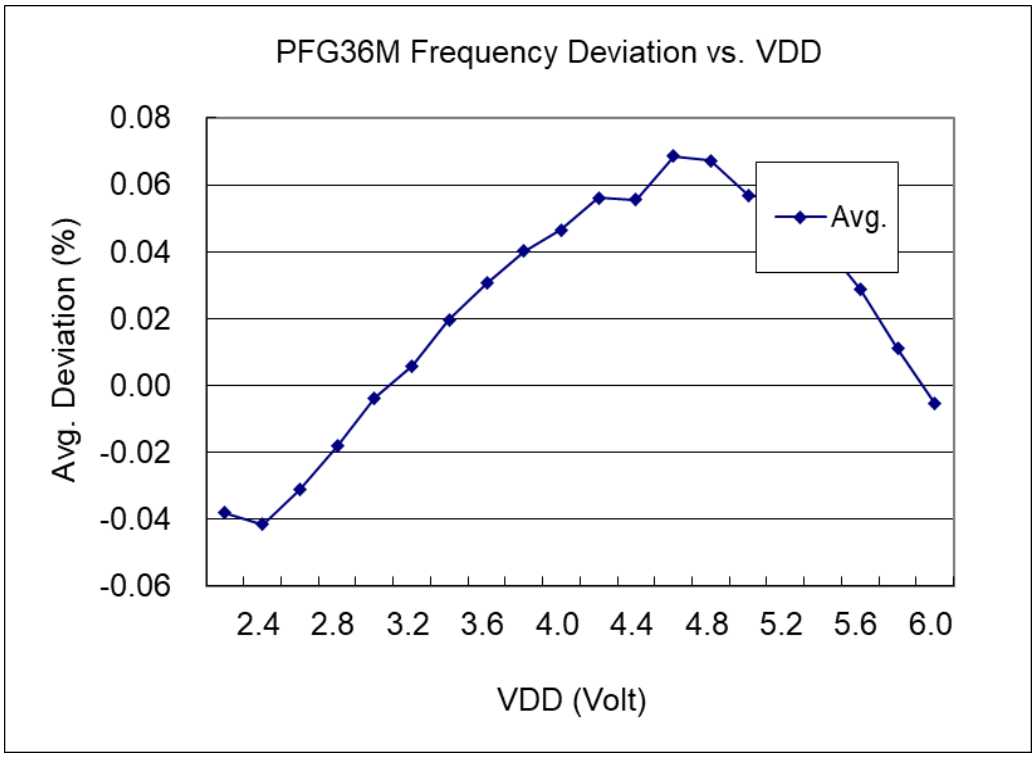
14.4. Typical NILRC frequency vs. VDD



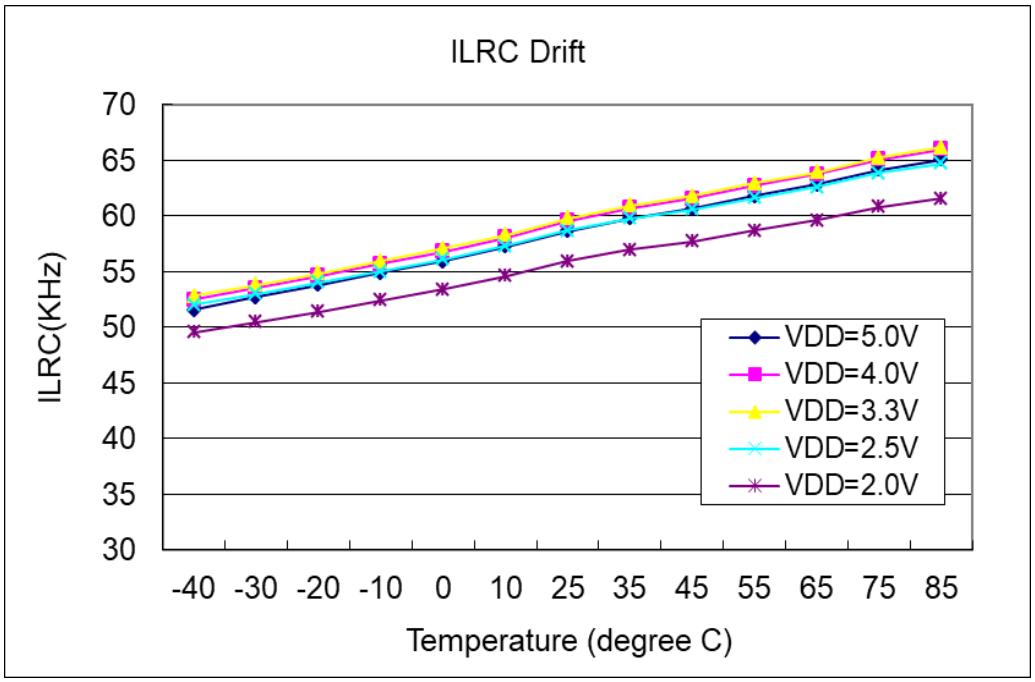
14.5. Typical IHRC frequency deviation vs. VDD(calibrated to 16MHz)



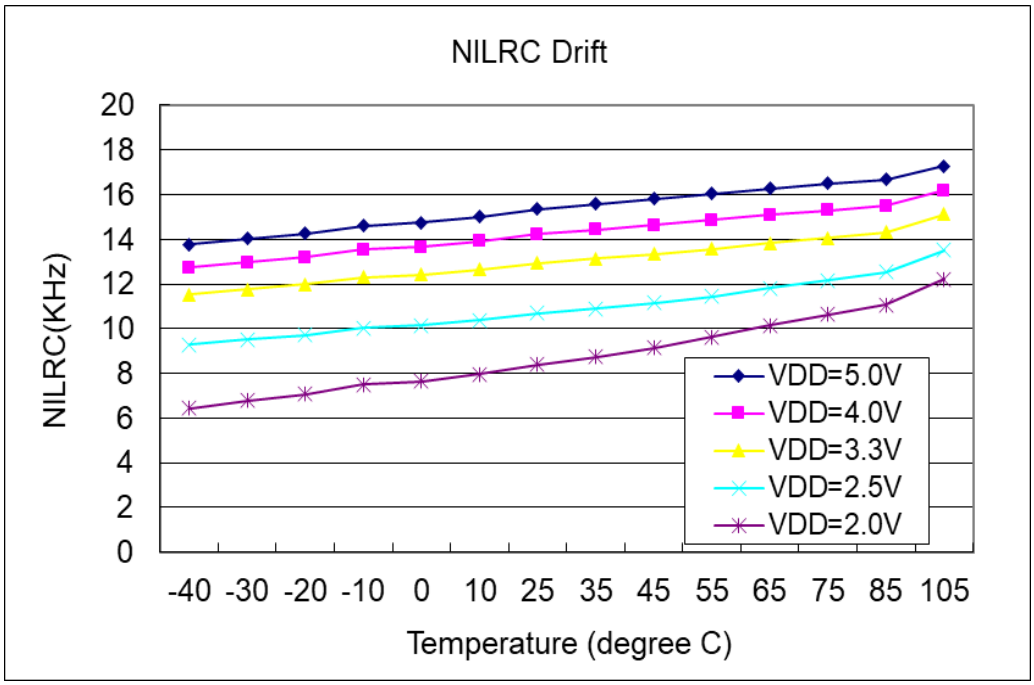
14.6. PFG frequency deviation vs. VDD (calibrated to 36MHz)



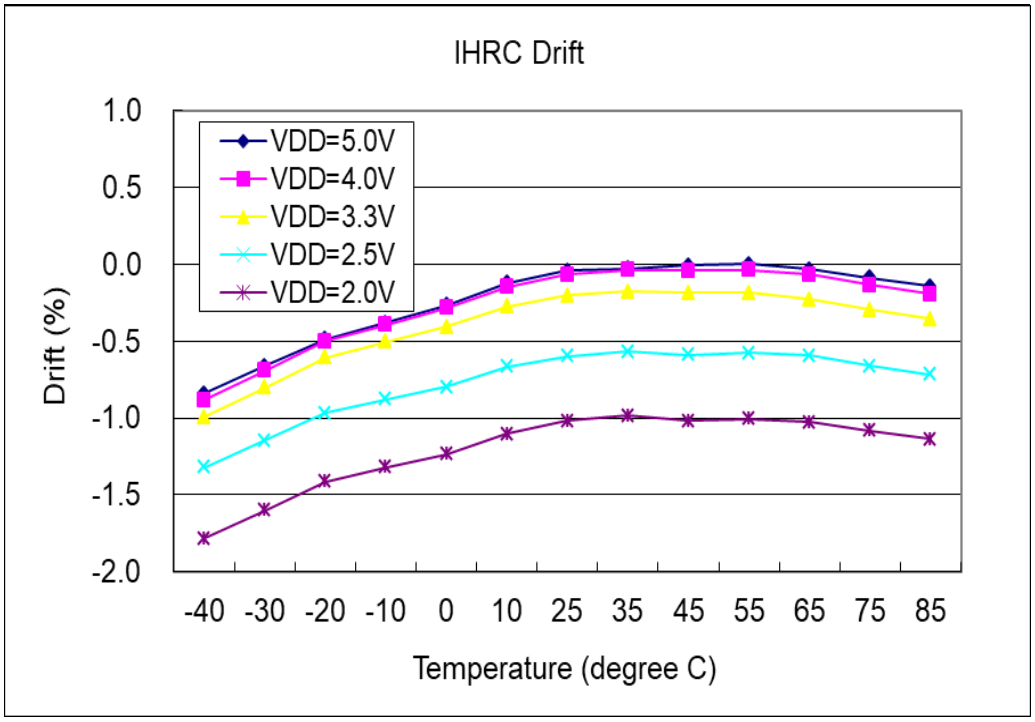
14.7. Typical ILRC frequency vs. Temperature



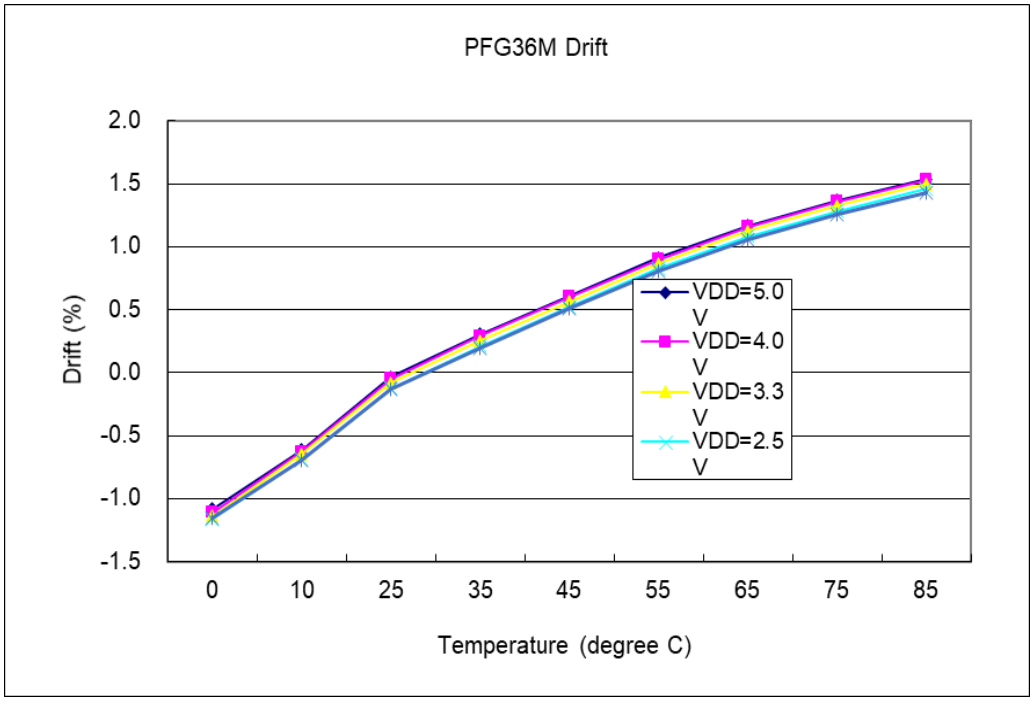
14.8. Typical NILRC frequency vs. Temperature



14.9. Typical IHRC frequency vs. Temperature (calibrated to 16MHz)



14.10. PFG frequency deviation vs. Temperature (calibrated to 36MHz)

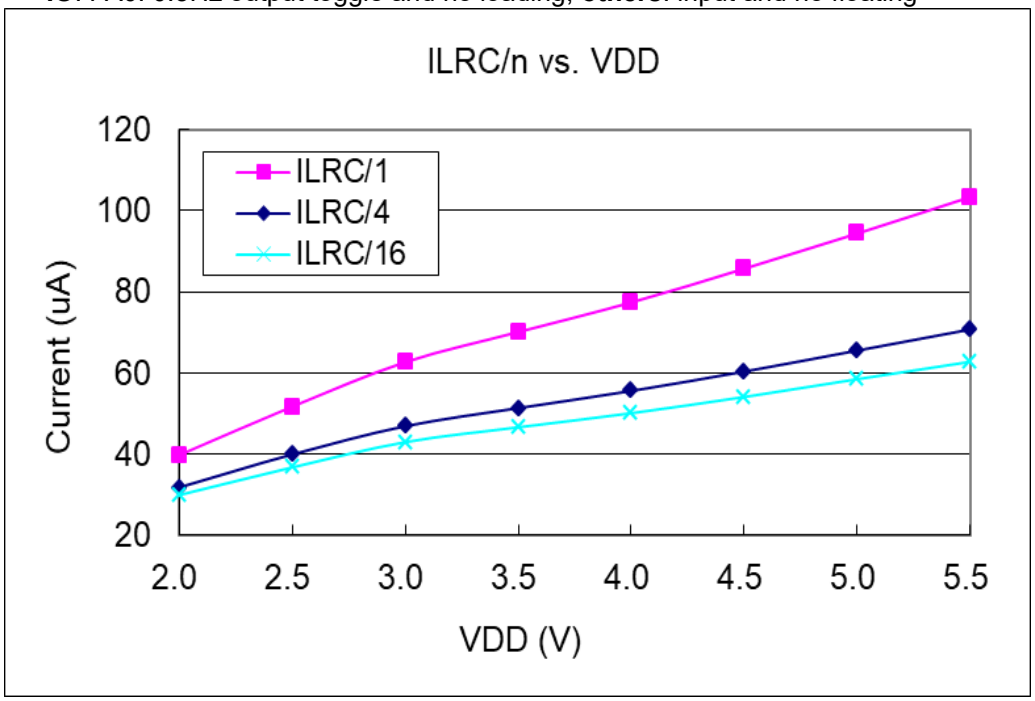


14.11. Typical operating current vs. VDD @ system clock = ILRC/n

Conditions: 1-FPPA (FPPA0: tog PA0)

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

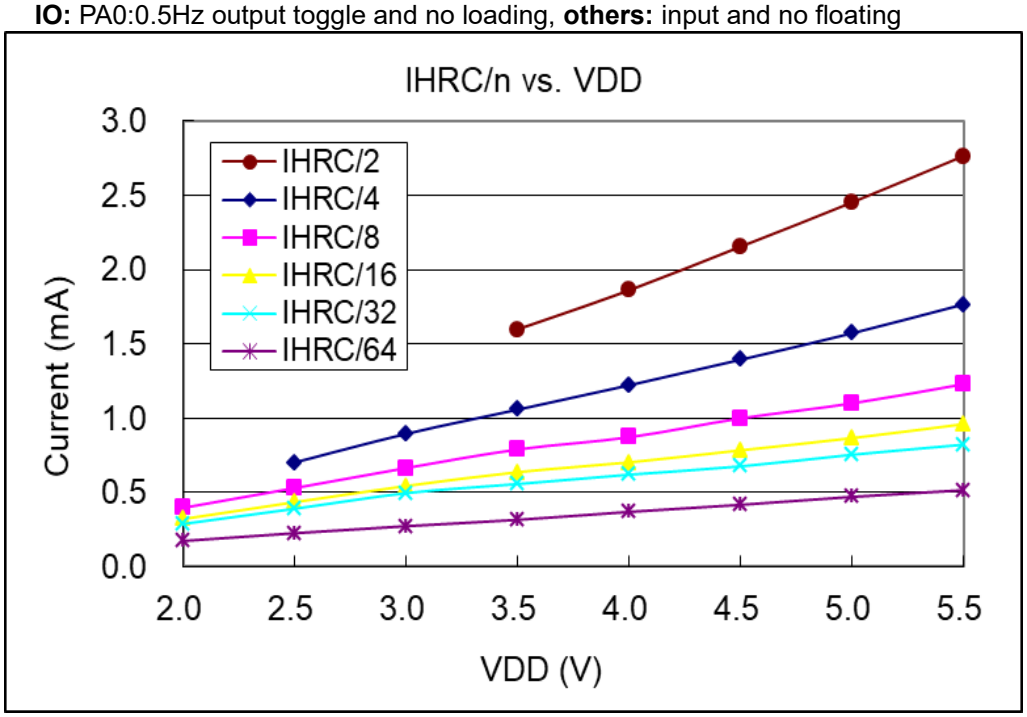


14.12. Typical operating current vs. VDD @ system clock = IHRC/n

Conditions: 1-FPPA (FPPA0: tog PA0)

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

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

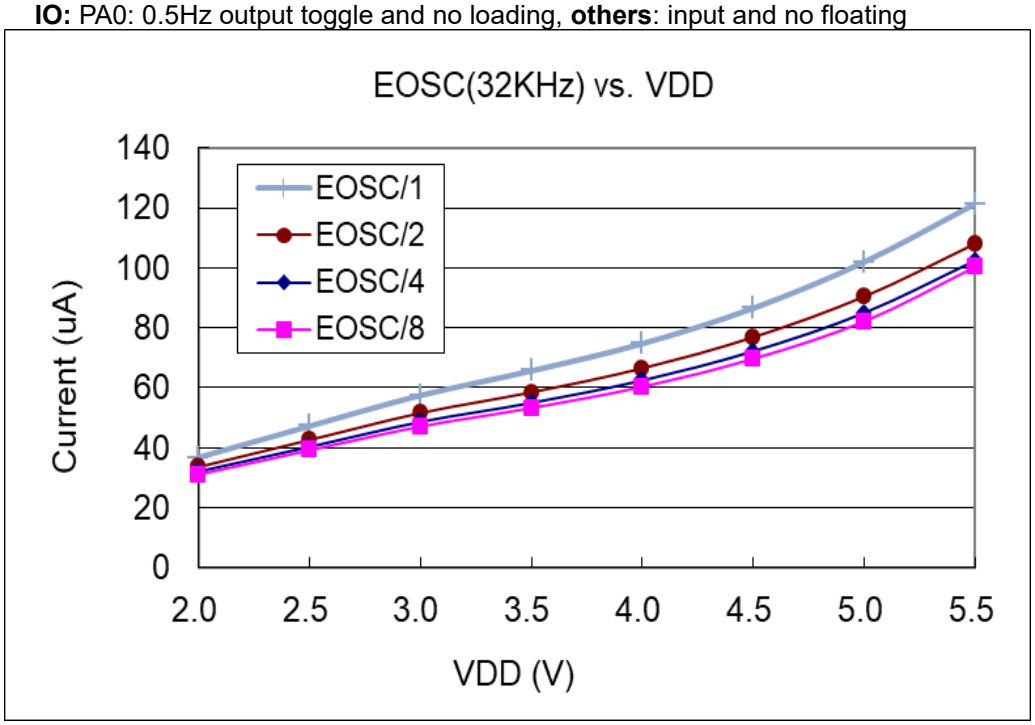


14.13. Typical operating current vs. VDD @ system clock = 32KHz EOSC / n

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

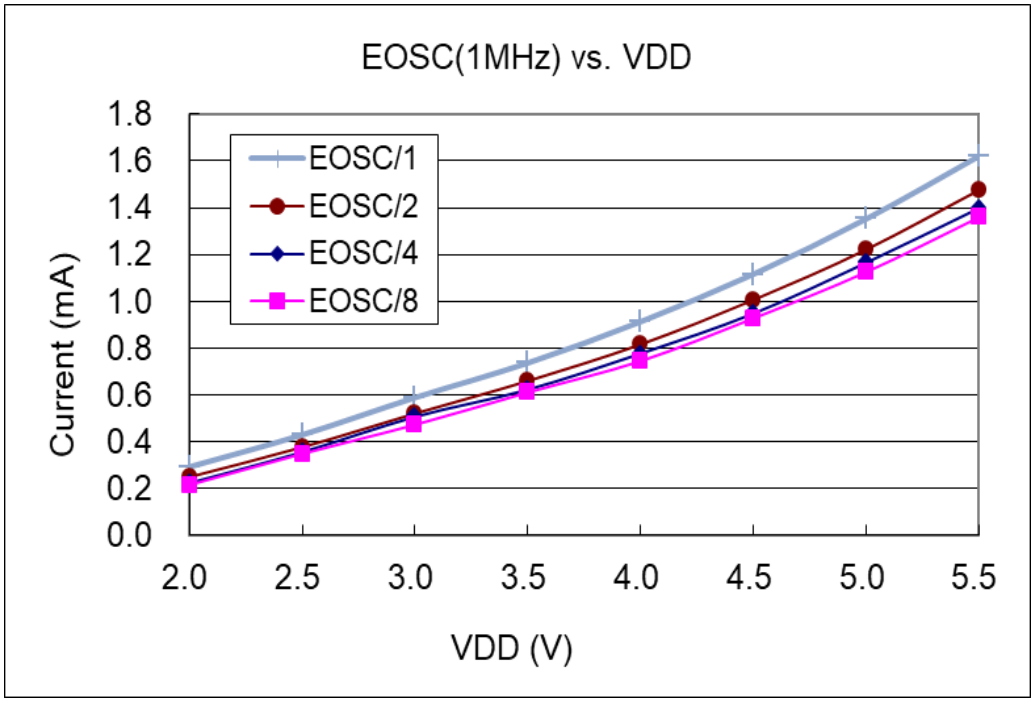


14.14. Typical operating current vs. VDD @ system clock = 1MHz EOSC / n

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

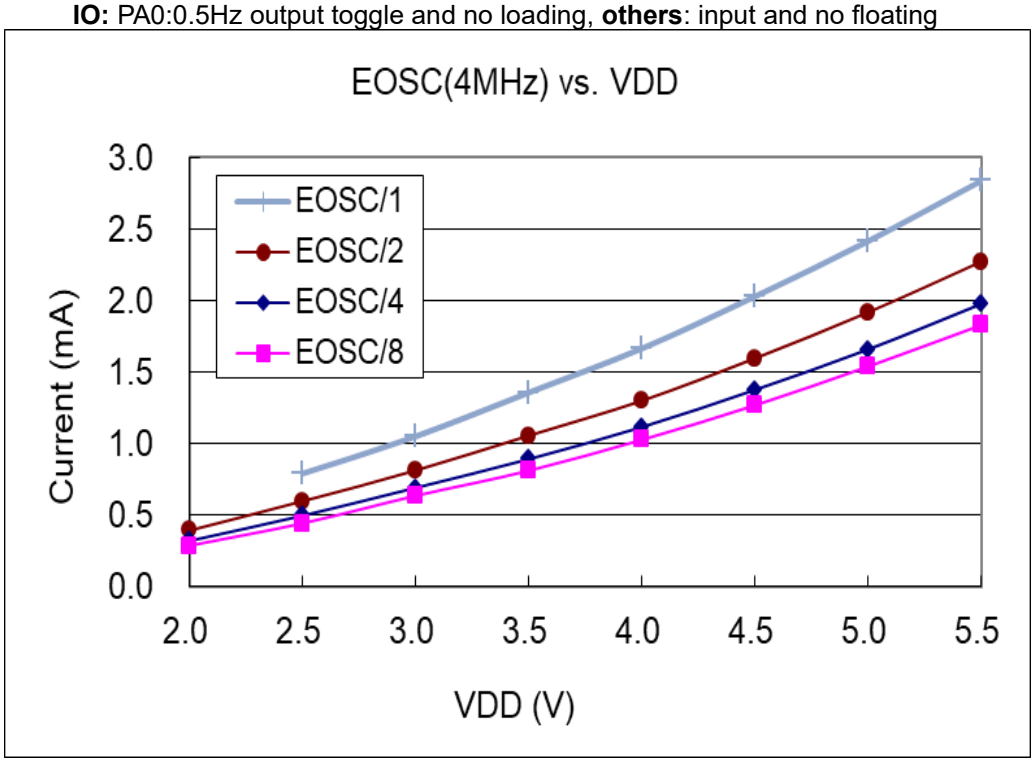


14.15. Typical operating current vs. VDD @ system clock = 4MHz EOSC / n

Conditions: **ON**: EOSC, MISC.6 = 1, Bandgap, LVR;

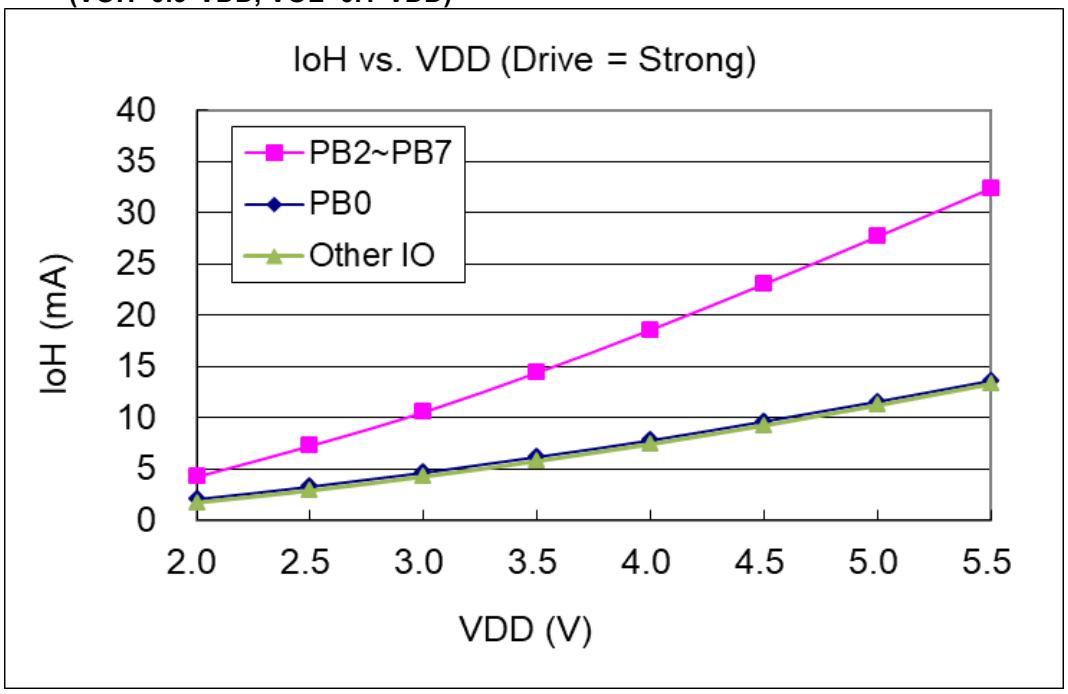
OFF: IHRC, ILRC, T16, TM2, TM3, ADC modules;

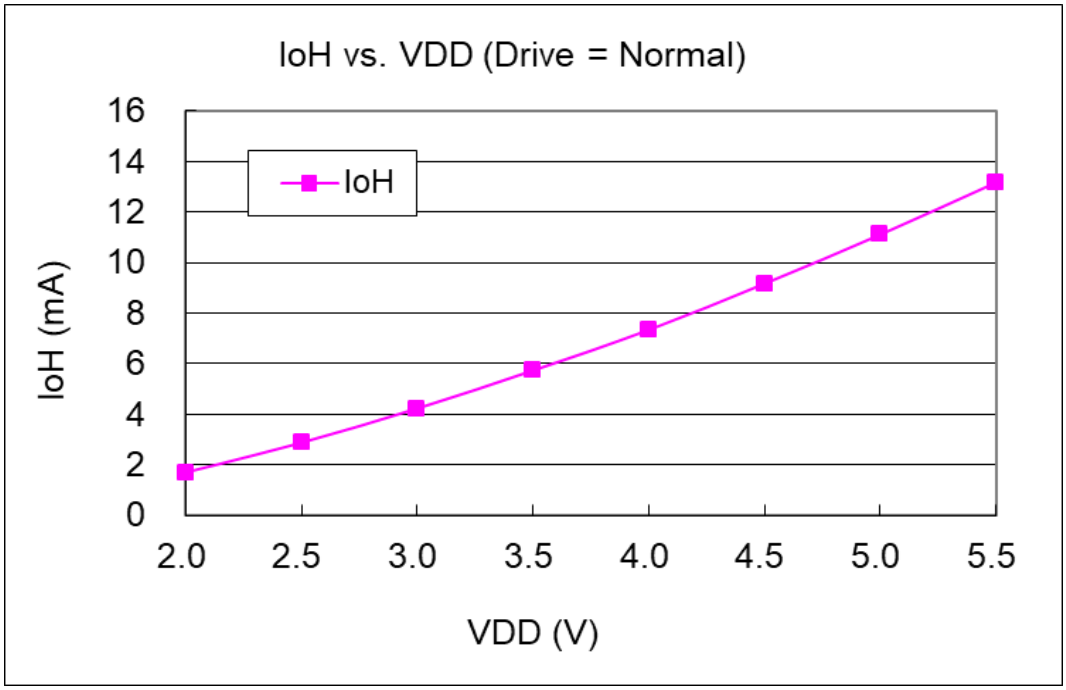
IO: PA0:0.5Hz output toggle and no loading, **others**: input and no floating

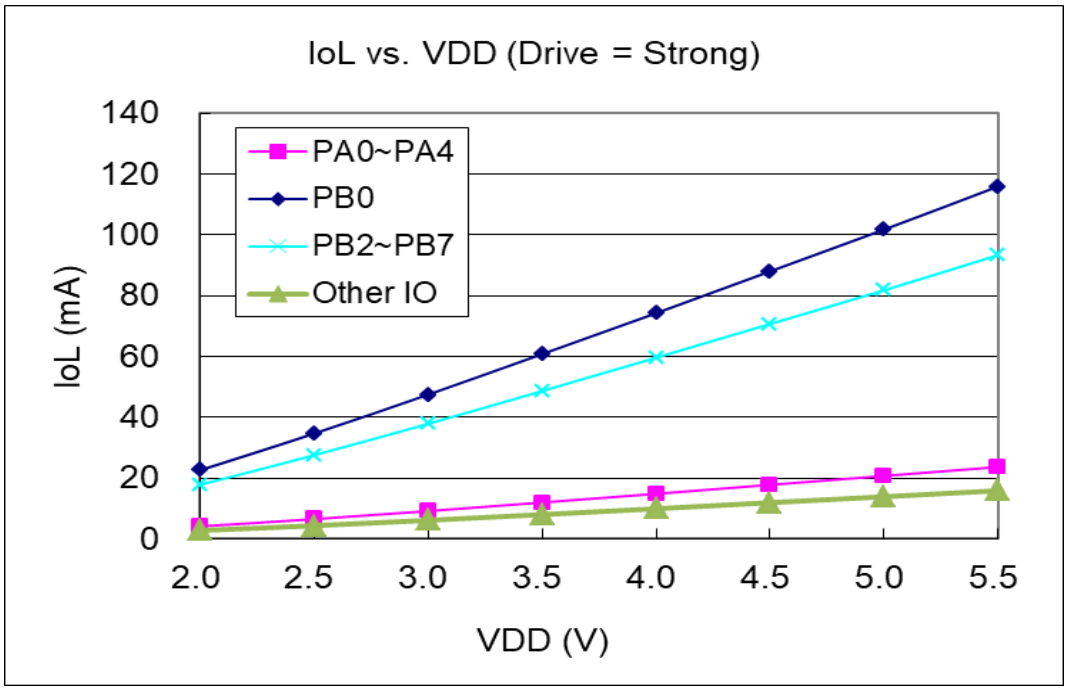


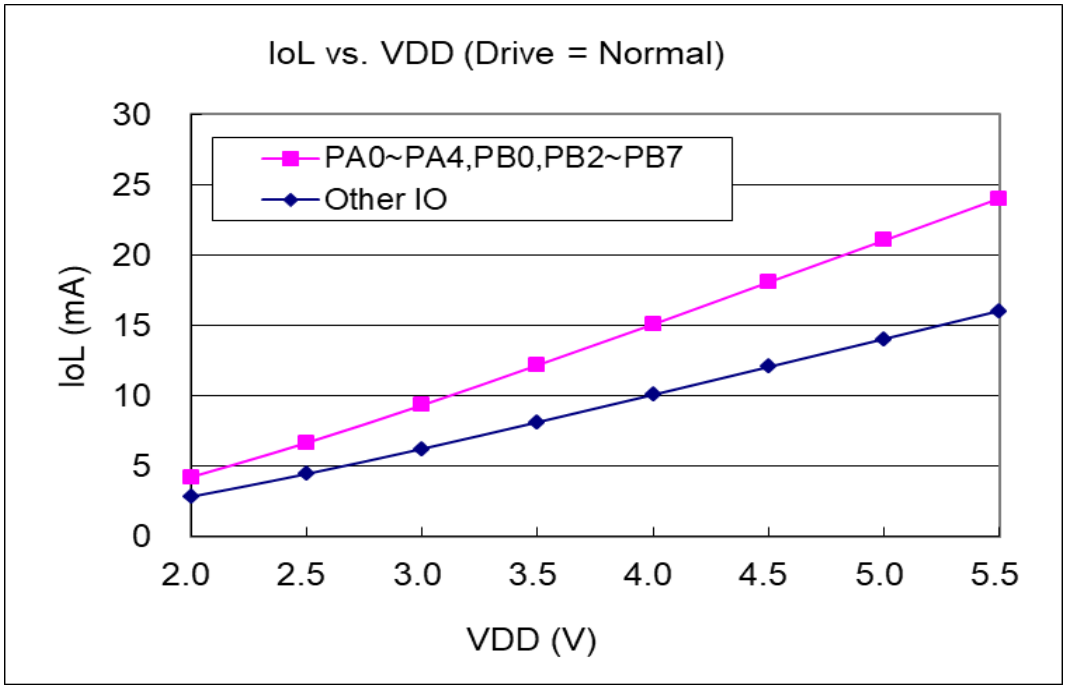
14.16. Typical IO driving current (I_{OH}) and sink current (I_{OL})

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

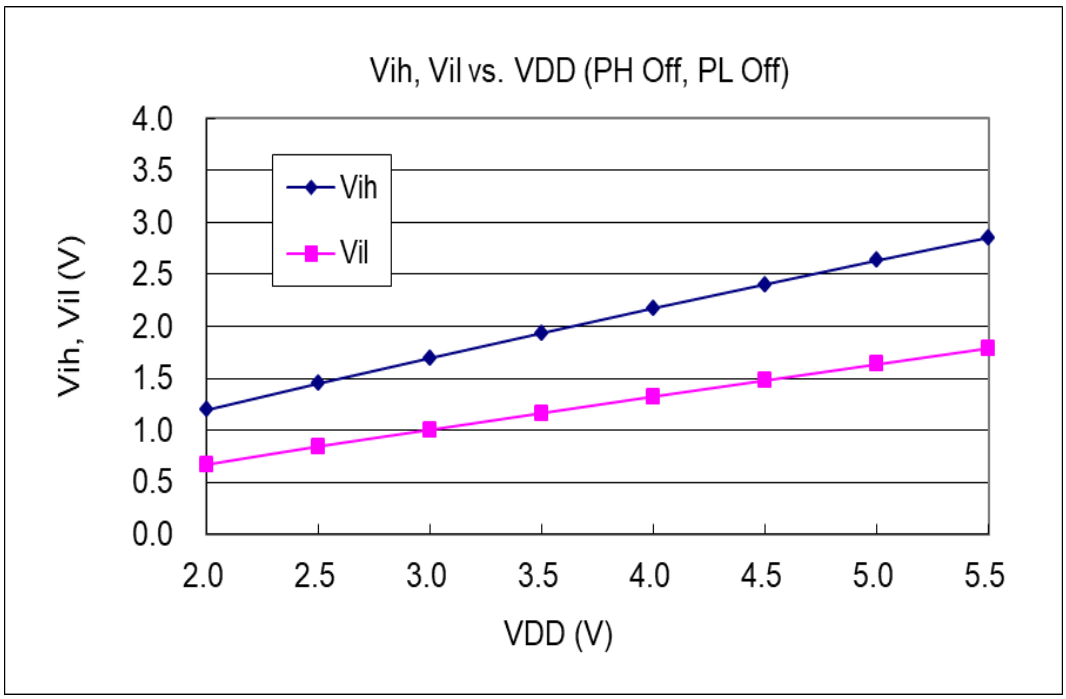



IoH vs. VDD (Drive = Normal)
IoH (mA)
16
14
12
10
8
6
4
2
0
IoH
2.0
2.5
3.0
3.5
4.0
4.5
5.0
5.5
VDD (V)

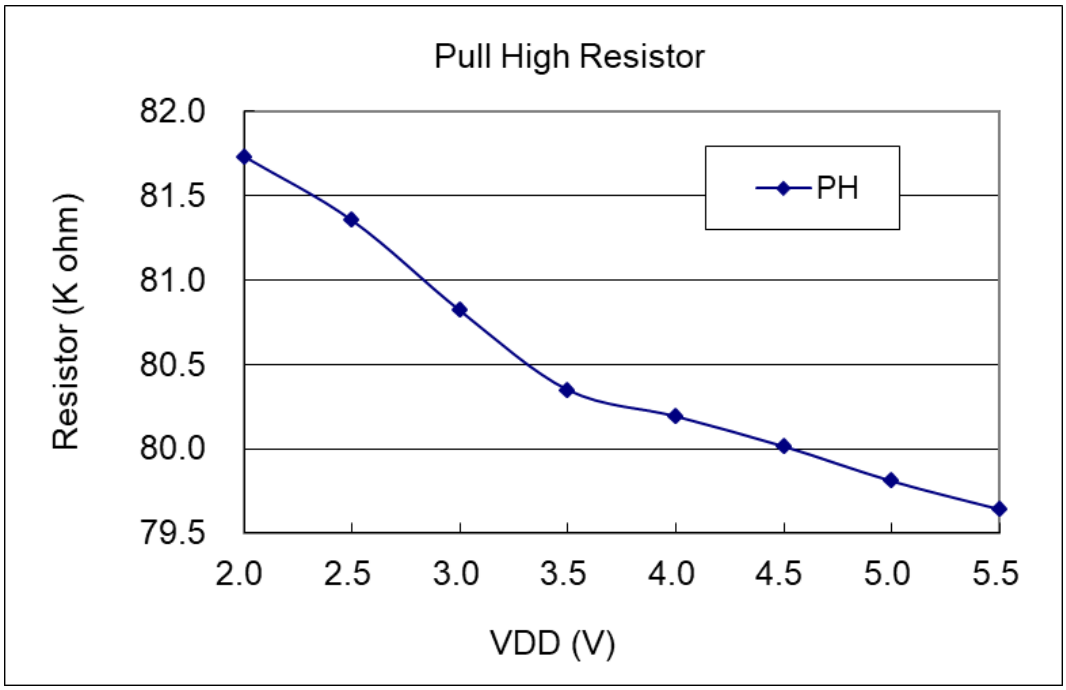

IoL vs. VDD (Drive = Strong)
IoL (mA)
140
120
100
80
60
40
20
0
PA0~PA4
PB0
PB2~PB7
Other IO
2.0
2.5
3.0
3.5
4.0
4.5
5.0
5.5
VDD (V)

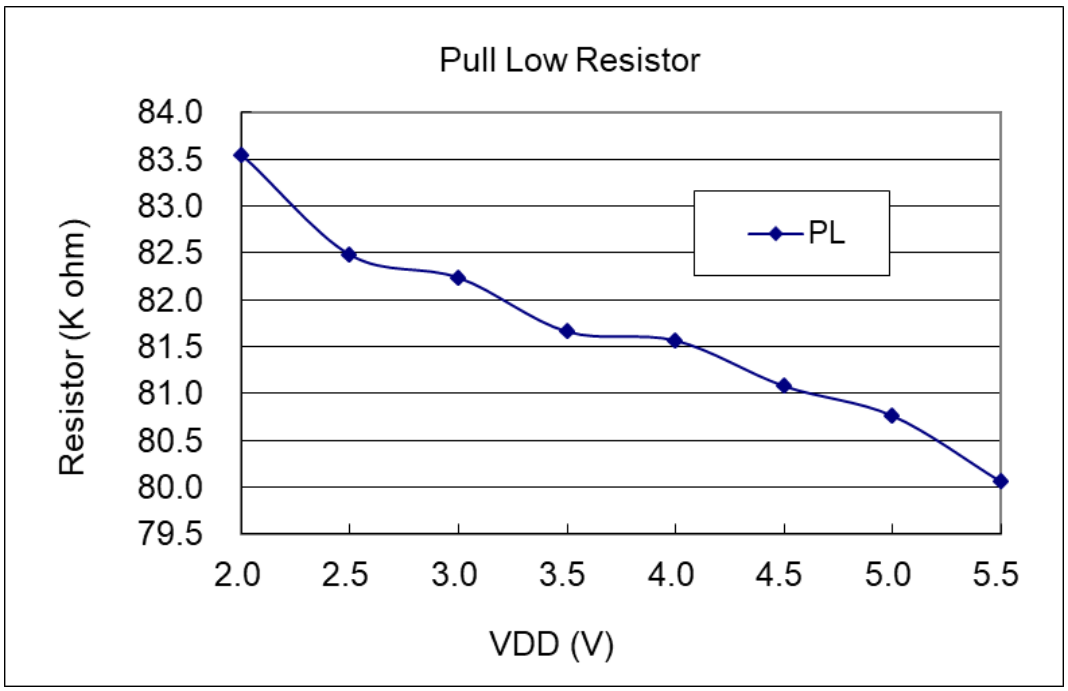


14.17. Typical IO input high/low threshold voltage (V_{IH}/V_{IL})

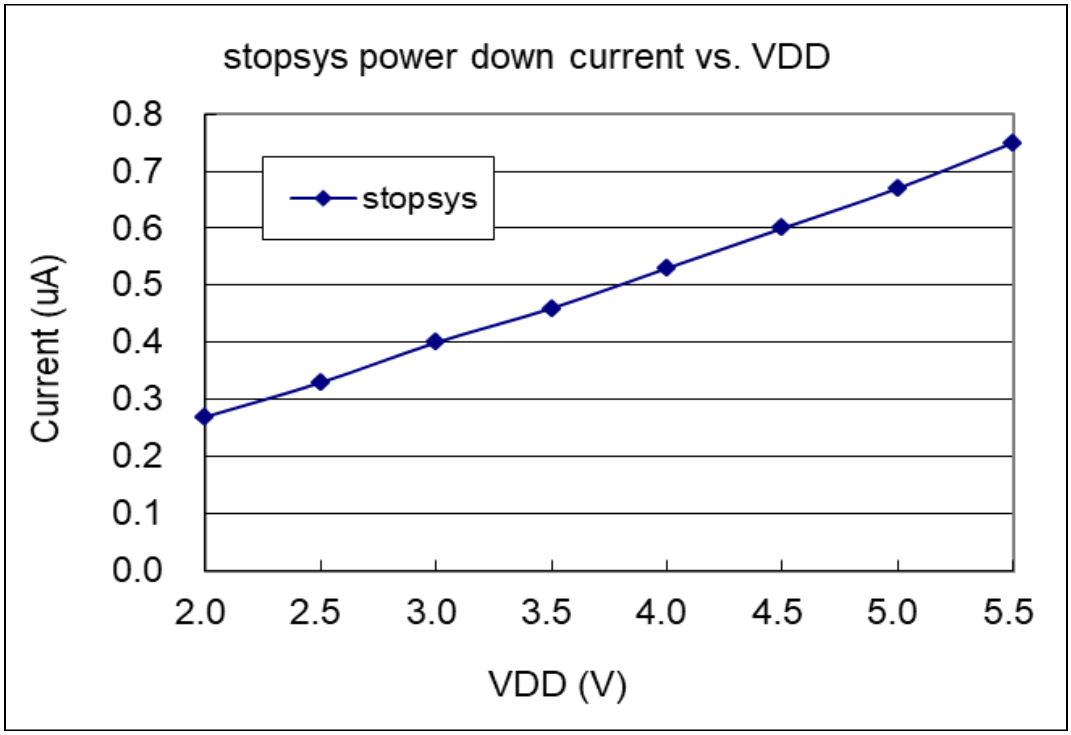


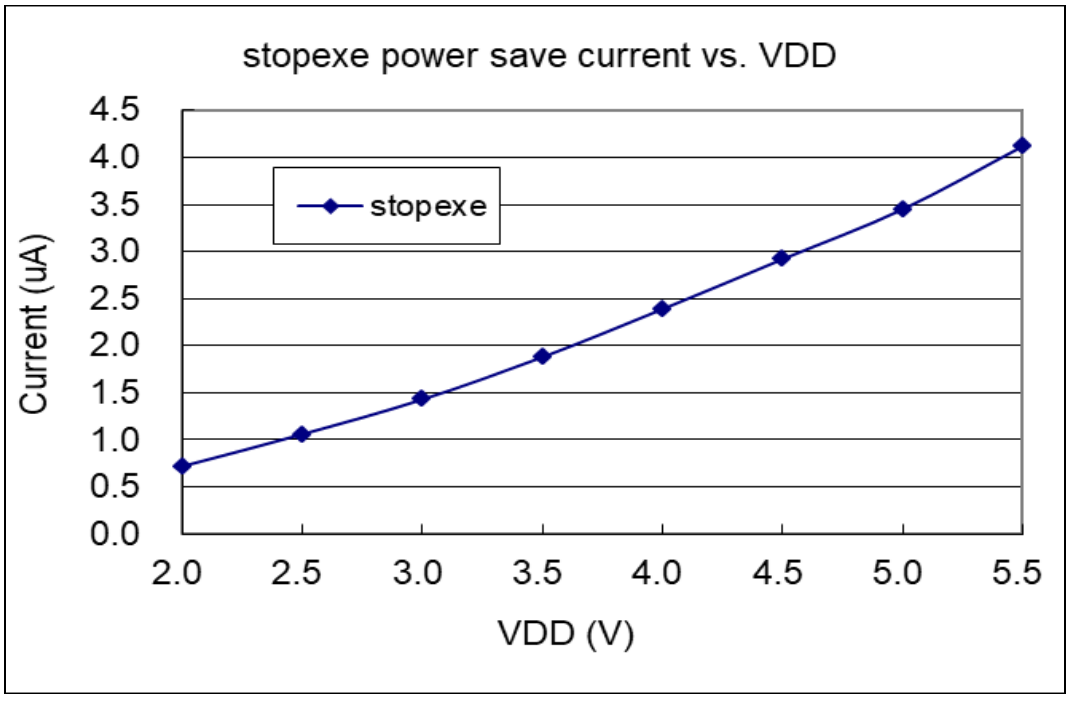
14.18. Typical resistance of IO pull high/low device





14.19. 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 *ALU* operation is zero, this bit is set to 1)
C	Carry (The operational result is to have carry out for addition or to borrow carry for subtraction in unsigned number system)
AC	Auxiliary Carry (If there is a carry out from low nibble after the result of *ALU* operation, this bit is set to 1)
IO.n	The bit of register, only addressed in 0~0x3F (0~63) is allowed
M.n	It can address in the full space of SRAM.
pc0	Program counter of FPPA0
pc1	Program counter of FPPA1
pc2	Program counter of FPPA2
pc3	Program counter of FPPA3

14.20. Data Transfer Instructions

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

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

---

ldtabl index

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

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

---

ldt16 word

Move 16-bit counting values in Timer16 to memory in word.
Example: ldt16 word;
Result: word ← 16-bit timer
Affected flags: 『N』Z 『N』C 『N』AC 『N』OV
Application Example:

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

---

stt16 word

Store 16-bit data from memory in word to Timer16.
Example: stt16 word;
Result: 16-bit timer ←word
Affected flags: 『N』Z 『N』C 『N』AC 『N』OV

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

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

14.21. Arithmetic Operation Instructions

add a, I	Add immediate data with ACC, then put result into ACC Example: *add a, 0x0f ;* Result: a ← a + 0fh Affected flags: 『Y』Z 『Y』C 『Y』AC 『Y』OV
add a, M	Add data in memory with ACC, then put result into ACC Example: *add a, MEM ;* Result: a ← a + MEM Affected flags: 『Y』Z 『Y』C 『Y』AC 『Y』OV
add M, a	Add data in memory with ACC, then put result into memory Example: *add MEM, a;* Result: MEM ← a + MEM Affected flags: 『Y』Z 『Y』C 『Y』AC 『Y』OV
addc a, M	Add data in memory with ACC and carry bit, then put result into ACC Example: addc a, MEM ; Result: a ← a + MEM + C Affected flags: 『Y』Z 『Y』C 『Y』AC 『Y』OV
addc M, a	Add data in memory with ACC and carry bit, then put result into memory Example: addc MEM, a ; Result: MEM ← a + MEM + C Affected flags: 『Y』Z 『Y』C 『Y』AC 『Y』OV
addc a	Add carry with ACC, then put result into ACC Example: addc a ; Result: a ← a + C Affected flags: 『Y』Z 『Y』C 『Y』AC 『Y』OV
addc M	Add carry with memory, then put result into memory Example: addc MEM ; Result: MEM ← MEM + C Affected flags: 『Y』Z 『Y』C 『Y』AC 『Y』OV
nadd a, M	Add negative logic (2's complement) of ACC with memory Example: nadd a, MEM ; Result: a ← ∓a + MEM Affected flags: 『Y』Z 『Y』C 『Y』AC 『Y』OV

nadd M, a	Add negative logic (2's complement) of memory with ACC Example: nadd MEM, a ; Result: MEM ← ∓MEM + a Affected flags: 『Y』Z 『Y』C 『Y』AC 『Y』OV
sub a, I	Subtraction immediate data from ACC, then put result into ACC. Example: sub a, 0x0f; Result: a ← a - 0fh (a + [2's complement of 0fh]) Affected flags: 『Y』Z 『Y』C 『Y』AC 『Y』OV
sub a, M	Subtraction data in memory from ACC, then put result into ACC Example: sub a, MEM ; Result: a ← a - MEM (a + [2's complement of M]) Affected flags: 『Y』Z 『Y』C 『Y』AC 『Y』OV
sub M, a	Subtraction data in ACC from memory, then put result into memory Example: sub MEM, a; Result: MEM ← MEM - a (MEM + [2's complement of a]) Affected flags: 『Y』Z 『Y』C 『Y』AC 『Y』OV
subc a, M	Subtraction data in memory and carry from ACC, then put result into ACC Example: subc a, MEM; Result: a ← a – MEM - C Affected flags: 『Y』Z 『Y』C 『Y』AC 『Y』OV
subc M, a	Subtraction ACC and carry bit from memory, then put result into memory Example: subc MEM, a ; Result: MEM ← MEM – a - C Affected flags: 『Y』Z 『Y』C 『Y』AC 『Y』OV
subc a	Subtraction carry from ACC, then put result into ACC Example: subc a; Result: a ← a - C Affected flags: 『Y』Z 『Y』C 『Y』AC 『Y』OV
subc M	Subtraction carry from the content of memory, then put result into memory Example: subc MEM; Result: MEM ← MEM - C Affected flags: 『Y』Z 『Y』C 『Y』AC 『Y』OV
inc M	Increment the content of memory Example: inc MEM ; Result: MEM ← MEM + 1 Affected flags: 『Y』Z 『Y』C 『Y』AC 『Y』OV
dec M	Decrement the content of memory Example: dec MEM; Result: MEM ← MEM - 1 Affected flags: 『Y』Z 『Y』C 『Y』AC 『Y』OV
clear M	Clear the content of memory Example: clear MEM ; Result: MEM ← 0 Affected flags: 『N』Z 『N』C 『N』AC 『N』OV

14.22. Shift Operation Instructions

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

14.23. Logic Operation Instructions

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

	Example: xor MEM, a ; Result: MEM ← a ^ MEM Affected flags: 『Y』Z 『N』C 『N』AC 『N』OV
not a	Perform 1's complement (logical complement) of ACC Example: not a ; Result: a ← ～a Affected flags: 『Y』Z 『N』C 『N』AC 『N』OV Application Example: --- mov a, 0x38 ; // ACC=0X38 not a ; // ACC=0XC7 ---
not M	Perform 1's complement (logical complement) of memory Example: not MEM ; Result: MEM ← ～MEM Affected flags: 『Y』Z 『N』C 『N』AC 『N』OV Application Example: --- mov a, 0x38 ; mov mem, a ; // mem = 0x38 not mem ; // mem = 0xC7 ---
neg a	Perform 2's complement of ACC Example: neg a; Result: a ← ∓a Affected flags: 『Y』Z 『N』C 『N』AC 『N』OV Application Example: --- mov a, 0x38 ; // ACC=0X38 neg a ; // ACC=0XC8 ---
neg M	Perform 2's complement of memory Example: neg MEM; Result: MEM ← ∓MEM Affected flags: 『Y』Z 『N』C 『N』AC 『N』OV
	Application Example: --- mov a, 0x38 ; mov mem, a ; // mem = 0x38 neg mem ; // mem = 0xC8 ---
comp a, I	Compare ACC with immediate data Example: comp a, 0x55;

	Result: Flag will be changed by regarding as (a - 0x55) Affected flags: 『Y』Z 『Y』C 『Y』AC 『Y』OV Application Example: --- mov a, 0x38 ; comp a, 0x38 ; // Z flag is set comp a, 0x42 ; // C flag is set comp a, 0x24 ; // C, Z flags are clear comp a, 0x6a ; // C, AC flags are set ---
comp a, M	Compare ACC with the content of memory Example: comp a, MEM; Result: Flag will be changed by regarding as (a - MEM) Affected flags: 『Y』Z 『Y』C 『Y』AC 『Y』OV Application Example: --- mov a, 0x38 ; mov mem, a ; comp a, mem ; // Z flag is set mov a, 0x42 ; mov mem, a ; mov a, 0x38 ; comp a, mem ; // C flag is set ---
comp M, a	Compare ACC with the content of memory Example: comp MEM, a; Result: Flag will be changed by regarding as (MEM - a) Affected flags: 『Y』Z 『Y』C 『Y』AC 『Y』OV

14.24. Bit Operation Instructions

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

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

14.25. Conditional Operation Instructions

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

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

14.26. System control Instructions

call label	Function call, address can be full range address space Example: *call function1;* Result: [sp] ← pc + 1 pc ← function1 sp ← sp + 2 Affected flags: 『N』Z 『N』C 『N』AC 『N』OV
goto label	Go to specific address which can be full range address space Example: *goto error;* Result: Go to error and execute program. Affected flags: 『N』Z 『N』C 『N』AC 『N』OV
delay a	Delay the (N＋1) cycles which N is specified by the content of ACC, the timing is based on the executing FPP unit. After the *delay* instruction is executed, the ACC will be zero. Example: *delay a;* 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.
delay I	Delay the (N＋1) cycles which N is specified by the immediate data, the timing is based on the executing FPP unit. After the *delay* instruction is executed, the ACC will be zero. Example: *delay 0x05;* 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.
delay M	Delay the (N＋1) cycles which N is specified by the content of memory, the timing is based on the executing FPP unit. After the delay instruction is executed, the ACC will be zero. Example: delay M; 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.
ret I	Place immediate data to ACC, then return Example: ret 0x55; Result: A← 55h ret ; Affected flags: 『N』Z 『N』C 『N』AC 『N』OV
ret	Return to program which had function call Example: ret; Result: sp ← sp - 2 pc ← [sp] Affected flags: 『N』Z 『N』C 『N』AC 『N』OV
reti	Return to program that is interrupt service routine. After this command is executed, global interrupt is enabled automatically. Example: reti; Affected flags: 『N』Z 『N』C 『N』AC 『N』OV
nop	No operation Example: nop; Result: nothing changed Affected flags: 『N』Z 『N』C 『N』AC 『N』OV
pcadd a	Next program counter is current program counter plus ACC. Example: pcadd a; Result: pc ← pc + a Affected flags: 『N』Z 『N』C 『N』AC 『N』OV Application Example: --- ... mov a, 0x02 ; pcadd a ; // PC <- PC+2 goto err1 ; goto correct ; // jump here

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

14.27. Summary of Instructions Execution Cycle

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