PMC251 Series

IO-Type Enhanced
Field Programmable Processor Array
(FPPA™) 8-bit Controller

Data Sheet

Version 0.12 – May 20, 2014

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

TEL: 886-3-532-7598 www.padauk.com.tw

IMPORTANT NOTICE

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

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

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

Table of Contents

Revision History:

Revision	Date	Description
0.02	2012/5/1	1ST version
0.03	2012/6/25	1. Amend 4.5 chart of IHRC Frequency vs. Temperature 2. Amend 4.6 chart of ILRC Frequency vs. Temperature 3. Amend clkmd register description
0.04	2012/7/25	1. Remove *delay* instruction
0.05	2012/8/2	1. Amend clkmd register: Watchdog timer is disabled when ILRC is disabled.
0.06	2012/8/8	1. Add padier, pbdier registers notice in ICE
0.07	2012/9/7	1. Amend PAPH and PBPH registers to be “read/write” 2. Add Order Information PMC251 – S14 3. Amend LVD DC specification 4. Amend Fig. 9 system clock options 5. Amend watchdog timeout period 6. Amend the description of 5.4. Boot up procedure 7. Amend the description of misc.4 -- Recover time from LVD reset 8. Arrange the example of 5.7.4 9. Amend bit[7:5] of t16m register to be consistent with ICE 10. Remove rstst register to be consistent with ICE 11. Amend 5.13. about rstst register description to be consistent with ICE 12. Add notes for using padier and pbdier registers to unify the same program for both ICE emulation and real chip. 13. Amend Chart 4.3 and 4.4 – IHRC/ILRC frequency vs. VDD 14. Amend t_{WUP} system wake-up time and 5-11-3, remove Fig. 15 15. Amend 5.7.2 chip calibration, **p2** =16～18;
0.08	2012/10/3	1. Amend Chart 4.3, 4.5, 4.6 2. Remove Band-gap specification 3. Amend Fig. 10, remove PA4 input 4. Amend f_{IHRC} specification 5. Add 5-11-1 application notes: (A) watchdog timer must be disabled before stopexe (B) the example of wake-up sources from stopexe
0.09	2012/11/23	1. Add section 5-7-6 system clock switching 2. Add DIP14 package information
0.10	2013/03/31	1. Replace $I_{stand\ by}$ as $I_{power\ down}$ 2. Add package DIP14 in Pin diagram 3. Add notice in page 40 : The fast wake-up will be turned off automatically when EOSC is selected as system clock. 4 Remove package information
0.11	2013/10/30	1. Add V_{IN} and $I_{INJ\ (PIN)}$ in DC/AC characteristics 2. Amend 7.8 Summary of instructions Execution cycles
0.12	2014/5/20	1. Add chapter 8 Special Notes

1. Features

1.1. CPU Features

- Operating modes: Two processing units FPPA™ mode or Traditional one processing unit mode
- 1KW OTP program memory
- 60 Bytes data RAM
- 99 powerful instructions
- Most instructions are 1T execution cycle
- Direct and indirect addressing modes for data and instructions
- Bit-manipulation instructions
- Programmable stack pointer and adjustable stack level
- All data memories are available for use as an index pointer
- Support security function to protect OTP data
- Separated IO space and memory space

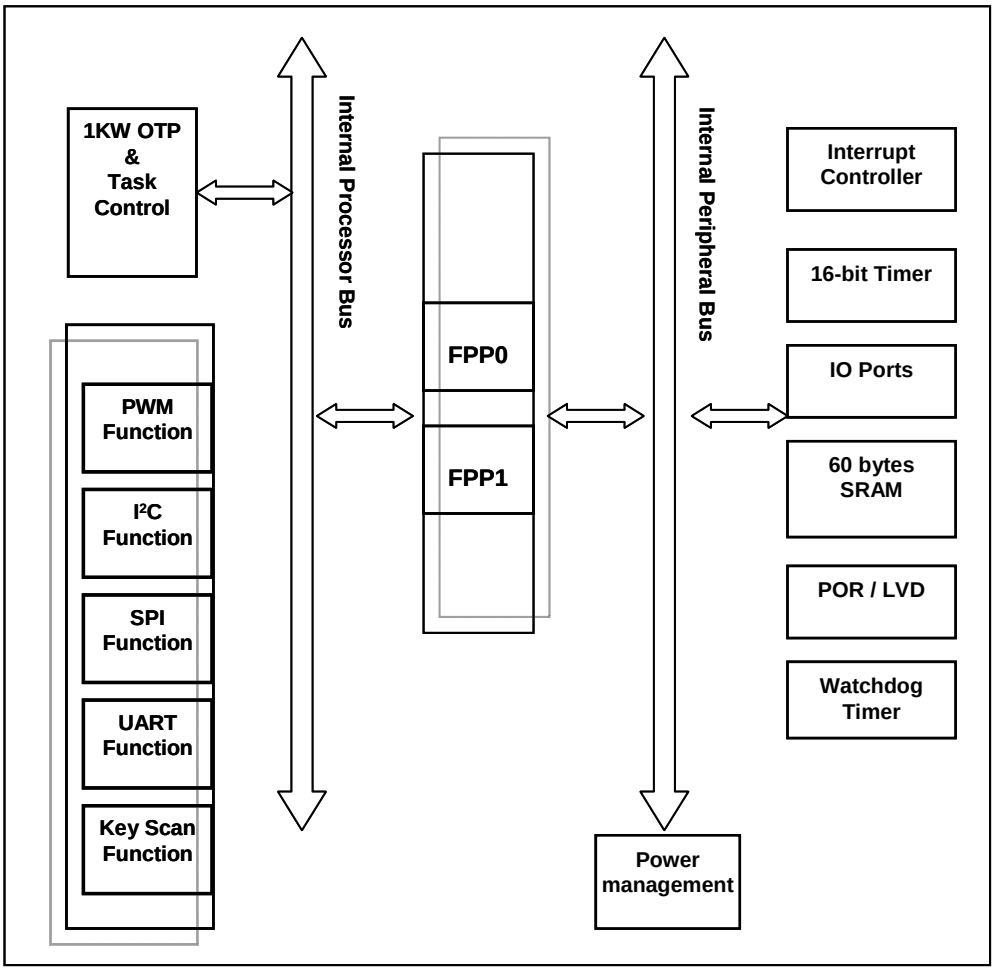
1.2. System Functions

- Clock sources: internal high RC oscillator, internal low RC oscillator and external crystal oscillator
- Internal High RC Oscillator (IHRC) frequency
- Band-gap circuit to provide 1.20V reference voltage
- One hardware 16-bit timer
- Support fast wake-up
- Eight levels of LVD reset ~ 4.1V, 3.6V, 3.1V, 2.8V, 2.5V, 2.2V, 2.0V, 1.8V
- 12 IO pins with 10mA capability and optional pull-high resistor
- Two external interrupt pins
- Every IO pin can be configured to enable wake-up function
- Operating frequency range:
  DC ~ 8MHz@VDD≧3.3V; DC ~ 4MHz@VDD≧2.5V; DC ~ 2MHz@VDD≧2.2V
- Operating voltage range: 2.2V ~ 5.5V
- Operating temperature range: -40°C ~ 85°C
- Low power consumption
  $I_{operating}$ ~ 1.7mA@1MIPS, VDD=5.0V $I_{operating}$ ~ 8uA@ILRC=12KHz, VDD=3.3V
  $I_{powerdown}$ ~ 0.7uA@VDD=5.0V $I_{powerdown}$ ~ 0.4uA@VDD=3.3V
- Order Information:
  PMC251 - S14: SOP14 (150mil); PMC251 - D14: DIP14 (300mil);

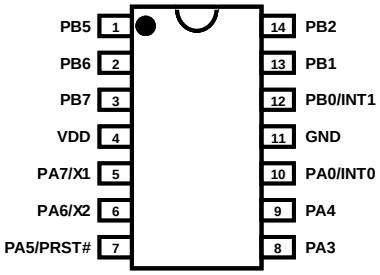
2. General Description and Block Diagram

The PMC251 family is an IO-Type of PADAUK's parallel processing, fully static, OTP-based CMOS 2x8 bit processor array that can execute two peripheral functions in parallel. Besides operated in two processing units, PMC251 can act as traditional MCU in one processing unit. The PMC251 employs RISC architecture based on patented FPPA™ (Field Programmable Processor Array) technology and most the instructions are executed in one cycle except that few instructions are two cycles that handle indirect memory access.

1KW OTP program memory and 60 bytes data SRAM are inside for two FPP units using, one hardware 16-bit timer is also provided in the PMC251. The functions for peripheral devices like UART and PWM can be easily implemented by using FPPA™ unique architecture.



3. Pin Assignment and Description



PMC251 (SOP14-150mil)
PMC251 (DIP14-300mil)

Pin Description

Pin Name	Pin & Buffer Type	Description
PA7/X1	IO ST / CMOS / Analog	The functions of this pin can be: (1) Bit 7 of port A. It can be configured as input or output with pull-up resistor. (2) X1 (input) when crystal oscillator is used. If this pin is used for crystal oscillator, bit 7 of ***padier*** register must be programmed "0" to avoid leakage current. This pin can be used to wake-up system during sleep mode; however, wake-up function is also disabled if bit 7 of ***padier*** register is "0".
PA6/X2	IO ST / CMOS / Analog	The functions of this pin can be: (1) Bit 6 of port A. It can be configured as input or output with pull-up resistor. (2) X2 (output) when crystal oscillator is used. If this pin is used for crystal oscillator, bit 6 of ***padier*** register must be programmed "0" to avoid leakage current. This pin can be used to wake-up system during sleep mode; however, wake-up function is also disabled if bit 6 of ***padier*** register is "0".
PA5/PRST#	IO ST / CMOS	The functions of this pin can be: (1) Bit 5 of port A. It can be configured as input or open-drain output pin. Please notice that there is no pull-up resistor in this pin. (2) Hardware reset. This pin can be used to wake-up system during sleep mode; however, wake-up function is also disabled if bit 5 of ***padier*** register is "0". Please put 33Ω resistor in series to have high noise immunity when this pin is in input mode.
PA4	IO ST / CMOS	The functions of this pin can be bit 4 of port A. It can be configured as input or output with pull-up resistor. This pin can be used to wake up system during sleep mode; however, wake-up function from this pin is disabled when bit 4 of ***padier*** register is "0"
PA3	IO ST / CMOS	The functions of this pin can be bit 3 of port A. It can be configured as input or output with pull-up resistor. This pin can be used to wake up system during sleep mode; however, wake-up function from this pin is disabled when bit 3 of ***padier*** register is "0"

Pin Name	Pin Type & Buffer Type	Description
PA0/INT0	IO ST / CMOS	The functions of this pin can be: (1) Bit 0 of port A. It can be configured as input or output with pull-up resistor. (2) External interrupt line 0. Both rising edge and falling edge are accepted to request interrupt service. This pin can be used to wake up system during sleep mode; however, wake-up function from this pin is also disabled when bit 0 of ***padier*** register is "0".
PB7	IO ST / CMOS	The function of this pin is Bit 7 of port B. It can be configured as input or output with pull-up resistor. This pin can be used to wake up system during sleep mode; however, wake-up function from this pin is also disabled when bit 7 of ***pbdier*** register is "0".
PB6	IO ST / CMOS	The function of this pin is Bit 6 of port B. It can be configured as input or output with pull-up resistor. This pin can be used to wake up system during sleep mode; however, wake-up function from this pin is also disabled when bit 6 of ***pbdier*** register is "0".
PB5	IO ST / CMOS	The function of this pin is Bit 5 of port B. It can be configured as input or output with pull-up resistor. This pin can be used to wake up system during sleep mode; however, wake-up function from this pin is also disabled when bit 5 of ***pbdier*** register is "0".
PB2	IO ST / CMOS	The function of this pin is Bit 2 of port B. It can be configured as input or output with pull-up resistor. This pin can be used to wake up system during sleep mode; however, wake-up function from this pin is also disabled when bit 2 of ***pbdier*** register is "0".
PB1	IO ST / CMOS	The function of this pin is Bit 1 of port B. It can be configured as input or output with pull-up resistor. This pin can be used to wake up system during sleep mode; however, wake-up function from this pin is also disabled when bit 1 of ***pbdier*** register is "0".
PB0/INT1	IO ST / CMOS	The functions of this pin can be: (1) Bit 0 of port B. It can be configured as input or output with pull-up resistor. (2) External interrupt line 1. Both rising edge and falling edge are accepted to request interrupt service. This pin can be used to wake up system during sleep mode; however, wake-up function from this pin is also disabled when bit 0 of ***pbdier*** register is "0".
VDD		Positive power
GND		Ground
Notes: IO: Input/Output; ST: Schmitt Trigger input; Analog: Analog input pin; CMOS: CMOS voltage level		

4. Device Characteristics

4.1. DC/AC Characteristics

All data are acquired under the conditions of Ta = - 40 °C ~ 85 °C, Vdd=5.0V, f_{SYS}=2MHz unless noted.

Symbol	Description	Min	Typ	Max	Unit	Conditions
V_{DD}	Operating Voltage	2.2	5.0	5.5	V	f_{SYS} = 2MHz
f_{SYS}	System clock* = IHRC/2 IHRC/4 & crystal oscillator IHRC/8 & crystal oscillator ILRC	 0 0 0	 24K	 8M 4M 2M	Hz	Under_20ms_Vdd_ok**=Y/N V_{DD}≧3.0V / V_{DD}≧3.3V V_{DD}≧2.5V / V_{DD}≧2.8V V_{DD}≧2.2V / V_{DD}≧2.2V V_{DD} = 5.0V
I_{OP}	Operating Current		1.7 8		mA uA	f_{SYS}=1MIPS@5.0V f_{SYS}=ILRC=12KHz@3.3V
I_{PD}	Power Down Current (by ***stopsy***s command)		0.7 0.4		uA uA	f_{SYS}= 0Hz,VDD=5.0V f_{SYS}= 0Hz,VDD=3.3V
I_{PS}	Power Save Current (by ***stopexe*** command)		0.4		mA	VDD=5.0V; Band-gap, LVD, IHRC, ILRC, Timer16 modules are ON.
V_{IL}	Input low voltage for IO lines	0		$0.3V_{DD}$	V	
V_{IH}	Input high voltage for IO lines	$0.7 V_{DD}$		V_{DD}	V	
I_{OL}	IO lines sink current	7	10	13	mA	V_{DD}=5.0V, V_{OL}=0.5V
I_{OH}	IO lines drive current	-5	-7	-9	mA	V_{DD}=5.0V, V_{OH}=4.5V
V_{IN}	Input voltage	-0.3		VDD+0.3	V	
$I_{INJ (PIN)}$	Injected current on pin			1	mA	VDD+0.3≧V_{IN}≧ -0.3
R_{PH}	Pull-high Resistance		62 100 210		KΩ	V_{DD}=5.0V V_{DD}=3.3V V_{DD}=2.2V
V_{BRD}	Low Voltage Detect Voltage * (Brown-out voltage)	3.86 3.35 2.84 2.61 2.37 2.04 1.86 1.67	4.15 3.60 3.05 2.80 2.55 2.20 2.00 1.80	4.44 3.85 3.26 3.00 2.73 2.35 2.14 1.93	V	
f_{IHRC}	Frequency of IHRC after calibration *	15.68* 14.72* 14.88*	16* 16* 16*	16.32* 17.28* 17.12*	MHz MHz	25°C, VDD=2.2V~5.5V VDD=2.2V~5.5V, -40°C <Ta<85°C* 0°C <Ta<70°C*
f_{ILRC}	Frequency of ILRC *	20.4* 15.6* 16.8* 10.2* 7.8* 8.4*	24* 24* 24* 12* 12* 12*	27.6* 32.4* 31.2* 13.8* 16.2* 15.6*	KHz	VDD=5.0V, Ta=25°C VDD=5.0V, -40°C <Ta<85°C* VDD=5.0V, 0°C <Ta<70°C* VDD=3.3V, Ta=25°C VDD=3.3V, -40°C <Ta<85°C* VDD=5.0V, 0°C <Ta<70°C*

Symbol	Description	Min	Typ	Max	Unit	Conditions
t_{INT}	Interrupt pulse width	30			ns	V_{DD} = 5.0V
V_{DR}	RAM data retention voltage*	1.5			V	In power-down mode.
t_{WDT}	Watchdog timeout period		2048		T_{ILRC}	misc[1:0]=00 (default)
			4096			misc[1:0]=01
			16384			misc[1:0]=10
			256			misc[1:0]=11
t_{SBP}	System boot-up period from power-on		1024		T_{ILRC}	Where T_{ILRC} is the clock period of ILRC
t_{WUP}	System wake-up period					
	Fast wake-up by IO toggle from STOPEXE suspend		128		T_{SYS}	Where T_{SYS} is the time period of system clock
	Fast wake-up by IO toggle from STOPSYS suspend, IHRC is the system clock		128 T_{SYS} + T_{SIHRC}			Where T_{SIHRC} is the stable time of IHRC from power-on. T_{SIHRC} = 5us@ VDD=5V
	Fast wake-up by IO toggle from STOPSYS suspend, ILRC is the system clock		128 T_{SYS} + T_{SILRC}			Where T_{SILRC} is the stable time of ILRC from power-on. T_{SILRC} = 43ms@ VDD=5V
	Normal wake-up from STOPEXE or STOPSYS suspend		1024		T_{ILRC}	Where T_{ILRC} is the clock period of ILRC
t_{RST}	External reset pulse width	120			us	@VDD=5V

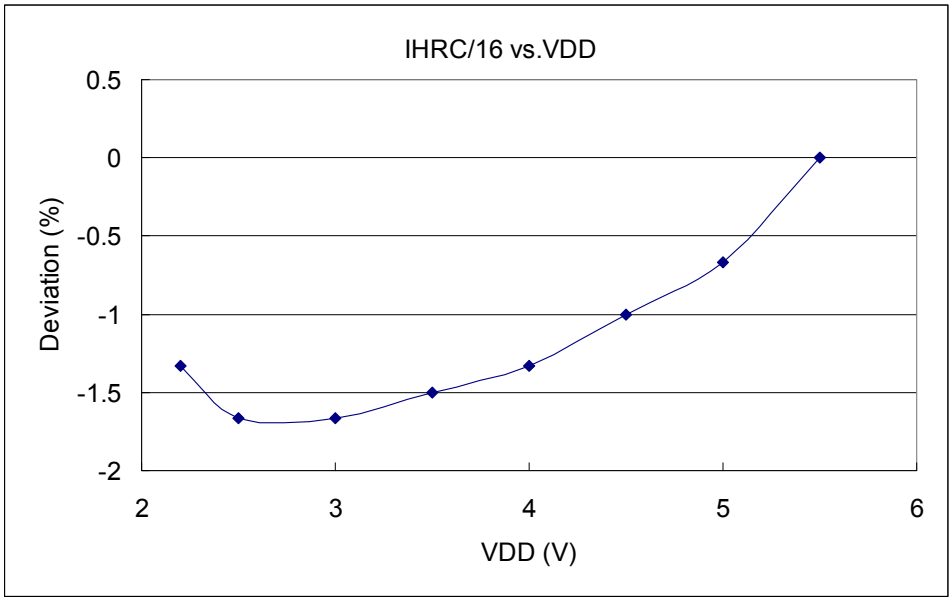
*These parameters are for design reference, not tested for every chip.

** Under_20ms_Vdd_Ok is a checking condition for the VDD rising from 0V to the stated voltage within 20ms.

4.2. Absolute Maximum Ratings

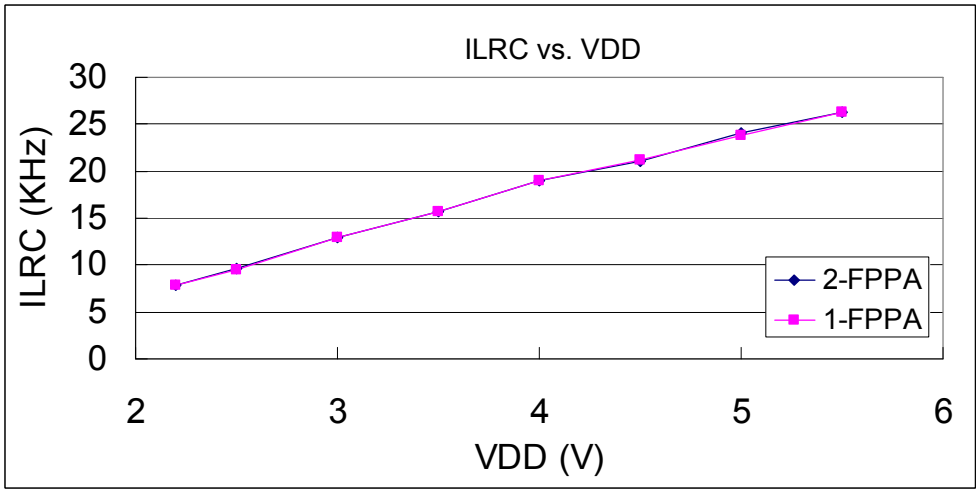
- Supply Voltage ……..... 2.2V ~ 5.5V
- Input Voltage …….. -0.3V ~ VDD + 0.3V
- Operating Temperature …… -40°C ~ 85°C
- Storage Temperature …… -50°C ~ 125°C

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

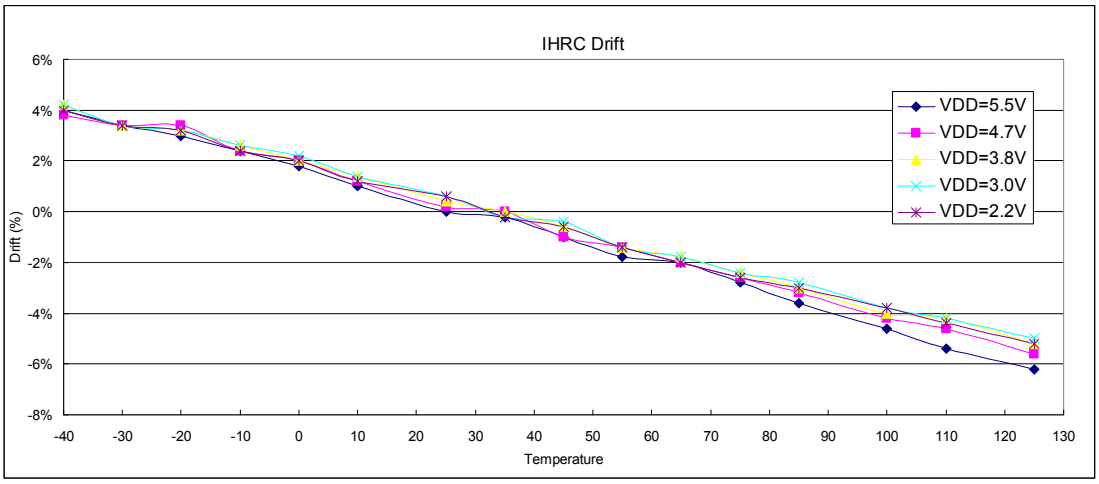


Note: The deviation may be slightly different under different system clock

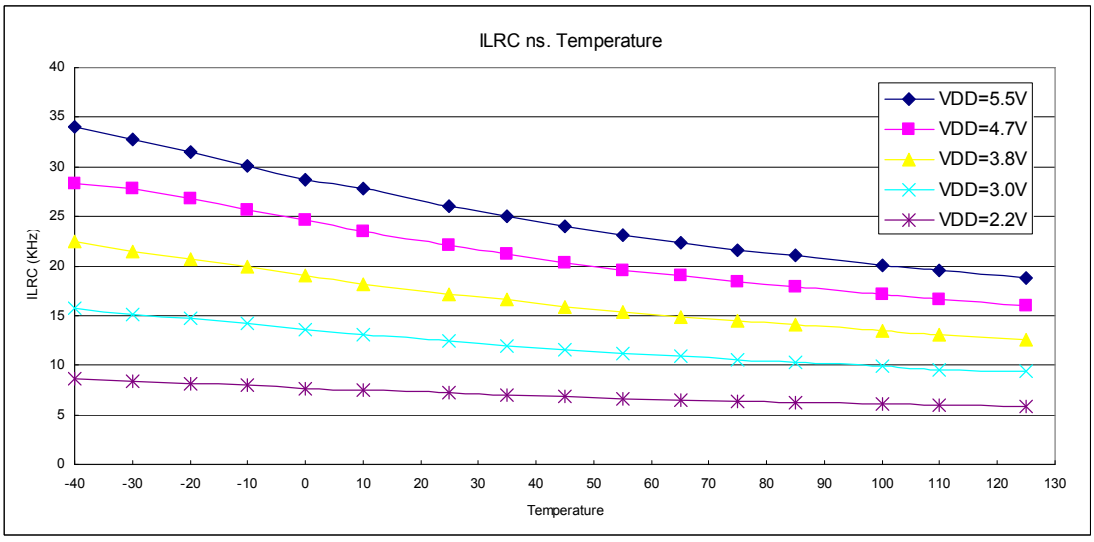
4.4. Typical ILRC Frequency vs. VDD



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



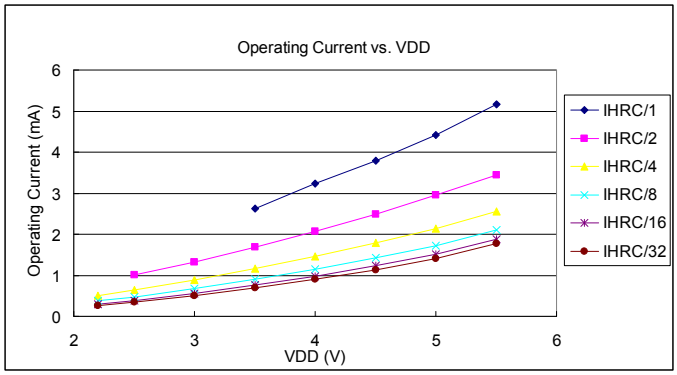
4.6. Typical ILRC Frequency vs. Temperature



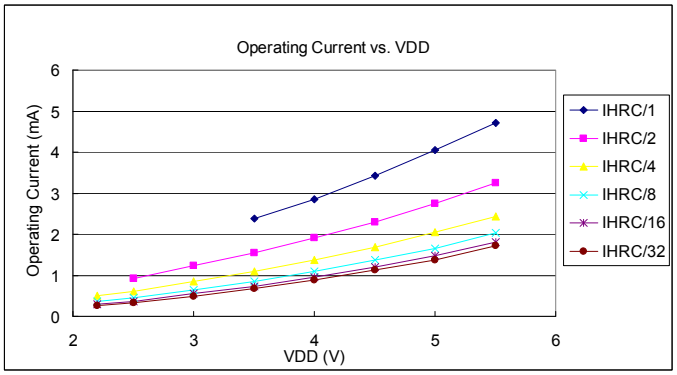
4.7. Typical Operating Current vs. VDD and CLK=IHRC/n

➢ Conditions: **ON**: Band-gap, LVD, IHRC, T16 modules; **OFF**: ILRC, EOSC modules;
IO: PA0:0.5Hz output toggle and no loading, others: input and no floating

Two FPP units mode



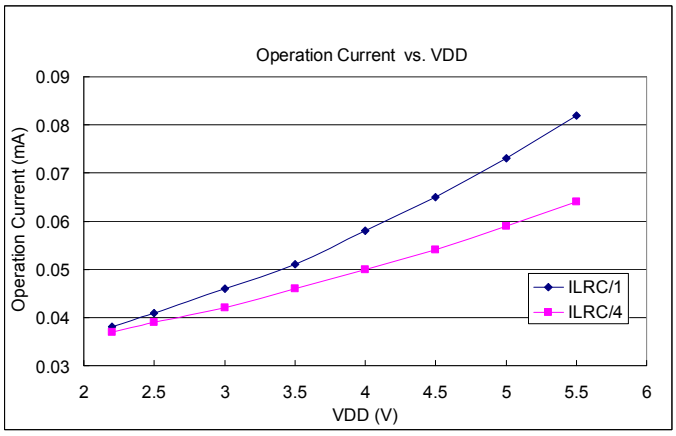
One FPP unit mode



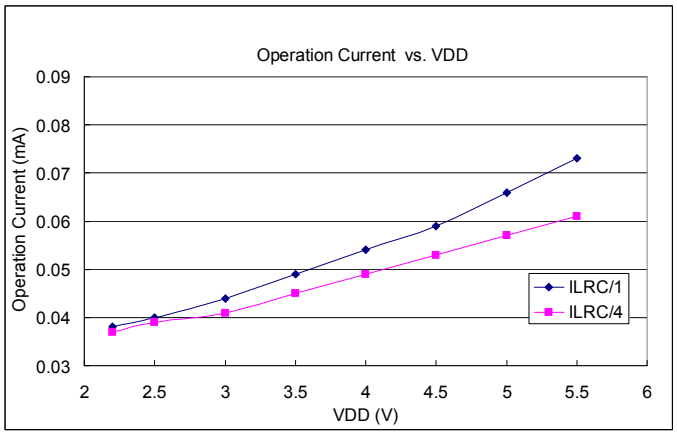
4.8. Typical Operating Current vs. VDD and CLK=ILRC/n

➢ Conditions: **ON**: Band-gap, LVD, ILRC, T16 modules; **OFF**: IHRC, EOSC modules;
IO: PA0:0.5Hz output toggle and no loading, others: input and no floating

Two FPP units mode



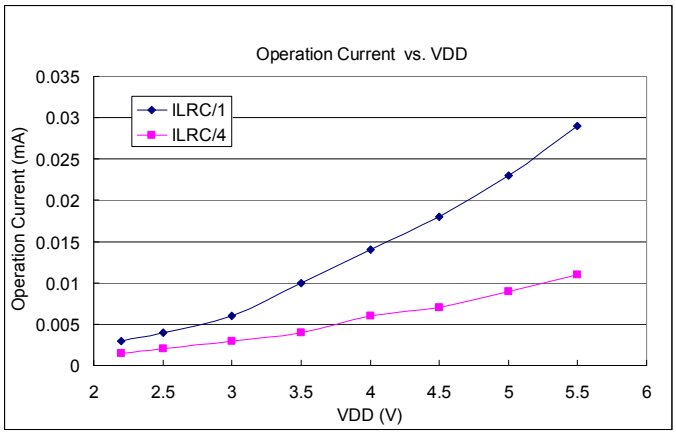
One FPP unit mode



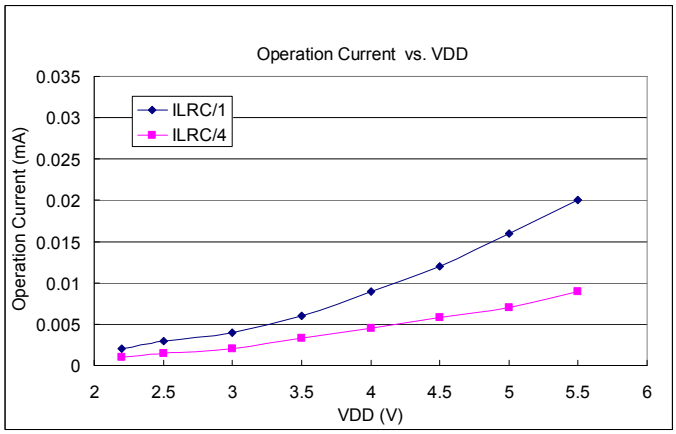
4.9. Typical Lowest Operating Current vs. VDD and CLK=ILRC/n

- Conditions: **ON**: ILRC , T16 module; **OFF**: IHRC, Band-gap, LVD, EOSC modules;
  IO: PA0:0.5Hz output toggle and no loading, others: input and no floating

Two FPP units mode



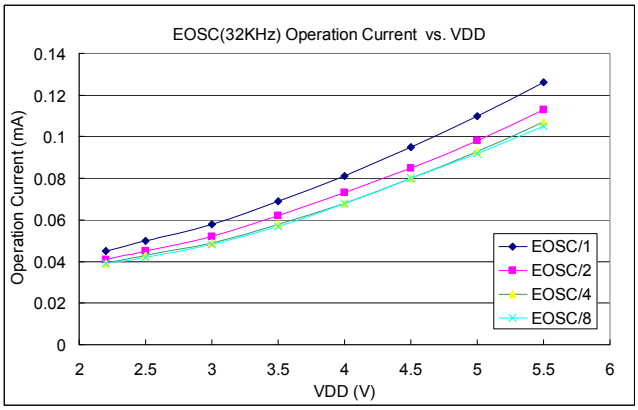
One FPP unit mode



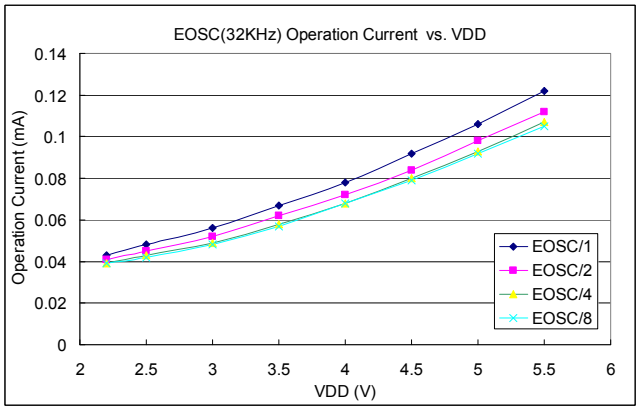
4.10. Typical Operating Current vs. VDD @CLK=32KHz EOSC/n

- Conditions: **ON**: EOSC, Band-gap, LVD, T16 modules; **OFF**: IHRC, ILRC modules;
  IO: PA0:0.5Hz output toggle and no loading, others: input and no floating
  EOSC: High driving current

Two FPP units mode

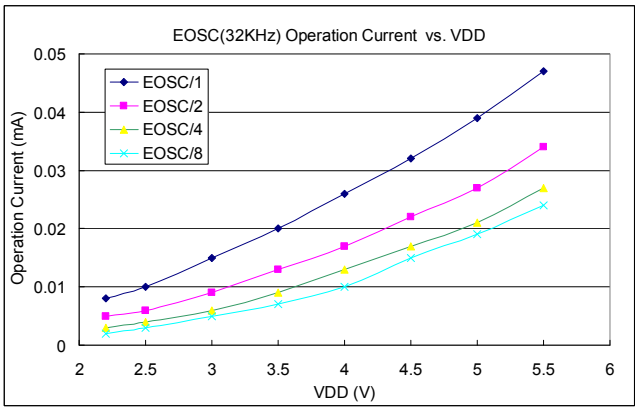


One FPP unit mode

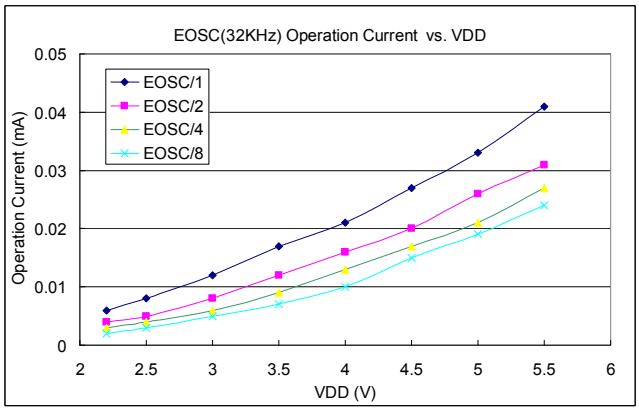


- Conditions: **ON**: EOSC, T16 module; **OFF**: IHRC, ILRC, Band-gap, LVD modules;
  IO: PA0:0.5Hz output toggle and no loading, others: input and no floating
  EOSC: Low driving current

Two FPP units mode



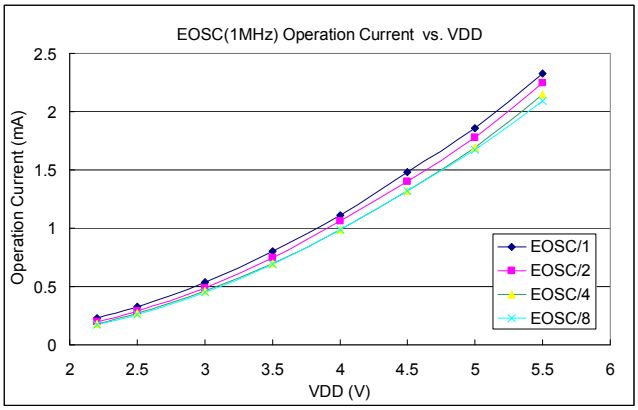
One FPP unit mode



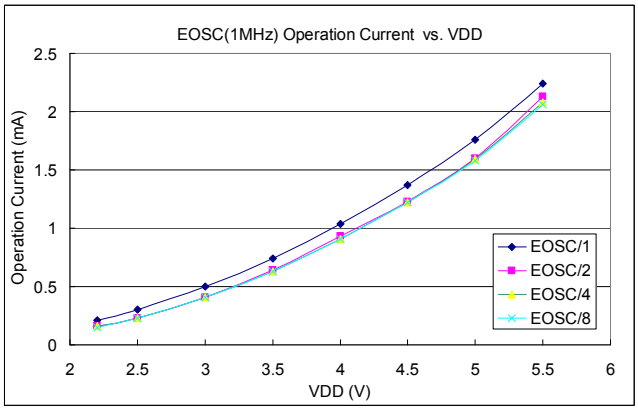
4.11. Typical Operating Current vs. VDD @CLK=1MHz EOSC/n

- Conditions: **ON**: EOSC, Band-gap, LVD, T16 modules; **OFF**: IHRC, ILRC modules;
  IO: PA0:0.5Hz output toggle and no loading, others: input and no floating
  EOSC: High driving current

Two FPP units mode

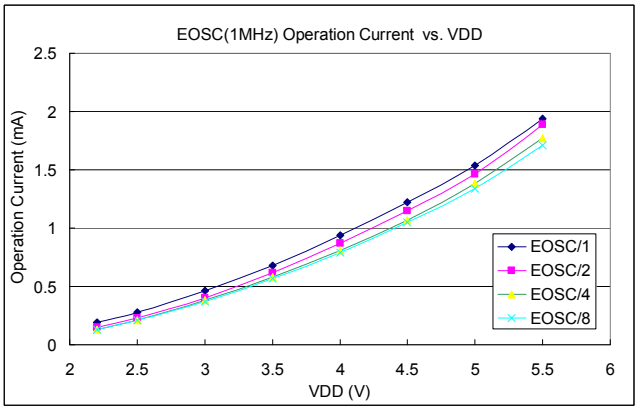


One FPP unit mode

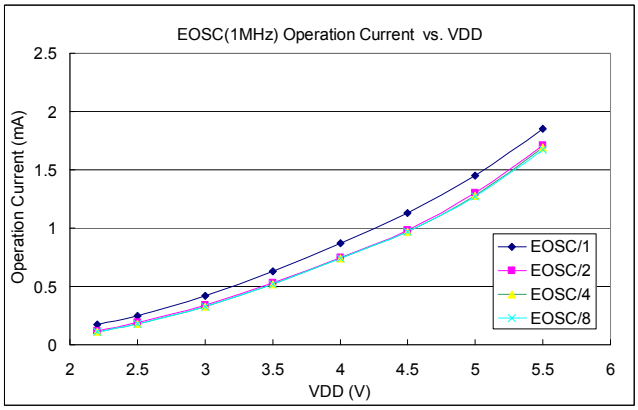


- Conditions: **ON**: EOSC, T16 module; **OFF**: IHRC, ILRC, Band-gap, LVD modules;
  IO: PA0:0.5Hz output toggle and no loading, others: input and no floating
  EOSC: Low driving current

Two FPP units mode



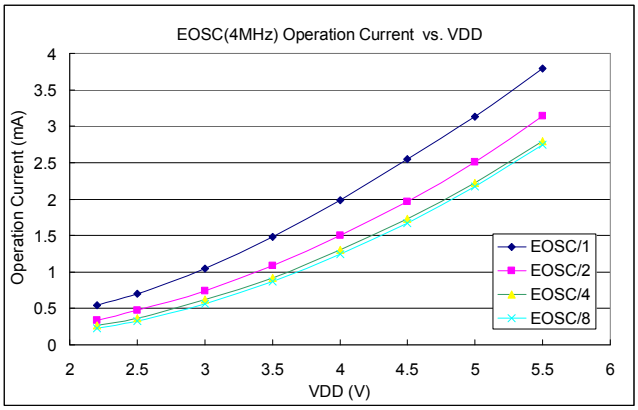
One FPP unit mode



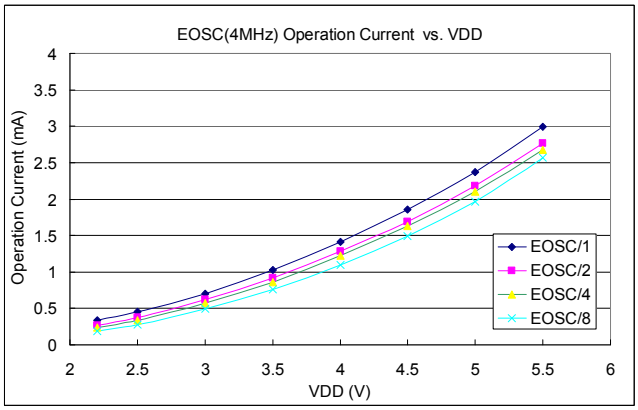
4.12. Typical Operating Current vs. VDD @CLK=4MHz EOSC/n

- Conditions: **ON**: EOSC, Band-gap, LVD, T16 modules; **OFF**: IHRC, ILRC modules;
  IO: PA0:0.5Hz output toggle and no loading, others: input and no floating
  EOSC: High driving current

Two FPP units mode

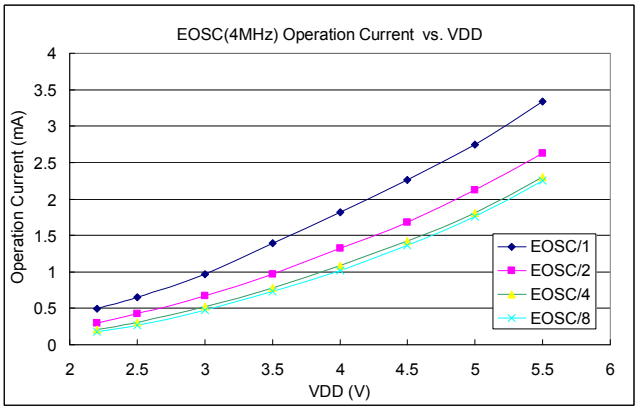


One FPP unit mode

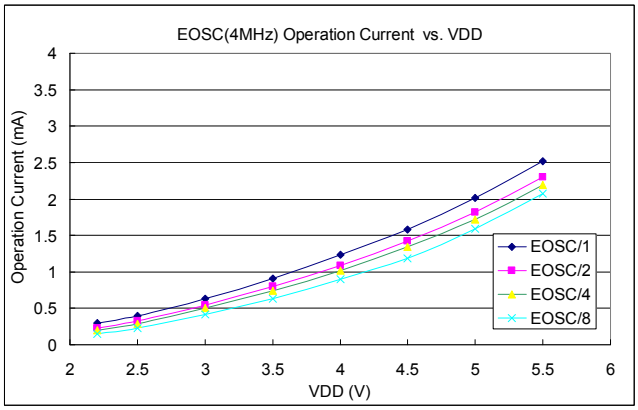


- Conditions: **ON**: EOSC, T16 module; **OFF**: IHRC, ILRC, Band-gap, LVD modules;
  IO: PA0:0.5Hz output toggle and no loading, others: input and no floating
  EOSC: Low driving current

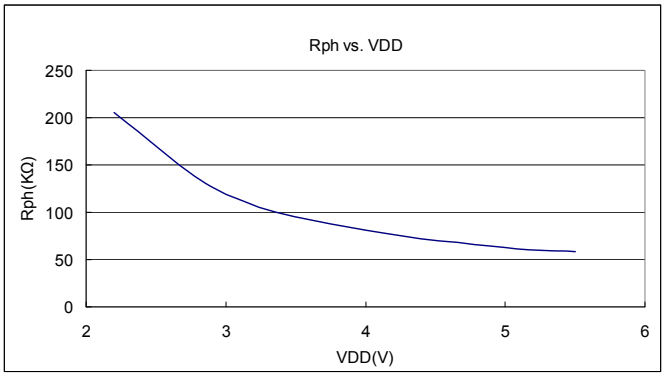
Two FPP units mode



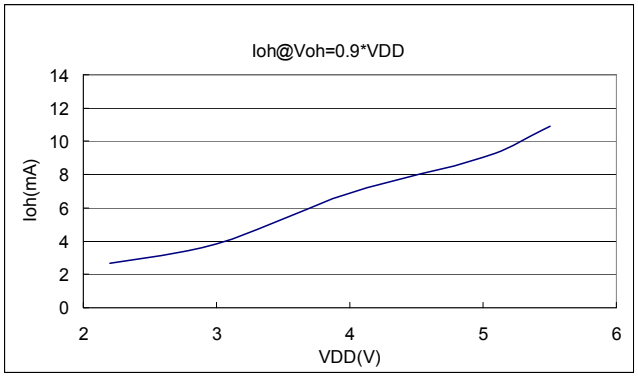
One FPP unit mode

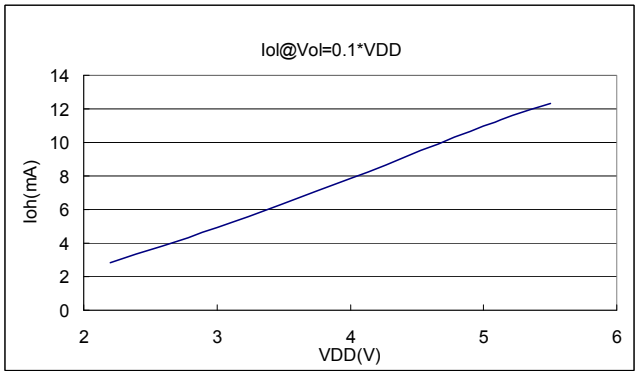


4.13. Typical IO pull high resistance

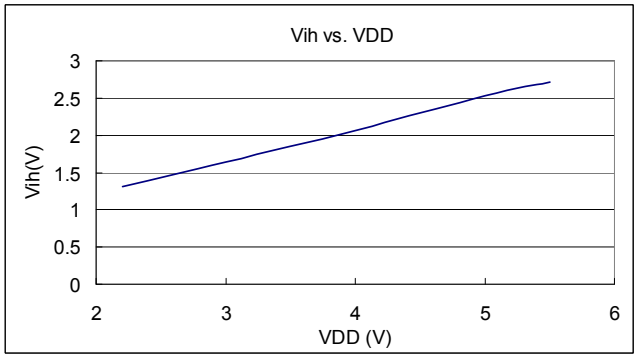


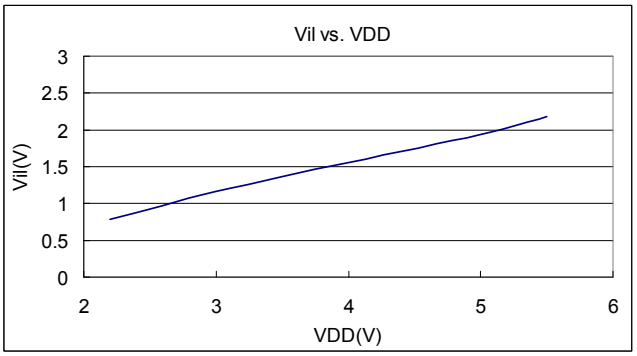
4.14. Typical IO driving current (I_{OH}) and sink current (I_{OL})





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





5. Functional Description

5.1. Processing Units

There are two processing units (FPP unit) inside the PMC251. In each processing unit, it includes (i) its own Program Counter to control the program execution sequence (ii) its own Stack Pointer to store or restore the program counter for program execution (iii) its own accumulator (iv) Status Flag to record the status of program execution. Each FPP unit has its own program counter and accumulator for program execution, flag register to record the status, and stack pointer for jump operation. Based on such architecture, FPP unit can execute its own program independently, thus parallel processing can be expected.

These two FPP units share the same 1Kx16 bits OTP program memory, 64 bytes data SRAM and all the IO ports, these two FPP units are operated at mutual exclusive clock cycles to avoid interference. One task switch is built inside the chip to decide which FPP unit should be active for the corresponding cycle. The hardware diagram and basic timing diagram of FPP units are illustrated in Fig. 1. For FPP0 unit, its program will be executed in sequence every other system clock, shown as $(M-1)_{th}$, M_{th} and $(M+1)_{th}$ instructions. For FPP1 unit, its program will be also executed in sequence every other system clock, shown as $(N-1)_{th}$, N_{th} and $(N+1)_{th}$ instructions.

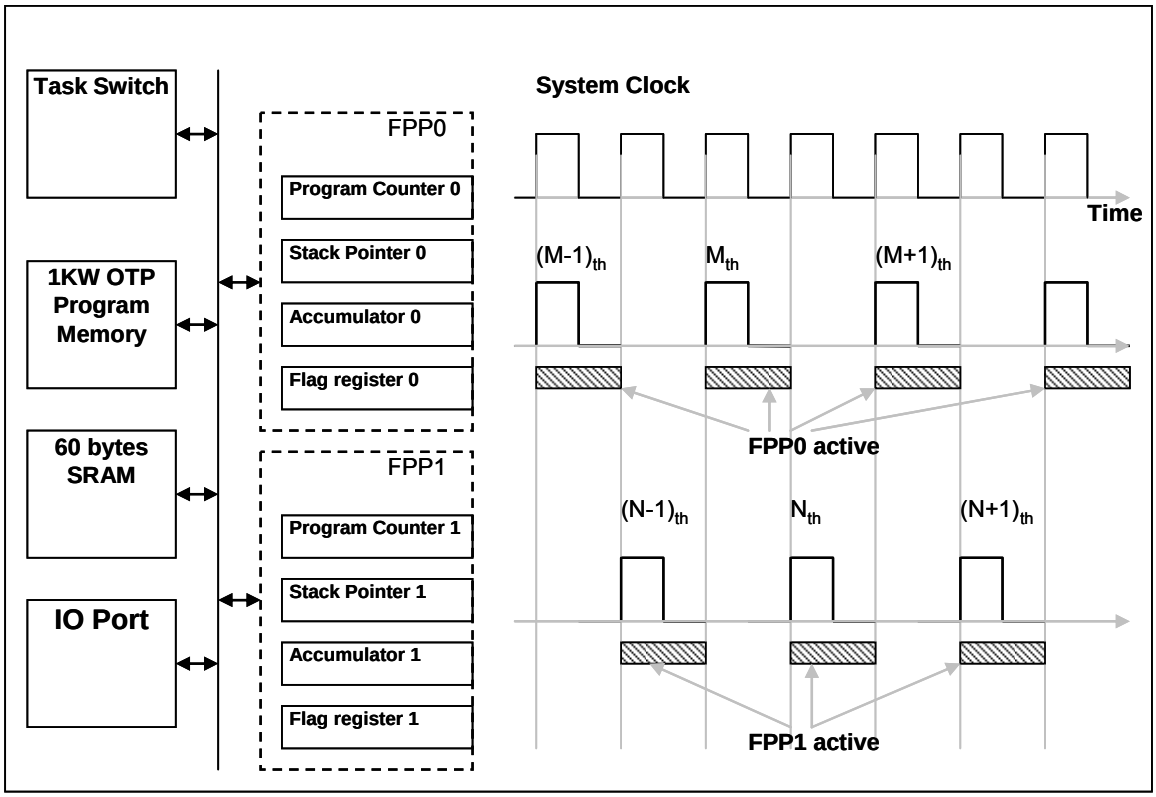


Fig. 1 Hardware and Timing Diagram of FPP units

Each FPP unit has half computing power of whole system; for example, FPP0 and FPP1will be operated at 4MHz if system clock is 8MHz. The FPP unit can be enabled or disabled by programming the FPP unit Enable Register, only FPP0 is enabled after power-on reset. The system initialization will be started from FPP0 and FPP1 unit can be enabled by user's program if necessary. Both FPP0 and FPP1 can be enabled or disabled by using any one FPP unit.

5.1.1. Program Counter

Program Counter (PC) is the unit that contains the address of an instruction to be executed next. The program counter is automatically incremented at each instruction cycle so that instructions are retrieved sequentially from the program memory. Certain instructions, such as branches and subroutine calls, interrupt the sequence by placing a new value in the program counter. The bit length of the program counter is 10 for PMC251. The initial program counter of FPP0 is 0 after hardware reset and 1 for FPP1. Whenever any interrupt happens, the program counter will jump to 'h10 for interrupt service routine, only FPP0 accept interrupt and each FPP unit has its own program counter to control the program execution sequence.

5.1.2. Stack Pointer

The stack pointer in each processing unit is used to point the top of the stack area where the local variables and parameters to subroutines are stored; the stack pointer register (***sp***) is located in IO address 0x02h. The bit number of stack pointer is 8 bit; the stack memory cannot be accessed over 64 bytes and should be defined within 64 bytes from 0x00h address. The stack memory of PMC251 for each FPP unit can be assigned by user via stack pointer register, means that the depth of stack pointer for each FPP unit is adjustable in order to optimize system performance. The following example shows how to define the stack in the ASM (assembly language) project:

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

FPPA1:
        SP =      Stack1;          // assign Stack1 for FPPA1, two level call because of Stack1[2]
 ...
        call    function2
 ...
```

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

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

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

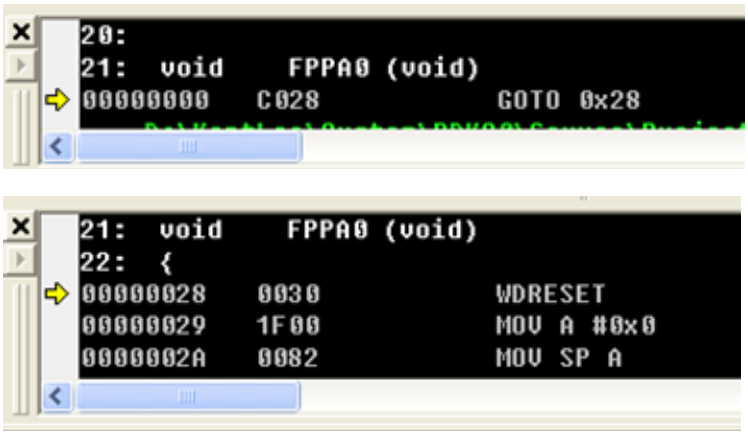


Fig. 2 Stack Assignment in Mini-C project

5.1.3. Single FPP mode

For traditional MCU user who does not need parallel processing capability, PMC251 provides single FPP mode optional to behave as traditional MCU. After single FPP mode is selected, FPP1 is always disabled and only FPP0 is active. Fig.3 shows the timing diagram for each FPP unit, FPP1 is always disabled and only FPP0 active. Please notice that ***wait*** instruction is NOT supported when single FPP mode is chosen.

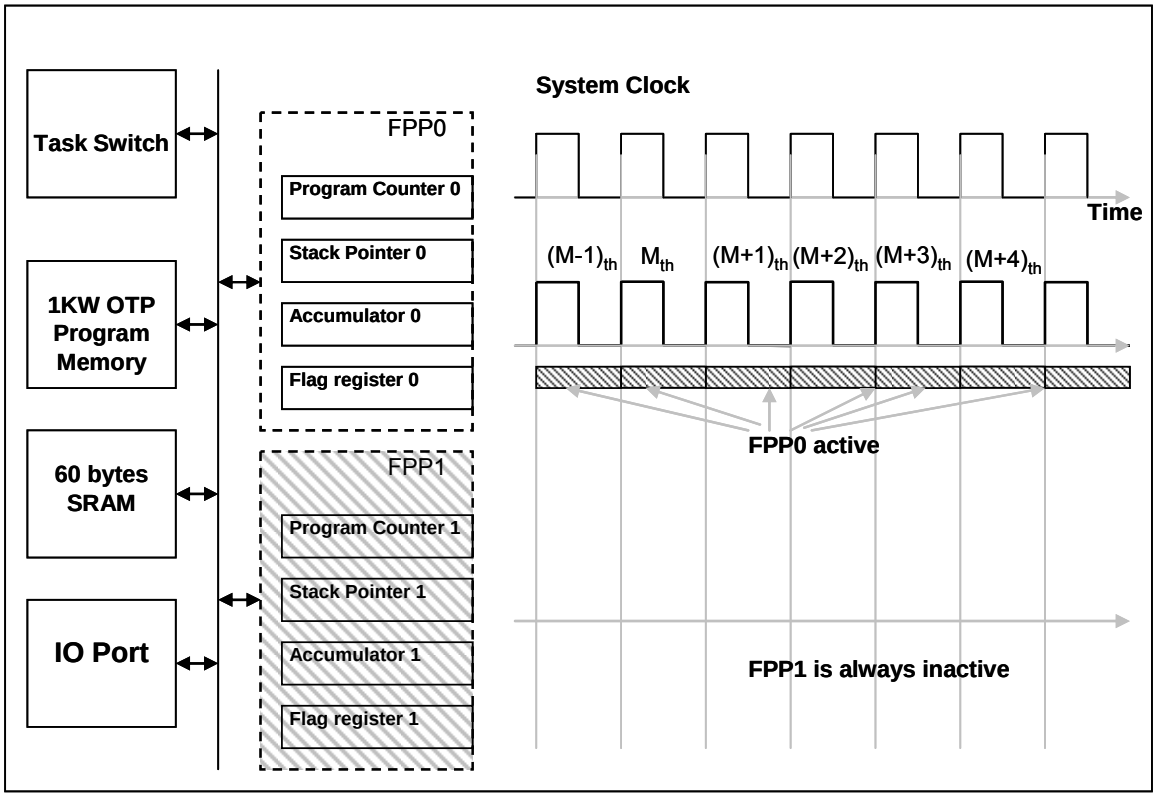


Fig. 3 Timing Diagram of single FPP mode

5.2. Program Memory – OTP

5.2.1. Program Memory Assignment

The OTP (One Time Programmable) program memory is used to store the program instructions to be executed. All program codes for each FPP unit are stored in this OTP memory for both FPP0 and FPP1. The OTP program memory may contains the data, tables and interrupt entry. After reset, the initial address for FPP0 is ‘h0 and ‘h1 for FPP1. The interrupt entry is ‘h10 if used and interrupt function is for FPP0 only, the last eight addresses are reserved for system using, like checksum, serial number, etc. The OTP program memory for PMC251 is a 1Kx16 bit that is partitioned as Table 1. The OTP memory from address ‘h3F8 to ‘h3FF is for system using, address space from ‘h002 to ‘h00F and from ‘h011 to ‘h3F7 are user program space. The address ‘h001 is the FPP1 initial address for two FPP units mode and user program for single FPP unit mode.

Address	Function
‘h000	FPP0 reset – goto instruction
‘h001	FPP1 reset – goto instruction
‘h002	User program
•	•
‘h00F	User program
‘h010	Interrupt entry address
‘h011	User program
•	•
‘h3F7	User program
‘h3F8	System Using
•	•
‘h3FF	System Using

Table 1: Program Memory Organization of PMC251

5.2.2. Example of Using Program Memory for Two FPP mode

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

Address	Function
000	FPP0 reset – goto instruction (goto ‘h020)
001	Begin of FPP1 program
•	•
00F	goto ‘h1A1 to continue FPP1 program
010	Interrupt entry address (FPP0 only)
•	•
01F	End of ISR
020	Begin of FPP0 user program
•	•
1A0	End of FPP0 user program
1A1	Continuing FPP1 program
•	•
3F7	End of FPP1 program
3F8	System Using
•	•
3FF	System Using

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

5.2.3. Example of Using Program Memory for Single FPP mode

The entire user's program memory can be assigned to FPP0 when single FPP mode is selected. Table 3 shows the example of program memory using for single FPP unit mode.

Address	Function
000	FPP0 reset
001	Begin of user program
002	user program
•	•
00F	goto instruction (goto 'h020)
010	Interrupt entry address
011	ISR
•	•
01F	End of ISR
020	user program
•	•
•	•
3F7	user program
3F8	System Using
•	•
3FF	System Using

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

5.3. Program Structure

5.3.1. Program structure of two FPP units mode

After power-up, the program starting address of FPP0 is 0x000 and 0x001 for FPP1. The 0x010 is the entry address of interrupt service routine, which belongs to FPP0 only. The basic firmware structure for PMC251 is shown as Fig. 4, the program codes of two FPP units are placed in one whole program space. Except for the initial addresses of processing units and entry address of interrupt, the memory location is not specially specified; the program codes of processing unit can be resided at any location no matter what the processing unit is. After power-up, the fpp0Boot will be executed first, which will include the system initialization and other FPP units enabled.

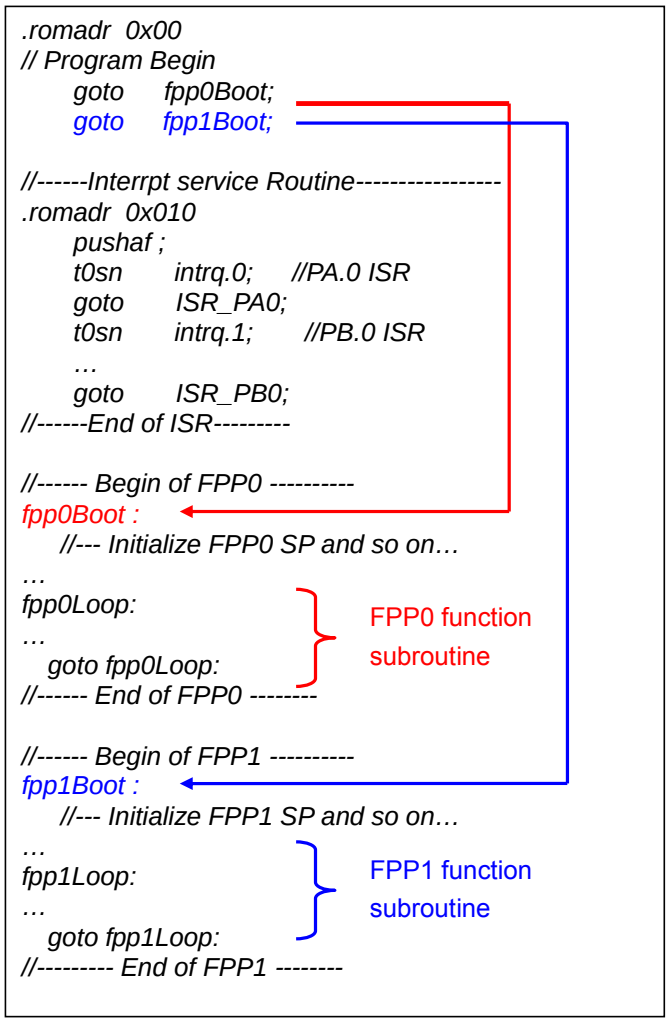


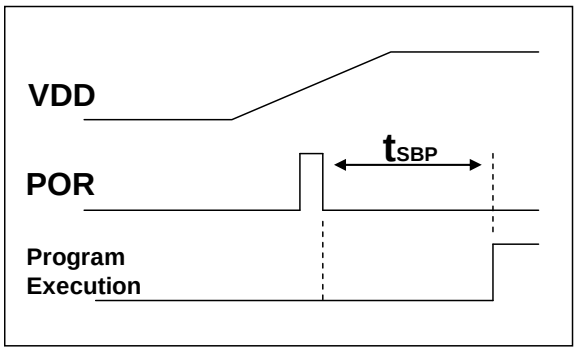
Fig. 4 Program Structure of two FPP units mode

5.3.2. Program structure of single FPP mode

After power-up, the program starting address of FPP0 is 0x000, 0x010 is the entry address of interrupt service routine. For single FPP 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.

5.4. Boot Procedure

POR (Power-On-Reset) is used to reset PMC251 when power up, however, the supply voltage may be not stable. To ensure the stability of supply voltage after power up, it will wait 1024 ILRC clock cycles before first instruction being executed, which is t_{SBP} and shown in the Fig. 5.



Boot up from Power-On Reset

Fig. 5 Power Up Sequence

Fig. 6 shows the typical program flow after boot up. Please notice that the FPP1 is disabled after reset and recommend NOT enabling FPP1 before system and FPP0 initialization.

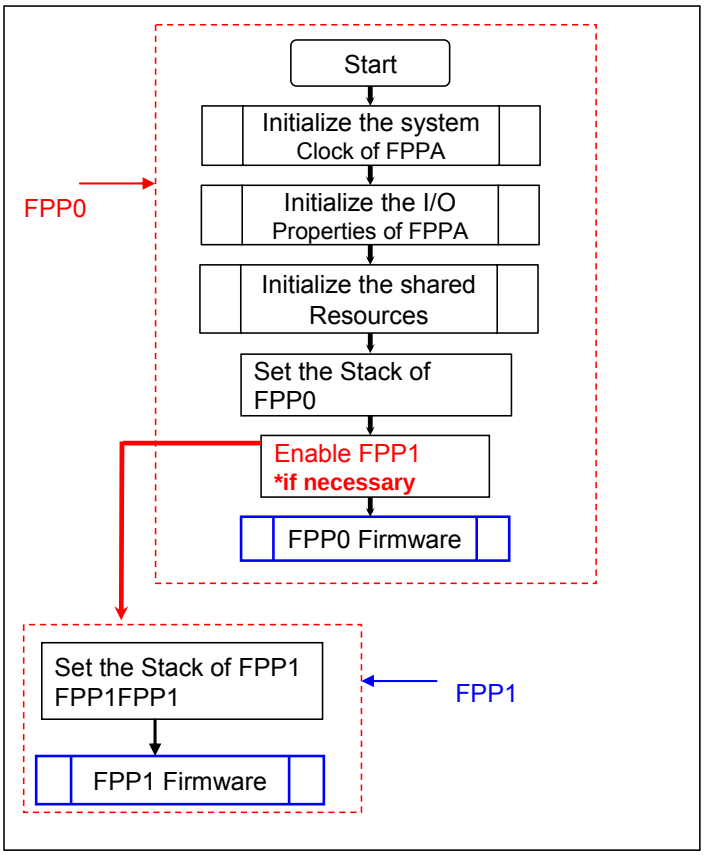


Fig. 6 Boot Procedure

5.5. Data Memory – SRAM

Fig. 7 shows the SRAM data memory organization of PMC251, all the SRAM data memory could be accessed by FPP0 and FPP1 directly with 1T clock cycle, the data access can be byte or bit operation. Besides data storage, the SRAM data memory is also served as data pointer of indirect access method and the stack memory for all FPP units.

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

For indirect memory access mechanism, the data memory is used as the data pointer to address the data byte. All the data memory could be the data pointer; it's quite flexible and useful to do the indirect memory access. All the 60 bytes data memory of PMC251 can be accessed by indirect access mechanism.

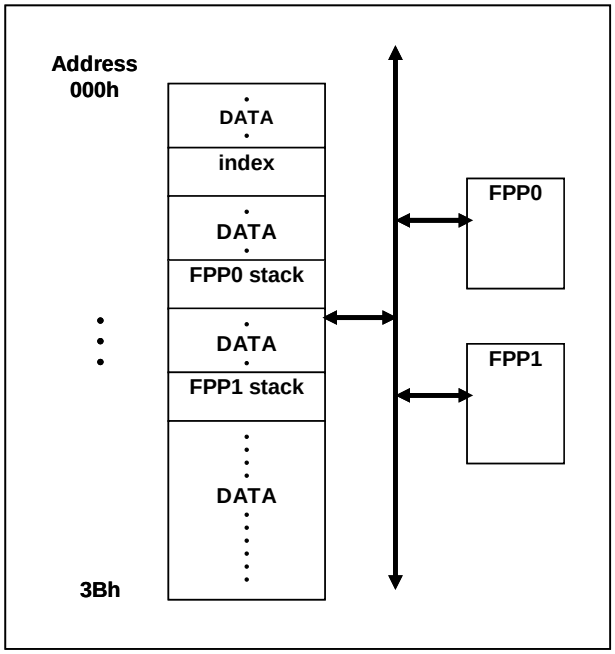


Fig. 7 Data Memory Organization

5.6. Arithmetic and Logic Unit

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

5.7. Oscillator and clock

There are three oscillator circuits provided by PMC251: external crystal oscillator (EOSC), internal high RC oscillator (IHRC) and internal low RC oscillator (ILRC), and these three oscillators are enabled or disabled by registers eoscr.7, clkmd.4 and clkmd.2 independently. User can choose one of these three oscillators as system clock source and use ***clkmd*** register to target the desired frequency as system clock to meet different application.

Oscillator Module	Enable/Disable	Default after boot-up
EOSC	**eoscr**.7	Disabled
IHRC	**clkmd**.4	Enabled
ILRC	**clkmd**.2	Enabled

5.7.1. Internal High RC oscillator and Internal Low RC oscillator

After boot-up, the IHRC and ILRC oscillators are enabled. The frequency of IHRC can be calibrated to eliminate process variation by ***ihrcr*** register; normally it is calibrated to 16MHz. The frequency deviation can be within 2% normally after calibration and it still drifts slightly with supply voltage and operating temperature, the total drift rate is about ±6% for VDD=2.2V~5.5V and -40°C~85°C operating conditions. Please refer to the measurement chart for IHRC frequency verse VDD and IHRC frequency verse temperature. The frequency of ILRC is around 20 KHz, however, its frequency will vary by process, supply voltage and temperature, please refer to DC specification and do not use for accurate timing application.

5.7.2. IHRC calibration

The IHRC frequency may be different chip by chip due to manufacturing variation, PMC251 provide the IHRC frequency calibration to eliminate this variation, and this function can be selected when compiling user's program and the command will be inserted into user's program automatically. The calibration command is shown as below:

.ADJUST_IC SYSCLK=IHRC/(**p1**), IHRC=(**p2**)MHz, VDD=(**p3**)V
Where,
p1=2, 4, 8, 16, 32; In order to provide different system clock.
p2=16 ~ 18; In order to calibrate the chip to different frequency, 16MHz is the usually one.
p3=2.5 ~ 5.5; In order to calibrate the chip under different supply voltage.

5.7.3 IHRC Frequency Calibration and System Clock

During compiling the user program, the options for IHRC calibration and system clock are shown as Table 4:

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

Table 4 Options for IHRC Frequency Calibration

Usually, .ADJUST_IC will be the first command after boot up, in order to set the target operating frequency whenever stating the system. The program code for IHRC frequency calibration is executed only one time that occurs in writing the codes into OTP memory; after then, it will not be executed again. If the different option for IHRC calibration is chosen, the system status is also different after boot. The following shows the status of PMC251 for different option:

(1) .ADJUST_IC SYSCLK=IHRC/2, IHRC=16MHz, VDD=5V
After boot up, CLKMD = 0x34：
- IHRC frequency is calibrated to 16MHz@VDD=5V and IHRC module is enabled
- System CLK = IHRC/2 = 8MHz
- Watchdog timer is disabled, ILRC is enabled, PA5 is in input mode

(2) .ADJUST_IC SYSCLK=IHRC/4, IHRC=16MHz, VDD=3.3V
After boot up, CLKMD = 0x14：
- IHRC frequency is calibrated to 16MHz@VDD=3.3V and IHRC module is enabled
- System CLK = IHRC/4 = 4MHz
- Watchdog timer is disabled, ILRC is enabled, PA5 is in input mode

(3) .ADJUST_IC SYSCLK=IHRC/8, IHRC=16MHz, VDD=2.5V
After boot up, CLKMD = 0x3C：
- IHRC frequency is calibrated to 16MHz@VDD=2.5V and IHRC module is enabled
- System CLK = IHRC/8 = 2MHz
- Watchdog timer is disabled, ILRC is enabled, PA5 is in input mode

(4) .ADJUST_IC SYSCLK=IHRC/16, IHRC=16MHz, VDD=2.5V
After boot up, CLKMD = 0x1C：
- IHRC frequency is calibrated to 16MHz@VDD=2.5V and IHRC module is enabled
- System CLK = IHRC/16 = 1MHz
- Watchdog timer is disabled, ILRC is enabled, PA5 is in input mode

(5) .ADJUST_IC SYSCLK=IHRC/32, IHRC=16MHz, VDD=5V
After boot up, CLKMD = 0x7C：
- IHRC frequency is calibrated to 16MHz@VDD=5V and IHRC module is enabled
- System CLK = IHRC/32 = 500KHz
- Watchdog timer is disabled, ILRC is enabled, PA5 is in input mode

(6) .ADJUST_IC SYSCLK=ILRC, IHRC=16MHz, VDD=5V
After boot up, CLKMD = 0XE4：
- IHRC frequency is calibrated to 16MHz@VDD=5V and IHRC module is disabled
- System CLK = ILRC
- Watchdog timer is enabled, ILRC is enabled, PA5 is input mode

(7) .ADJUST_IC DISABLE
After boot up, CLKMD is not changed (Do nothing)：
- IHRC is not calibrated and IHRC module is disabled
- System CLK = ILRC
- Watchdog timer is enabled, ILRC is enabled, PA5 is in input mode

5.7.4. External Crystal Oscillator

If crystal oscillator is used, a crystal or resonator is required between X1 and X2. Fig. 8 shows the hardware connection under this application; the range of operating frequency of crystal oscillator can be from 32 KHz to 4MHz, depending on the crystal placed on; higher frequency oscillator than 4MHz is NOT supported.

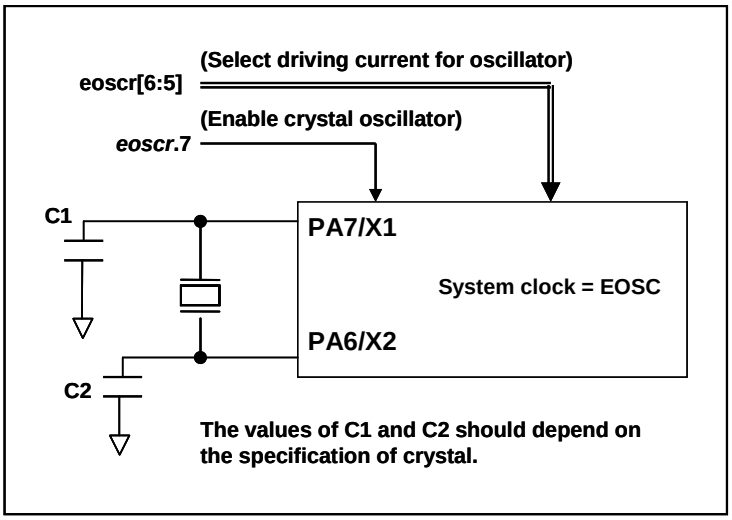


Fig. 8 Connection of crystal oscillator

Besides crystal, external capacitor and options of PMC251 should be fine tuned in *eoscr* (0x0b) register to have good sinusoidal waveform. The *eoscr*.7 is used to enable crystal oscillator module, *eoscr*.6 and *eoscr*.5 are used to set the different driving current to meet the requirement of different frequency of crystal oscillator:

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

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

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

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

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

```
void  FPPA0 (void)
{
    .ADJUST IC  DISABLE       // IHRC is not calibrated, WDT is enabled
    ...
    $ EOSCR Enable, 4Mhz;   // EOSCR = 0b110_00000;
    $ T16M     EOSC, /1, BIT13;   // T16 receive 2^14=16384 clocks of
                                          crystal osc.,
        // Intrq.T16 =>1, crystal osc. is stable
    WORD    count = 0;
    stt16 count;
    Intrq.T16 = 0;
    wait1 Intrq.T16;        // count fm 0x0000 to 0x2000, then set
                      INTRQ.T16
    clkmd = 0xA4;       // switch system clock to EOSC;
    ...
}
```

Please notice that the crystal oscillator should be fully turned off before entering the power-down mode, in order to avoid unexpected wakeup event. If the 32KHz crystal oscillator is used and extremely low operating current is required, *misc*.6 can be set to reduce current after crystal oscillator is running normally.

5.7.5. System Clock and LVD levels

The clock source of system clock comes from EOSC, IHRC and ILRC, the hardware diagram of system clock in the PMC251 is shown as Fig. 9.

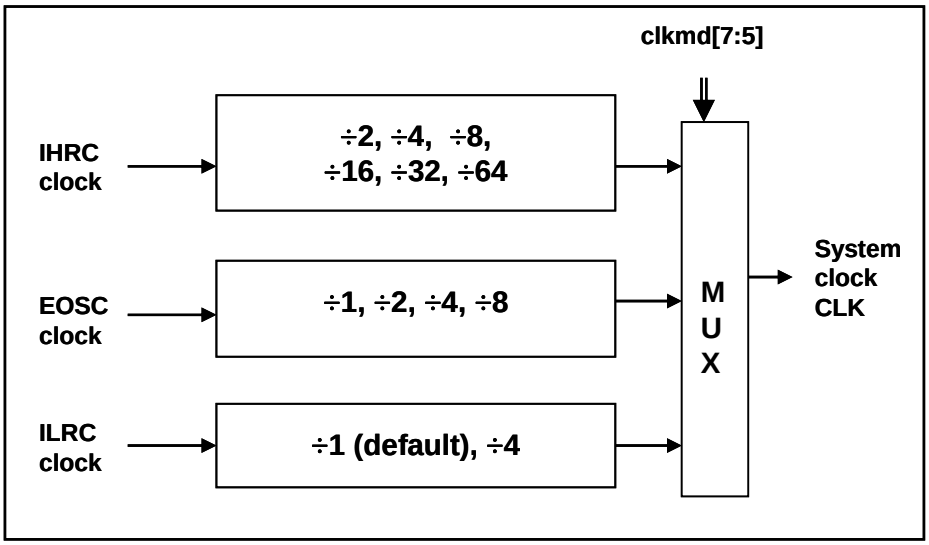


Fig. 9 Options of System Clock

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

- system clock = 8MHz with LVD=3.1V
- system clock = 4MHz with LVD=2.5V
- system clock = 2MHz with LVD=2.2V

5-7-6. System Clock Switching

After IHRC calibration, user may want to switch system clock to a new frequency or may switch system clock at any time to optimize the system performance and power consumption. Basically, the system clock of PMC251 can be switched among IHRC, ILRC and EOSC by setting the ***clkmd*** register at any time; system clock will be the new one after writing to ***clkmd*** register immediately. Please notice that the original clock module can NOT be turned off at the same time as writing command to ***clkmd*** register. The examples are shown as below and more information about clock switching, please refer to the "Help → Application Note → IC Introduction → Register Introduction → CLKMD".

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

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

Case 2: Switching system clock from ILRC to EOSC

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

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

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

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

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

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

```
...
CLKMD      = 0X14 ; //  switch to IHRC/4
...
```

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

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

5.8. 16-bit Timer (Timer16)

PMC251 provides a 16-bit hardware timer (Timer16) and its clock source may come from system clock (CLK), external crystal oscillator (EOSC), internal high RC oscillator (IHRC), internal low RC oscillator (ILRC), PA0 or PA4. Before sending clock to the 16-bit counter, a pre-scaling logic with divided-by-1, 4, 16 or 64 is selectable for wide range counting. The 16-bit counter performs up-counting operation only, the counter initial values can be stored from data memory by issuing the ***stt16*** instruction and the counting values can be loaded to data memory by issuing the ***ldt16*** instruction. The interrupt request from Timer16 will be triggered by the selected bit which comes from bit[15:8] of this 16-bit counter, rising edge or falling edge can be optional chosen by register ***integs.4***. The hardware diagram of Timer16 is shown as Fig. 10.

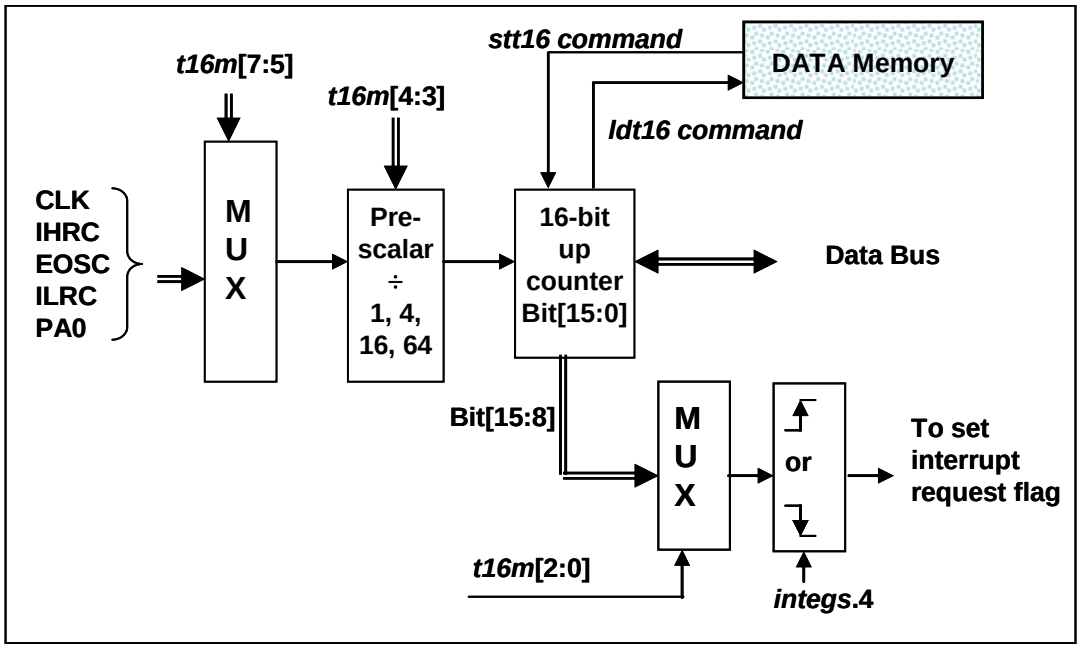


Fig. 10 Hardware diagram of Timer16

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

```
T16M    IO_RW 0x06
$ 7~5:      STOP, SYSCLK, X, X, 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 / 2 = 8 MHz
      // SYSCLK/64 = 8 MHz/64 = 8 us, about every 524 ms to generate INTRQ.2=1
```

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

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

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

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

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

Where

F is the frequency if selected clock source to Timer16

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

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

5.9. Watchdog Timer

The watchdog timer (WDT) is a counter with clock coming from ILRC and its frequency is about 24 KHz. There are four different timeout periods of watchdog timer can be chosen by setting the ***misc*** register, it is:

- 256 ILRC clock period when misc[1:0]=11
- 16384 ILRC clock period when misc[1:0]=10
- 4096 ILRC clock period when misc[1:0]=01
- 2048 ILRC clock period when misc[1:0]=00 (default)

The frequency of ILRC may drift a lot due to the variation of manufacture, supply voltage and temperature; user should reserve guard band for save operation. WDT can be cleared by power-on-reset or by command ***wdreset*** at any time. When WDT is timeout, PMC251 will be reset to restart the program execution. The relative timing diagram of watchdog timer is shown as Fig. 11.

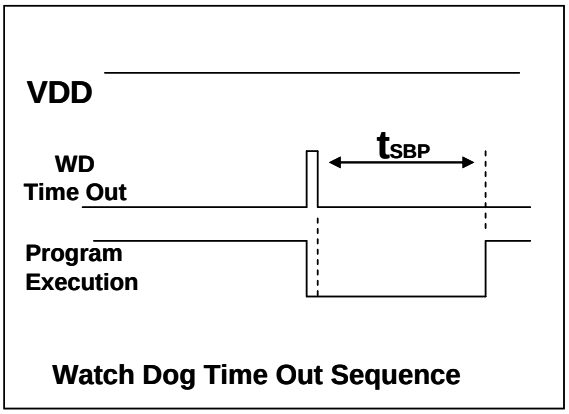


Fig. 11 Sequence of Watch Dog Time Out

5.10. Interrupt

There are three interrupt lines for PMC251:

- External interrupt PA0
- External interrupt PB0
- Timer16 interrupt

Every interrupt request line has its own corresponding interrupt control bit to enable or disable it; the hardware diagram of interrupt function is shown as Fig. 12. All the interrupt request flags are set by hardware and cleared by writing ***intrq*** register. When the request flags are set, it can be rising edge, falling edge or both, depending on the setting of register ***integs***. All the interrupt request lines are also controlled by ***engint*** instruction (enable global interrupt) to enable interrupt operation and ***disgint*** instruction (disable global interrupt) to disable it. Only FPP0 can accept the interrupt request, other FPP unit will not be interfered by interrupt. The stack memory for interrupt is shared with data memory and its address is specified by stack register ***sp.*** Since the program counter is 16 bits width, the bit 0 of stack register ***sp*** should be kept 0. Moreover, user can use ***pushaf*** / ***popaf*** instructions to store or restore the values of ***ACC*** and ***flag*** register ***to*** / ***from*** stack memory.

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

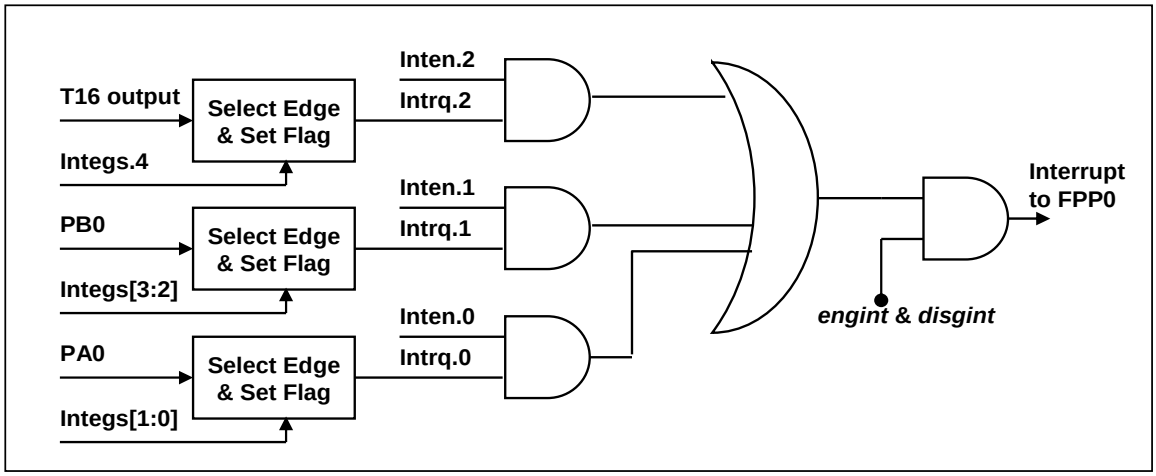


Fig. 12 Hardware diagram of Interrupt controller

Once the interrupt occurs, its operation will be:

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

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

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

The program counter will be restored automatically from the stack memory specified by register ***sp.***

- New ***sp*** will be updated to sp-2.
- Global interrupt will be enabled automatically.
- The next instruction will be the original one before interrupt.

User must reserve enough stack memory for interrupt, two bytes stack memory for one level interrupt and four bytes for two levels interrupt. For interrupt operation, the following sample program shows how to handle the interrupt, noticing that it needs four bytes stack memory to handle 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  (INTRQ.0)
  {             //  Here for PA0 interrupt service routine
    INTRQ.0  =  0;
    ...
    }
    ...
  POPAF         //  restore ALU and FLAG register
 }
```

5.11. Power-Save and Power-Down

There are three operational modes defined by hardware: ON mode, Power-Save mode and Power-Down modes. ON mode is the state of normal operation with all functions ON, Power-save mode ("***stopexe***") is the state to reduce operating current and CPU keeps ready to continue, Power-Down mode ("***stopsys***") is used to save power deeply. Therefore, Power-save mode is used in the system which needs low operating power with wake-up occasionally and Power-Down mode is used in the system which needs power down deeply with seldom wake-up. Fig. 13 shows the differences in oscillator modules between Power-Save mode ("***stopexe***") and Power-Down mode ("***stopsys***").

Differences in oscillator modules between STOPSYS and STOPEXE			
	IHRC	ILRC	EOSC
STOPSYS	Stop	Stop	Stop
STOPEXE	No Change	No Change	No Change

Fig. 13 Differences in oscillator modules between STOPSYS and STOPEXE

5-11-1. Power-Save mode ("*stopexe*")

Using "***stopexe***" instruction to enter the Power-Save mode, only system clock is disabled, remaining all the oscillator modules be active. For CPU, it stops executing; however, for Timer16, counter keep counting if its clock source is not the system clock. The wake-up sources for "***stopexe***" can be IO-toggle or Timer16 counts to set values when the clock source of Timer16 is IHRC, ILRC or EOSC modules. Wake-up from input pins can be considered as a continuation of normal execution, ***nop*** command is recommended to follow the ***stopexe*** command, the detail information for Power-Save mode shown below:

- IHRC, ILRC and EOSC oscillator modules: No change, keep active if it was enabled
- System clock: Disable, therefore, CPU stops execution
- OTP memory is turned off
- Timer16: Stop counting if system clock is selected or the corresponding oscillator module is disabled; otherwise, it keeps counting.
- Wake-up sources: IO toggle or Timer16.

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

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

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

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

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

5-11-2. 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. The internal low frequency RC oscillator must be enabled before entering the Power-Down mode, means that bit 2 of register ***clkmd*** (0x03) must be set to high before issuing "***stopsys***" command in order to resume the system when wakeup. The following shows the internal status of PMC251 in detail when "***stopsys***" command is issued:

- All the oscillator modules are turned off
- Enable internal low RC oscillator (set bit 2 of register clkmd)
- OTP memory is turned off
- The contents of SRAM and registers remain unchanged
- Wake-up sources: ANY IO toggle.
- If PA or PB is input mode and set to analog input by padier or pbdier register, it can NOT be used to
- Wake-up the system.

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

```
CLKMD   = 0xF4;  // Change clock from IHRC to ILRC, disable watchdog timer
CLKMD.4 = 0;     // disable IHRC
...
while (1)
  {
    STOPSYS;         // enter power-down
    if (...) break;  // if wakeup happen and check OK, then return to high speed,
                     // else stay in power-down mode again.
  }
  CLKMD = 0x34;      // Change clock from ILRC to IHRC/2
```

5-11-3. Wake-up

After entering the Power-Down or Power-Save modes, the PMC251 can be resumed to normal operation by toggling IO pins, Timer16 interrupt is available for Power-Save mode ONLY. Fig. 14 shows the differences in wake-up sources between STOPSYS and STOPEXE.

Differences in wake-up sources between STOPSYS and STOPEXE		
	IO Toggle	T16 Interrupt
STOPSYS	Yes	No
STOPEXE	Yes	Yes

Fig. 14 Differences in wake-up sources between Power-Save mode and Power-Down mode

When using the IO pins to wake-up the PMC251, registers ***padier*** and ***pbdier*** should be properly set to enable the wake-up function for every corresponding pin. The wake-up time for normal wake-up is about 1024 ILRC clocks counting from wake-up event; fast wake-up can be selected to reduce the wake-up time by ***misc*** register. For fast wake-up mechanism, the wake-up time is 128 system clocks from IO toggling if STOPEXE was issued, and 128 system clocks plus oscillator (IHRC or ILRC) stable time from IO toggling if STOPSYS was issued. The oscillator stable time is the time for IHRC or ILRC oscillator from power-on, depending on which oscillator is used as system clock source. Please notice that the fast wake-up will turn off automatically when EOSC is selected as system clock.

Suspend mode	Wake-up mode	System clock source	Wake-up time (t_{WUP}) from IO toggle
STOPEXE suspend	fast wake-up	IHRC/ILRC	128 * $T_{SYS,}$ Where T_{SYS} is the time period of system clock
STOPSYS suspend	fast wake-up	IHRC	128 T_{SYS} + T_{SIHRC}; Where T_{SIHRC} is the stable time of IHRC from power-on. T_{SIHRC} = 5us@5V
STOPSYS suspend	fast wake-up	ILRC	128 T_{SYS} + T_{SILRC}; Where T_{SILRC} is the stable time of ILRC from power-on. T_{SILRC} = 43ms@5V
STOPSYS or STOPEXE suspend	fast wake-up	EOSC	1024 * T_{ILRC}, Where T_{ILRC} is the clock period of ILRC
STOPEXE suspend	normal wake-up	Any one	1024 * T_{ILRC}, Where T_{ILRC} is the clock period of ILRC, T_{SILRC} = 43ms@5V
Suspend mode	wake-up mode	system clock source	wake-up time (t_{WUP}) from IO toggle

Fig. 15 Differences in wake-up time from IO toggle

5.12. IO Pins

Other than PA5, all the pins can be independently set into two states output or input by configuring the data registers (***pa, pb***), control registers (***pac, pbc***) and pull-high registers (***paph, pbph).*** All these pins have Schmitt-trigger input buffer and output driver with CMOS level. When it is set to output low, the pull-up resistor is turned off automatically. If user wants to read the pin state, please notice that it should be set to input mode before reading the data port; if user reads the data port when it is set to output mode, the reading data comes from data register, NOT from IO pad. As an example, Table 6 shows the configuration table of bit 0 of port A. The hardware diagram of IO buffer is also shown as Fig. 16.

pa.0	*pac.0*	*paph.0*	Description
X	0	0	Input without pull-up resistor
X	0	1	Input with pull-up resistor
0	1	X	Output low without pull-up resistor
1	1	0	Output high without pull-up resistor
1	1	1	Output high with pull-up resistor

Table 6 PA0 Configuration Table

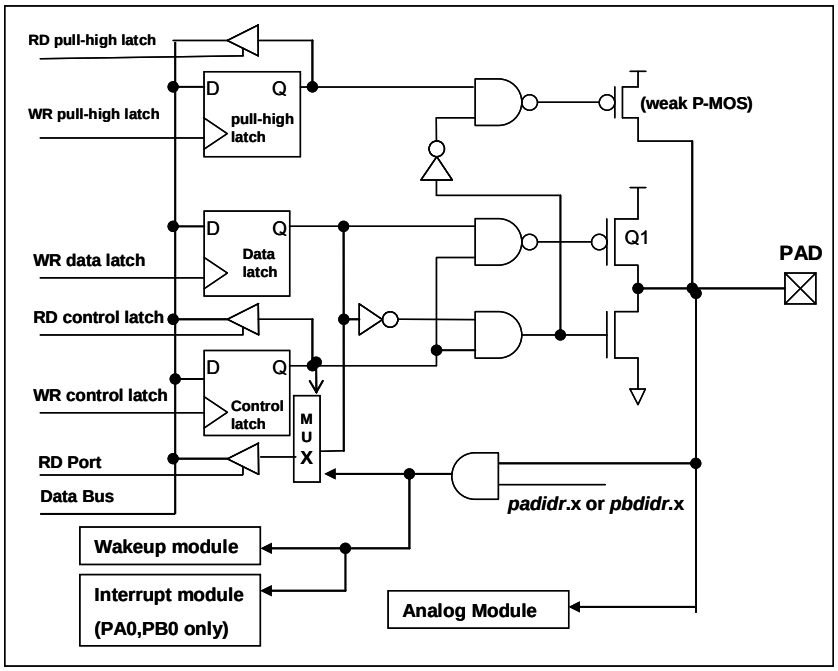


Fig. 16 Hardware diagram of IO buffer

Other than PA5, all the IO pins have the same structure; PA5 can is open-drain ONLY when setting to output mode (without Q1). The corresponding bits in registers ***padier*** / ***pbdier*** should be set to low to prevent leakage current for those pins are selected to be analog function. When PMC251 is put in power-down or power-save mode, every pin can be used to wake-up system by toggling its state. Therefore, those pins needed to wake-up system must be set to input mode and set the corresponding bits of registers ***padier*** and ***pbdier*** to high***.*** The same reason, ***padier***.0 should be set high when PA0 is used as external interrupt pin and ***pbdier***.0 for PB0.

5.13. Reset and LVD

5.13.1. Reset

There are many causes to reset the PMC251, once reset is asserted, most of all the registers in PMC251 will be set to default values, only ***gdio*** register (IO address 0x7) keeps the same value, system should be restarted once abnormal cases happen, or by jumping program counter to address 'h0. The data memory is in uncertain state when reset comes from power-up and LVD; however, the content will be kept when reset comes from PRST# pin or WDT timeout.

5.13.2. LVD reset

By code option, there are 8 different levels of LVD for reset ~ 4.1V, 3.6V, 3.1V, 2.8V, 2.5V, 2.2V, 2.0V and 1.8V; usually, user selects LVD reset level to be in conjunction with operating frequency and supply voltage.

6. IO Registers

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

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

6.2. FPP unit Enable Register (*fppen*), IO address = 0x01

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

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

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

6.4. Clock Mode Register (*clkmd*), IO 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: reserved 011: IHRC/32 100: IHRC/64 101: EOSC/8 11x: reserved.
4	1	R/W	IHRC oscillator Enable. 0 / 1: disable / enable	
3	0	RW	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/PRST# function. 0 / 1: PA5 / PRST#.	

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

Bit	Reset	R/W	Description
7–3	-	R/W	Reserved.
2	-	R/W	Enable interrupt from Timer16 overflow. 0 / 1: disable / enable.
1	-	R/W	Enable interrupt from PB0. 0 / 1: disable / enable.
0	-	R/W	Enable interrupt from PA0. 0 / 1: disable / enable.

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

Bit	Reset	R/W	Description
7–3	-	R/W	Reserved.
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, this bit is set by hardware and cleared by software. 0 / 1: No request / Request
0	-	R/W	Interrupt Request from pin PA0, this bit is set by hardware and cleared by software. 0 / 1: No request / Request

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

Bit	Reset	R/W	Description
7–5	000	R/W	Timer Clock source selection 000: Timer 16 is disabled 001: CLK (system clock) 010: reserved 011: reserved 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 goes high. 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

6.8. General Data register for IO (*gdio*), IO address = 0x07

Bit	Reset	R/W	Description
7–0	00	R/W	General data for IO. This port is the general data buffer in IO space and cleared only when POR, LVD or pin PRST# is active, NOT cleared by watch-dog timeout reset. It can perform the IO operation, like ***wait0*** gdio.x, ***wait1*** gdio.x and ***tog*** gdio.x to take the replace of operations which instructions are supported in memory space (ex: ***wait1*** mem; ***wait0*** mem; *tog* mem).

6.9. External Oscillator setting Register (*eoscr, write only*), IO address = 0x0a

Bit	Reset	R/W	Description
7	0	WO	Enable 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
4–1	-	-	Reserved. Please keep 0.
0	0	WO	Power-down the Band-gap and LVD hardware modules. 0 / 1: normal / power-down.

6.10. IHRC oscillator control Register (*ihrcr, write only*), IO address = 0x0b

Bit	Reset	R/W	Description
7–0	--	WO	Bit [7:0] of internal high RC oscillator for frequency calibration. For system using only, please user do NOT write this register.

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

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

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

Bit	Reset	R/W	Description
7	1	WO	Enable PA7 digital input and wake up event. 1 / 0 : enable / disable. This bit should be set to high to prevent leakage current when external crystal oscillator is used. If this bit is set to low, PA7 can NOT be used to wake up the system. Note: For ICE emulation, the function is disabled when this bit is “1” and “0” is enabled.
6	1	WO	Enable PA6 digital input and wake up event. 1 / 0 : enable / disable. This bit should be set to high to prevent leakage current when external crystal oscillator is used. If this bit is set to low, PA6 can NOT be used to wake up the system. Note: For ICE emulation, the function is disabled when this bit is “1” and “0” is enabled.
5	1	WO	Enable PA5 wake up event. 1 / 0 : enable / disable. This bit can be set to low to disable wake up from PA5 toggling. Note: For ICE emulation, the function is disabled when this bit is “1” and “0” is enabled.
4	1	WO	Enable PA4 wake up event. 1 / 0 : enable / disable. If this bit is set to low, PA4 can NOT be used to wake up the system. Note: For ICE emulation, the function is disabled when this bit is “1” and “0” is enabled.
3	1	WO	Enable PA3 wake up event. 1 / 0 : enable / disable. If this bit is set to low, PA3 can NOT be used to wake up the system. Note: For ICE emulation, the function is disabled when this bit is “1” and “0” is enabled.
2 – 1	-	-	Reserved.
0	1	WO	Enable PA0 wake up event and interrupt request. 1 / 0 : enable / disable. This bit can be set to low to disable wake up from PA0 toggling and interrupt request from this pin. Note: For ICE emulation, the function is disabled when this bit is “1” and “0” is enabled.

Note: Due to the controlling polarity of this register is different between ICE and real chip. In order to unify the program for both ICE emulation and real chip to be the same one, please use the following command to write this register:

“$ PADIER 0xhh” ;

For example:

$ PADIER 0xF0;

It is used to enable the digital input and wakeup function of bit [7:4] of port A for both ICE and real chip, IDE will handle the difference between ICE and real chip automatically.

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

Bit	Reset	R/W	Description
7	1	WO	Enable PB7 wake up event. 1 / 0 : enable / disable. If this bit is set to low, PB7 can NOT be used to wake up the system. Note: For ICE emulation, the function is disabled when this bit is “1” and “0” is enabled.
6	1	WO	Enable PB6 wake up event. 1 / 0 : enable / disable. This bit can be set to low to disable wake up from PB6 toggling. Note: For ICE emulation, the function is disabled when this bit is “1” and “0” is enabled.
5	1	WO	Enable PB5 wake up event. 1 / 0 : enable / disable. This bit can be set to low to disable wake up from PB5 toggling. Note: For ICE emulation, the function is disabled when this bit is “1” and “0” is enabled.
4 – 3	-	-	Reserved.
2	1	WO	Enable PB2 wake up event. 1 / 0 : enable / disable. If this bit is set to low, PB2 can NOT be used to wake up the system. Note: For ICE emulation, the function is disabled when this bit is “1” and “0” is enabled.
1	1	WO	Enable PB1 wake up event. 1 / 0 : enable / disable. If this bit is set to low, PB1 can NOT be used to wake up the system. Note: For ICE emulation, the function is disabled when this bit is “1” and “0” is enabled.
0	1	WO	Enable PB0 wake up event and interrupt request. 1 / 0 : enable / disable. If this bit is set to low, PB0 can NOT be used to wake up the system and interrupt request from this pin. Note: For ICE emulation, the function is disabled when this bit is “1” and “0” is enabled.

Note: Due to the controlling polarity of this register is different between ICE and real chip. In order to unify the program for both ICE emulation and real chip to be the same one, please use the following command to write this register:

“$ PBDIER 0xhh” ;

For example:

$ PBDIER 0xF0;

It is used to enable the digital input and wakeup function of bit [7:4] of port B for both ICE and real chip, IDE will handle the difference between ICE and real chip automatically.

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

Bit	Reset	R/W	Description
7–0	8'h00	R/W	Data registers for Port A.

6.15. Port A Control Registers (*pac*), IO address = 0x11

Bit	Reset	R/W	Description
7–0	8'h00	R/W	Port A control registers. This register is used to define input mode or output mode for each corresponding pin of port A. 0 / 1: input / output.

6.16. Port A Pull-High Registers (*paph*), IO address = 0x12

Bit	Reset	R/W	Description
7–0	8'h00	R/W	Port A pull-high registers. This register is used to enable the internal pull-high device on each corresponding pin of port A. 0 / 1 : disable / enable Please note that the PA5 does not have pull-up resistor.

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

Bit	Reset	R/W	Description
7–0	8'h00	R/W	Data registers for Port B.

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

Bit	Reset	R/W	Description
7–0	8'h00	R/W	Port B control registers. This register is used to define input mode or output mode for each corresponding pin of port B. 0 / 1: input / output

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

Bit	Reset	R/W	Description
7–0	8'h00	R/W	Port B pull-high registers. This register is used to enable the internal pull-high device on each corresponding pin of port B. 0 / 1 : disable / enable

6.20. MISC Register (*misc*), IO address = 0x3b

Bit	Reset	R/W	Description
7	0	-	Reserved
6	0	WO	Enable extremely low current for 32KHz crystal oscillator AFTER oscillation. 0: Normal. 1: Low driving current for 32KHz crystal oscillator.
5	0	WO	Enable fast Wake up. 0: Normal wake up. The wake-up time is 1024 ILRC clocks 1: Fast wake up. The wake-up time is 128 CLKs (system clock) if IHRC is used and 1024 CLKs (system clock) + crystal oscillator stable time if crystal oscillator is used.
4	-	-	Reserved
3	0	WO	Recover time from LVD reset. 0: Normal. The system will take about 1024 ILRC clocks to boot up from LVD reset. 1: Fast. The system will take about 64 ILRC clocks to boot up from LVD reset. Please notice that the clock source will be switched to system clock (for example: 4MHz) when fast wakeup is enabled, therefore, it is recommended to turn off the watchdog timer **before** enabling the fast wakeup and turn on the watchdog timer **after** disabling the fast wakeup.
2	0	WO	Disable LVD function. 0 / 1 : Enable / Disable
1 – 0	00	WO	Watch dog time out period 00: 2048 ILRC clock period 01: 4096 ILRC clock period 10: 16384 ILRC clock period 11: 256 ILRC clock period

7. Instructions

Symbol	Description
ACC	Accumulator (Abbreviation of accumulator)
a	Accumulator (symbol of accumulator in program)
sp	Stack pointer
flag	ACC status flag register
I	Immediate data
&	Logical AND
|	Logical OR
←	Movement
^	Exclusive logic OR
+	Add
−	Subtraction
~	NOT (logical complement, 1's complement)
∓	NEG (2's complement)
OV	Overflow (The operational result is out of range in signed 2's complement number system)
Z	Zero (If the result of ALU operation is zero, this bit is set to 1)
C	Carry (The operational result is to have carry out for addition or to borrow carry for subtraction in unsigned number system)
AC	Auxiliary Carry (If there is a carry out from low nibble after the result of ALU operation, this bit is set to 1)
pc0	Program counter for FPP0
pc1	Program counter for FPP1

7.1. 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 Timer 16
	ldt16 T16val ; // save the T16 counting value to T16val
	….
	--- 	

stt16 **word**	Store 16-bit data from memory in word to Timer16.
	Example: *stt16* word;
	Result: 16-bit timer ←word
	Affected flags: 『N』Z 『N』C 『N』AC 『N』OV
	
	Application Example:
	--- 	
	word T16val ; // declare a RAM word
	…
	mov a, 0x34 ;
	mov lb@ T16val , a ; // move 0x34 to T16val (LSB)
	mov a, 0x12 ;
	mov hb@ T16val , a ; // move 0x12 to T16val (MSB)
	stt16 T16val ; // initial T16 with 0x1234
	…
	--- 	

idxm **a, index**	Move data from specified memory to ACC by indirect method. It needs 2T to execute this instruction.
	Example: *idxm* a, index;
	Result: a ← [index], where index is declared by word.
	Affected flags: 『N』Z 『N』C 『N』AC 『N』OV
	
	Application Example:
	--- 	
	word RAMIndex ; // declare a RAM pointer
	…
	mov a, 0x5B ; // assign pointer to an address (LSB)
	mov lb@RAMIndex, a ; // save pointer to RAM (LSB)
	mov a, 0x00 ; // assign 0x00 to an address (MSB), should be 0
	mov hb@RAMIndex, a ; // save pointer to RAM (MSB)
	…
	idxm a, RAMIndex ; // mov memory data in address 0x5B to ACC
	--- 	

Idxm **index, a**	Move data from ACC to specified memory by indirect method. It needs 2T to execute this instruction.
	Example: *idxm* index, a;
	Result: [index] ← a; where index is declared by word.
	Affected flags: 『N』Z 『N』C 『N』AC 『N』OV
	
	Application Example:
	--- 	
	word RAMIndex ; // declare a RAM pointer
	…
	mov a, 0x5B ; // assign pointer to an address (LSB)
	mov lb@RAMIndex, a ; // save pointer to RAM (LSB)
	mov a, 0x00 ; // assign 0x00 to an address (MSB), should be 0
	mov hb@RAMIndex, a ; // save pointer to RAM (MSB)
	…
	mov a, 0xA5 ;
	idxm RAMIndex, a ; // mov 0xA5 to memory in address 0x5B
	--- 	

xch **M**	Exchange data between ACC and memory Example: *xch* MEM ; Result: MEM ← a , a ← MEM Affected flags: 『N』Z 『N』C 『N』AC 『N』OV

pushaf	Move the *ACC* and *flag* register to memory that address specified in the stack pointer.
	Example: *pushaf*;
	Result: [sp] ← {flag, ACC};
	sp ← sp + 2 ;
	Affected flags: 『N』Z 『N』C 『N』AC 『N』OV
	
	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

7.2. 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

7.3. 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

7.4. 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, 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

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

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

7.5. Bit Operation Instructions

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

7.6. Conditional Operation Instructions

ceqsn **a, I**	Compare ACC with immediate data and skip next instruction if both are equal. Flag will be changed like as (a ← a - I) Example: *ceqsn* a, 0x55 ; *inc* MEM ; *goto* error ; Result: If a=0x55, then “goto error”; otherwise, “inc MEM”. Affected flags: 『Y』Z 『Y』C 『Y』AC 『Y』OV
ceqsn **a, M**	Compare ACC with memory and skip next instruction if both are equal. 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 **M, a**	Compare memory with ACC and skip next instruction if both are not equal. Flag will be changed like as (M ← M - a) Example: *cneqsn MEM, a;* 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**	Go next instruction until bit n of IO port is low, otherwise, wait here. 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 **Note:** This instruction is not supported in single FPP mode.
wait1 **IO.n**	Go next instruction until bit n of IO port is high, otherwise, wait here. 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 **Note:** This instruction is not supported in single FPP mode.

7.7. 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
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

7.8. Summary of Instructions Execution Cycle

2T	*jump, call, ceqsn, cneqsn, t0sn, t1sn, dzsn, izsn, ldtabh, ldtabl, idxm*
1T	Others

7.9. Summary of affected flags by Instructions

Instruction	Z	C	AC	OV	Instruction	Z	C	AC	OV	Instruction	Z	C	AC	OV
mov a, I	-	-	-	-	*mov* M, a	-	-	-	-	*mov* a, M	Y	-	-	-
mov a, IO	Y	-	-	-	*mov* IO, a	-	-	-	-	*nmov* M, a	-	-	-	-
nmov a, M	Y	-	-	-	*ldtabh* index	-	-	-	-	*ldtabl* index	-	-	-	-
ldt16 index	-	-	-	-	*stt16* index					*xch* M	-	-	-	-
idxm a, index	-	-	-	-	*idxm* index, a	-	-	-	-	*wdreset*	-	-	-	-
pushaf	-	-	-	-	*popaf*	Y	Y	Y	Y	*add* a, I	Y	Y	Y	Y
add a, M	Y	Y	Y	Y	*add* M, a	Y	Y	Y	Y	*addc* a, M	Y	Y	Y	Y
addc M, a	Y	Y	Y	Y	*addc* a	Y	Y	Y	Y	*addc* M	Y	Y	Y	Y
nadd a, M	Y	Y	Y	Y	*nadd* M, a	Y	Y	Y	Y	*sub* a, I	Y	Y	Y	Y
sub a, M	Y	Y	Y	Y	*sub* M, a	Y	Y	Y	Y	*subc* a, M	Y	Y	Y	Y
subc M, a	Y	Y	Y	Y	*subc* a	Y	Y	Y	Y	*subc* M	Y	Y	Y	Y
inc M	Y	Y	Y	Y	*dec* M	Y	Y	Y	Y	*clear* M	-	-	-	-
sr a	-	Y	-	-	*src* a	-	Y	-	-	*sr* M	-	Y	-	-
src M	-	Y	-	-	*sl* a	-	Y	-	-	*slc* a	-	Y	-	-
sl M	-	Y	-	-	*slc* M	-	Y	-	-	*swap* a	-	-	-	-
swap M	-	-	-	-	*and* a, I	Y	-	-	-	*and* a, M	Y	-	-	-
and M, a	Y	-	-	-	*or* a, I	Y	-	-	-	*or* a, M	Y	-	-	-
or M, a	Y	-	-	-	*xor* a, I	Y	-	-	-	*xor* a, M	Y	-	-	-
xor M, a	Y	-	-	-	*xor* a, IO	Y	-	-	-	*xor* IO, a	-	-	-	-
not a	Y	-	-	-	*not* M	Y	-	-	-					
neg a	Y	-	-	-	*neg* M	Y	-	-	-	*comp* a, I	Y	Y	Y	Y
comp a, M	Y	Y	Y	Y	*comp* M, a	Y	Y	Y	Y	*set0* IO.n	-	-	-	-
set1 IO.n	-	-	-	-	*tog* IO.n	-	-	-	-	*set0* M.n	-	-	-	-
set1 M.n	-	-	-	-	*swapc* IO.n	-	Y	-	-	*ceqsn* a, I	Y	Y	Y	Y
ceqsn a, M	Y	Y	Y	Y	*ceqsn* M, a	Y	Y	Y	Y	*cneqsn* a, I	Y	Y	Y	Y
cneqsn a, M	Y	Y	Y	Y	*cneqsn* M, a	Y	Y	Y	Y	*t0sn* IO.n	-	-	-	-
t1sn IO.n	-	-	-	-	*t0sn* M.n	-	-	-	-	*t1sn* M.n	-	-	-	-
izsn a	Y	Y	Y	Y	*dzsn* a	Y	Y	Y	Y	*izsn* M	Y	Y	Y	Y
dzsn M	Y	Y	Y	Y	*wait0* IO.n	-	-	-	-	*wait1* IO.n	-	-	-	-
call label	-	-	-	-	*goto* label	-	-	-	-	*ret* I	-	-	-	-
ret					*reti*	-	-	-	-	*nop*	-	-	-	-
pcadd a					*engint*	-	-	-	-	*disgint*	-	-	-	-
stopsys					*reset*	-	-	-	-	*stopexe*	-	-	-	-

8. Special Notes

This chapter is to remind user who use PMC251 series IC in order to avoid frequent errors upon operation.

8.1. Using IC

8.1.1. IO pin usage and setting

(1) IO pin as analog input

- Configure IO pin as input
- Set PADIER and PBDIER registers to configure corresponding IO as analog input
- Set PAPH and PBPH registers to disable corresponding IO pull-up resistor
- The functions of PADIER and PBDIER registers of PMC251 series IC is contrary to ICE functions. Please use following program in order to keep ICE emulation consisting with PMC251 series IC procedure.

```
$ PADIER 0xF0;
$ PBDIER 0x07;
```

(2) PA5 as input pin

PA5 can only be Open-Drain output pin. Output high requires adding pull-up resistor

(3) PA5 as PRST# input

- No internal pull-up resistor for PA5
- Configure PA5 as input
- Set CLKMD.0=1 to enable PA5 as PRST# input pin

(4) PA4/PA5 as input pin to connect with a push button or a switch by a long wire

- Needs to put a >10Ω resistor in between PA4/PA5 and the long wire
- Avoid using PA5 as input

(5) PA7 and PA6 as external crystal oscillator pin

- Configure PA7 and PA6 as input
- Disable PA7 and PA6 internal pull-up resistor
- Configure PADIER register to set PA6 and PA7 as analog input
- EOSCR register bit [6:5] selects corresponding crystal oscillator frequency :
  - ✧ 01 : for lower frequency, ex : 32kHz
  - ✧ 10 : for middle frequency, ex : 455kHz、1MHz
  - ✧ 11 : for higher frequency, ex : 4MHz
- Program EOSCR.7 =1 to enable crystal oscillator
- Ensure EOSC working well before switching from IHRC or ILRC to EOSC. Refer to 8.1.3.(2)

8.1.2. Interrupt

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

Step1: Set INTEN register, enable the interrupt control bit

Step2: Clear INTRQ register

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

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

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

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

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

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

(3) FPPA1 will not be affected by interrupt at all

8.1.3. System clock switching

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

◆ Case 1 : Switch system clock from ILRC to IHRC/2

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

◆ Case 2 : Switch system clock from ILRC to EOSC

```
CLKMD   =   0xA6;          // switch to EOSC, ILRC can not be disabled here
CLKMD.2 =   0;             // ILRC can be disabled at this time
```

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

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

(2) Please ensure the EOSC oscillation has established before switching from ILRC or IHRC to EOSC. MCU will not check its status. Please wait for a while after enabling EOSC. System clock can be switched to EOSC afterwards. Otherwise, MCU will hang. The example for switching system clock from ILRC to 4MHz EOSC after boot up is as below:

```
.ADJUST_IC     DISABLE
CLKMD.1 = 0;                        // turn off WDT for executing delay instruction.
$   EOSCR      Enable, 4MHz;        // 4MHz EOSC start to oscillate.
// Delay for EOSC establishment
$ T16M EOSC,/1,BIT10
Word Count = 0;
Stt16 Count;
Intrq.T16 = 0;
wait1       Intrq.T16;
CLKMD   = 0xA4;             // ILRC -> EOSC;
CLKMD.2 = 0;                // turn off ILRC only if necessary
```

The delay duration should be adjusted in accordance with the characteristic of the crystal and PCB. To measure the oscillator signal by the oscilloscope, please select (x10) on the probe and measure through PA6(X2) pin to avoid the interference on the oscillator.

8.1.4. Power down mode, wakeup and watchdog

(1) Watchdog will be inactive once ILRC is disabled

(2) Please turn off watchdog before executing STOPSYS or STOPEXE instruction, otherwise IC will be reset due to watchdog timeout. It is the same as in ICE emulation.

(3) The clock source of Watchdog is ILRC if the fast wakeup is disabled; otherwise, the clock source of Watchdog will be the system clock and the reset time becomes much shorter. It is recommended to disable Watchdog and enable fast wakeup before entering STOPSYS mode. When the system is waken up from power down mode, please firstly disable fast wakeup function, and then enable Watchdog. It is to avoid system to be reset after being waken up.

(4) If enable Watchdog during programming and also wants the fast wakeup, the example as below:

```
CLKMD.En_WatchDog     =     0;          // disable watchdog timer
$ MISC     Fast_Wake_Up;
stopexe;
nop;
$ MISC     WT_xx;                       // Reset Watchdog time to normal wake-up
Wdreset;
CLKMD.En_WatchDog     =     1;          // enable watchdog timer
```

8.1.5. TIMER time out

When select T16M counter BIT8 as 1 to generate interrupt, 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.

8.1.6. Delay instruction

PMC251 does not support delay instruction. Please use program loop to perform delay function. Please notice that the conditional branch instructions (ex. ceqsn…etc) will consume different instruction cycle in accordance with the conditions fulfilled or not. Please refer to the table at topic 8 to calculate the total delay period.

8.1.7. LVR

(1) VDD must reach or above 2.2V for successful power-on process; otherwise IC will be inactive.

(2) The setting of LVR (1.8V, 2.0V, 2.2V etc) will be valid just after successful power-on process.

8.1.8. Differences in command timing between single / double FPPA mode

Differences on PMC251 series instruction 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		2T	2T
Others		1T	1T

8.1.9. Program writing

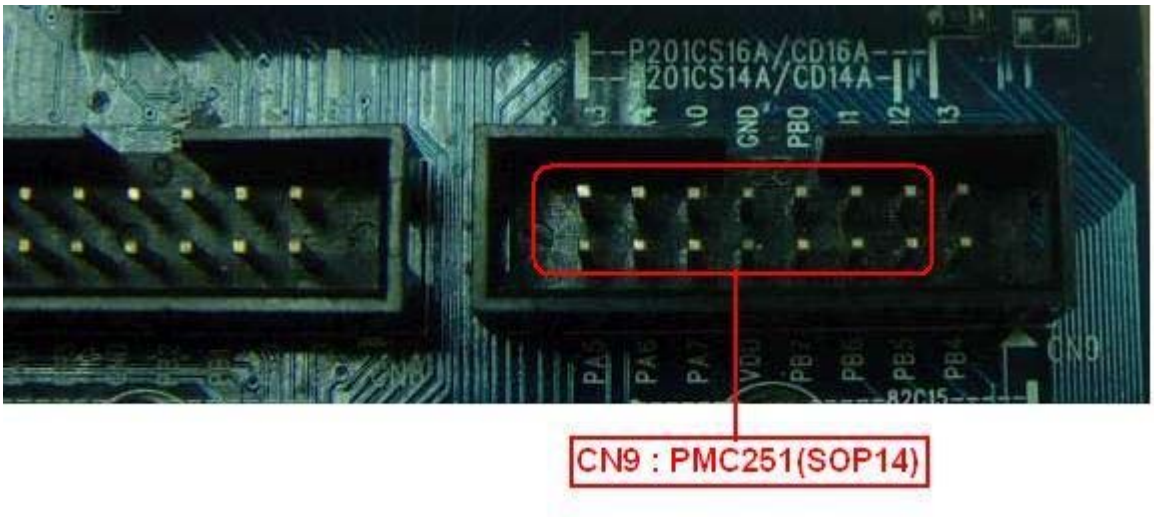
PMC251 8pin packages: Put the jumper over the P201CS14A location, and put the IC 3 pins downwards on the socket.

PMC251 other packages: Put the jumper over the P201-14PIN location, and put the IC at the top location of the socket.

8.2. Using ICE

8.2.1. Emulating PMC251 series IC on ICE PDK3S-I-001/002/003

PMC251 series I/O pin is defined as compatible with PDK201 series. When use ICE PDK3S-I-001/002/003 to emulate PMC251 series IC, please use cable labeled CN9：P201CS14A/CD14A to connect CN9 connector on ICE PDK3S-I-001/002/003 by matching each assembled pins respectively. Connection is shown in the figure below:



8.2.2. Important Notice for ICE operation

IC DK3S-I-001/002/003 does not support emulating below PMC251 series IC functions from (a) to (h)**.** Thus, user needs to take PMC251 Real Chip for test. Please notice：In order to avoid the problems on difference between ICE and Real Chip test procedure, those functions will be temporarily removed from datasheet before ICE is set to support them. PMC251 Real Chip is defaulted with these functions and performing ordinarily. Customer is recommended to consider these exceptions and careful handle whenever doing the test.

(a) PMC251 series IC is able to set LVD minimum to 1.8V, total 8 stages 4.0V, 3.5V, 3.0V, 2.75V, 2.5V, 2.2V, 2.0V, 1.8V.

(b) PMC251 series IC LVD function can be disabled by register (misc.2).

(c) PMC251 series IC is able to support LVD reset and Fast-recover by register (misc.3).

(d) PMC251 series IC is able to be set as single core operating model.

(e) PMC251 series IC supports VDD less than 4V, 3V, 2V voltage level detection and store detecting result to internal register (rstst).

(f) PMC251 series IC supports reset-source detection.

(g) PMC251 series IC Timer 16 Clock Source supports external PA4 signal input.

(h) PMC251 series IC provides watch-dog time out function which is assigned by register misc[1:0]