MCS11 ADC-Type Enhanced Field Programmable Processor Array (FPPA™)

Data Sheet

Preliminary

Version 0.20 – Apr. 20, 2018

IMPORTANT NOTICE

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

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

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

Table of Contents

Revision History:

Revision	Date	Description
0.10	2015/08/05	Draft version
0.20	2018/04/20	Amend PWM generator to 10-bit

1. Features

1-1. High Performance RISC CPU Array

- Patented Field Programmable Processor Array (FPPA™) Technology
- Operating modes: 8 processing units FPPA™ mode
- 4Kx16 bits OTP user program memory for 8 FPP units
- 208 Bytes data RAM for all FPP units
- 106 powerful instructions
- All instructions are 1T except indirect memory access, including branch instructions
- Programmable stack pointer to provide adjustable stack level
- Direct and indirect addressing modes for data and instructions
- All data memories are available for use as an index pointer
- Support security function to protect OTP data
- Separated IO and memory space to reduce firmware overhead in space exchange

1-2. System Functions

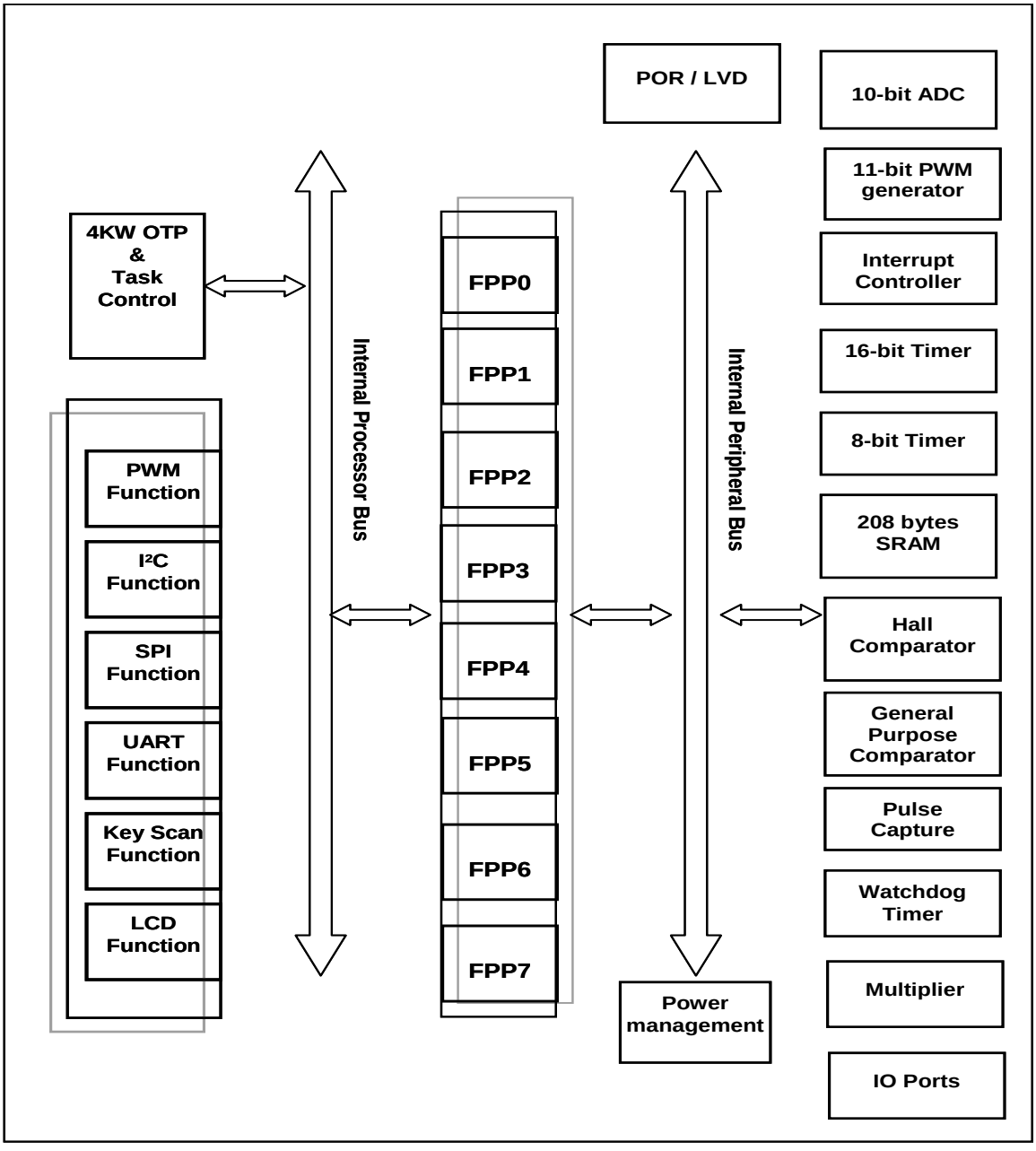
- Clock sources : internal high RC oscillator (IHRC), internal low RC oscillator (ILRC) and crystal oscillator
- Internal High RC Oscillator (IHRC) frequency
- Band-gap circuit to provide 1.20V reference voltage
- One hardware 16-bit timer
- One hardware 8-bit timer
- Up to 11-channel 10-bit resolution ADC with 1-channel for internal band-gap reference voltage
- One 10-bit hardware PWM generator
- PWM protection mechanism
- One 1T hardware multiplier
- One Hall comparator
- One general purpose comparator
- 16 IO pins with 10 mA capability and optional pull-high resistor
- Three levels of VDD voltage detection: 4.0V, 3.0V, 2.0V.
- Eight levels of LVD reset ~ 4.1V, 3.6V, 3.1V, 2.8V, 2.5V, 2.2V, 2.0V, 1.8V
- Selectable four external interrupt pins: PA0 or PA7, PB0 or PB7
- Support fast wake-up
- Every IO pin can be configured to enable wake-up function
- Operating voltage range: 2.2V ~ 5.5V
- Operating temperature range: -40°C ~ 85°C for normal condition, -40°C ~ 105°C for special condition*
- Operating frequency range:
  DC ~ 8MHz@VDD≧3.3V; DC ~ 4MHz@VDD≧2.5V; DC ~ 2MHz@VDD≧2.2V
- Low power consumption
  $I_{operating}$ ~ 1.7mA@1MIPS, VDD=5.0V; $I_{operating}$ ~ 10uA@ ILRC~12KHz, VDD=3.3V
  $I_{power\ down}$ ~ 1uA@VDD=5.0V; $I_{power\ down}$~ 0.5uA@VDD=3.3V
- 10-pin MSOP10 package

*Please see the item 4.1 for special condition.

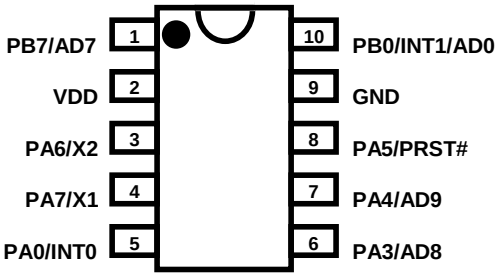
2. General Description and Block Diagram

The MCS11 is an ADC-Type of PADAUK's parallel processing, fully static, OTP-based CMOS 8x8 bit processor array that can execute eight peripheral functions in parallel. It employs RISC architecture based on patent pending FPPA™ (Field Programmable Processor Array) technology and all the instructions are executed in one cycle except that some instructions are two cycles that handle indirect memory access.

4Kx16 bits OTP user program memory and 208 bytes data SRAM are inside for 8 FPP units using, one up to 11 channels 10-bit ADC is built inside the chip with one channel for internal band-gap reference voltage; one general purpose comparator and one hall comparator are provided. There are two hardware timers are also provided: one is 16-bit timer and one is 8-bit timer with PWM generation. One hardware Pulse Capture, 10-bit hardware PWM generator and two PWM protection modules are also built inside the MCS11 in order to provide the best solution for BLDC controller.



3. MCS11 Pin Assignment and Description



MCS11(MSOP10-118mil)

Pin Name	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 when crystal oscillator is used. (3) External interrupt line 2. Both rising edge and falling edge are accepted to request interrupt service. (4) Output of PWM generator. 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 and interrupt functions are 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 when crystal oscillator is used. (3) Input of Pulse Capture. (4) Output of PWM generator. 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. (3) Output of Hall comparator. (4) Input of Pulse Capture. 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/AD9	IO ST / CMOS / Analog	The functions of this pin can be: (1) Bit 4 of port A. It can be configured as input or output with pull-up resistor. (2) Channel 9 input of ADC. (3) Output of PWM generator. (4) Minus input of general purpose comparator If this pin acts as analog input, bit 4 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 from this pin is also disabled when bit 4 of ***padier*** register is "0".

Pin Name	Buffer Type	Description
PA3/AD8	IO ST / CMOS / Analog	The functions of this pin can be: (1) Bit 3 of port A. It can be configured as input or output with pull-up resistor. (2) Channel 8 input of ADC. (3) Output of PWM generator. (4) Minus input of general purpose comparator. If this pin acts as analog input, bit 3 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 from this pin is also disabled when bit 3 of ***padier*** register is "0".
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. (3) Input of Pulse Capture. 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/AD7	IO ST / CMOS / Analog	The functions of this pin can be: (1) Bit 7 of port B. It can be configured as input or output with pull-up resistor. (2) Channel 7 input of ADC. (3) Input of Pulse Capture. (4) Minus input of general purpose comparator (5) External interrupt line 3. Both rising edge and falling edge are accepted to request interrupt service. If this pin acts as analog input, bit 7 of ***pbdier*** 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 from this pin is also disabled when bit 7 of ***pbdier*** register is "0".
PB0/INT1 /AD0	IO ST / CMOS / Analog	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) Channel 0 input of ADC. (3) Minus input of general purpose comparator (4) Plus input of general purpose comparator (5) Output of general purpose comparator (6) External interrupt line 1. Both rising edge and falling edge are accepted to request interrupt service. If this pin acts as analog input, bit 0 of ***pbdier*** 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 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. AC/DC Device Characteristics

Symbol	Description	Min	Typ	Max	Unit	Conditions (Ta=25℃)
V_{DD}	Operating Voltage	3.3	5.0	5.5	V	For Servo application -40 °C <Ta<85 °C
V_{FSV}	Forbidden V_{DD} Startup voltage Range*	0.7		1.6	V	
V_{PDRV}	V_{DD} power down release voltage			0.7	V	
T_{POR}	V_{DD} power on time (V_{DD} from 0V to 5V)			50	ms	
T_{FSV}	V_{DD} power on time during V_{FSV} range			10	ms	
f_{SYS}	System clock IHRC IHRC & crystal oscillator IHRC & crystal oscillator Internal low RC oscillator	 0 0 0 	 24K	 8M 4M 2M 	Hz	 V_{DD} ≧ 3.3V V_{DD} ≧ 2.5V 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		$0.2V_{DD}$	V	
V_{IH}	Input high voltage for IO lines	$0.8\ V_{DD}$		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
R_{PH}	Pull-high Resistance		62 100 210		KΩ	V_{DD}=5.0V V_{DD}=3.3V V_{DD}=2.2V
V_{LVR}	Low Voltage Reset*	3.82 3.31 2.8 2.57 2.34	4.15 3.60 3.05 2.80 2.55	4.48 3.89 3.29 3.02 2.75	V	
V_{POR}	Power-On Reset Voltage	1.8	2	2.2	V	Ta=25°C
V_{BG}	Band-gap Reference Voltage (before calibration)	1.12	1.20	1.28	V	VDD=5V, 25°C
	Band-gap Reference Voltage * (after calibration)	1.17*	1.20*	1.23*		VDD=2.2V ~ 5.5V, -40°C <Ta<105°C*

Symbol	Description	Min	Typ	Max	Unit	Conditions (Ta=25℃)
f_{IHRC}	Frequency of IHRC after calibration *	15.84*	16*	16.16*	MHz	25°C, VDD=3.5V~5.5V
		15.36*	16*	16.64*		VDD=3.5V~5.5V, 0°C <Ta<85°C*
		15.2*	16*	16.8*		VDD=4.75V~5.5V, 0°C <Ta<105°C*
		14.72*	16*	17.28*		VDD=3.5V~5.5V -20 °C <Ta<85 °C
		14.56*	16*	17.44*		VDD=4.75V~5.5V -20 °C <Ta<105 °C*
		14.08*	16*	17.92*		VDD=3.5V~5.5V -40 °C <Ta<85 °C*
		13.92*	16*	18.08*		VDD=4.75V~5.5V -40 °C <Ta<105 °C*
f_{ILRC}	Frequency of ILRC *	20.4*	24*	27.6*	KHz	VDD=5.0V, Ta=25°C
		15.6*	24*	32.4*		VDD=5.0V, -40°C <Ta<85°C*
		10.2*	12*	13.8*		VDD=3.3V, Ta=25°C
		7.8*	12*	16.2*		VDD=3.3V, -40°C <Ta<85°C*
V_{ADC}	Workable ADC operating Voltage	2.5		5.0	V	
V_{AD}	AD Input Voltage	0		VDD	V	
ADrs	ADC resolution			10	bit	
ADclk	ADC clock period		2		us	2.5V ~ 5.5V
t_{ADCONV}	ADC conversion time (T_{ADCLK} is the period of the selected AD conversion clock)		13 14 15		T_{ADCLK}	8-bit resolution 9-bit resolution 10-bit resolution
AD DNL	ADC Differential NonLinearity		±2*		LSB	
AD INL	ADC Integral NonLinearity		±4*		LSB	
ADos	ADC offset*		3 4		mV	-40 °C <Ta<85 °C -40 °C <Ta<105 °C
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 (T_{ILRC} is the clock period of ILRC)		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

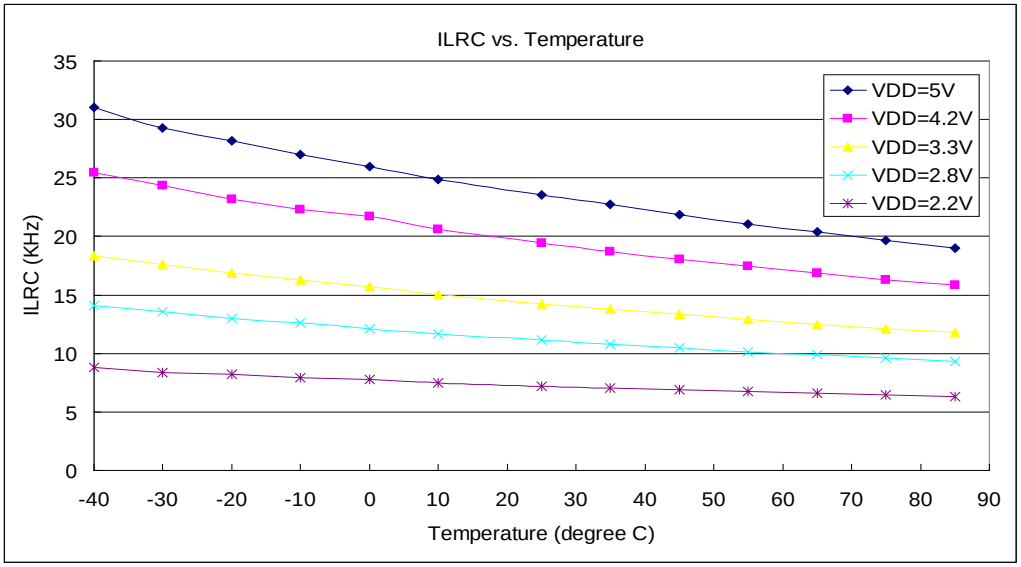
Symbol	Description	Min	Typ	Max	Unit	Conditions (Ta=25℃)
t_{WUP}	System wake-up period					
	Fast wake-up by IO toggle from STOPEXE suspend		128		T_{SYS}	Where T_{SYS} is the time period of system clock
	Fast wake-up by IO toggle from STOPSYS suspend, IHRC is the system clock		128 T_{SYS} + T_{SIHRC}			Where T_{SIHRC} is the stable time of IHRC from power-on.
	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.
	Normal wake-up from STOPEXE or STOPSYS suspend		1024		T_{ILRC}	Where T_{ILRC} is the clock period of ILRC
HCPos	Comparator offset*	-	±10	±20	mV	
HCPcm	Comparator input common mode*	0		VDD-1.5	V	
HCPspt	Comparator response time**		100	500	ns	Both Rising and Falling
HCPmc	Stable time to change comparator mode		2.5	7.5	us	

*These parameters are for design reference, not tested for each chip.
The larger f_{IHRC} drift will have adverse effect on the RPM accuracy for MCS11 SERVO application.
** Response time is measured with comparator input at (VDD-1.5)/2 -100mV, and (VDD-1.5)/2+100mV.

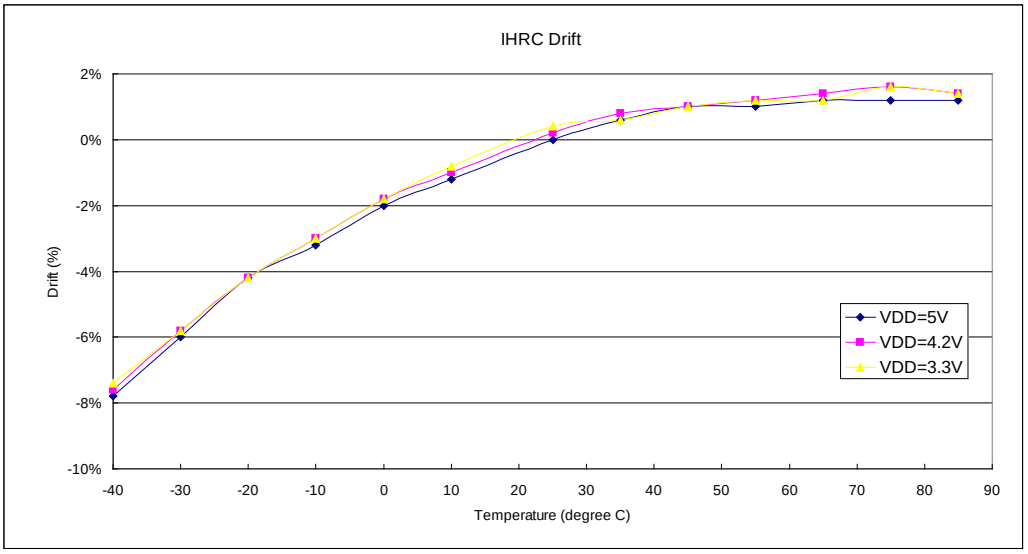
4-2. Absolute Maximum Ratings

- Supply Voltage ……..... 2.2V ~ 5.5V
- Input Voltage …….. -0.3V ~ VDD + 0.3V
- Operating Temperature …… -40℃ ~ 105℃
- Junction Temperature …….. 150℃
- Storage Temperature …… -50℃ ~ 125℃

4-3. Typical ILRC frequency vs. VDD and temperature



4-4. Typical IHRC frequency deviation vs. VDD and temperature



Note: IHRC is calibrated to 16MHz

4-5. Typical Operating Current vs. VDD and CLK=IHRC/n

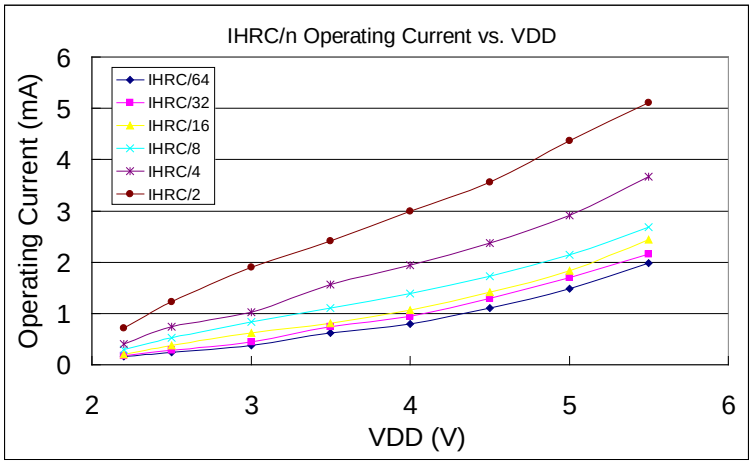
Conditions:

1-FPPA (code option)

ON: Band-gap, LVD, IHRC;

OFF: ILRC, EOSC, T16, TM2, ADC, PWM, Hall Comparator modules;

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



4-6. Typical Operating Current vs. VDD and CLK=ILRC/n

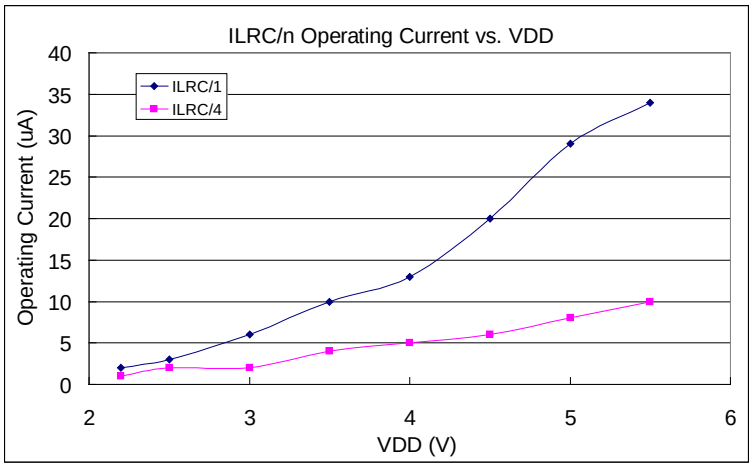
Conditions:

1-FPPA (code option)

ON: ILRC;

OFF: Band-gap, LVD, IHRC, EOSC, T16, TM2, ADC, PWM, Hall Comparator modules;

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



4-7. Typical Operating Current vs. VDD @CLK=32KHz EOSC/n

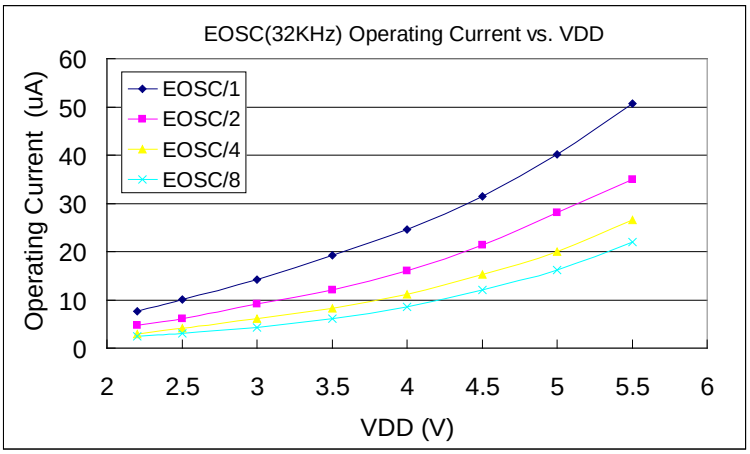
Conditions:

1-FPPA (code option)

ON: EOSC, MISC.6 = 1;

OFF: Band-gap, LVD, IHRC, ILRC, T16, TM2, ADC, PWM, Comparator modules;

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



4-8. Typical Operating Current vs. VDD @CLK=1MHz EOSC/n

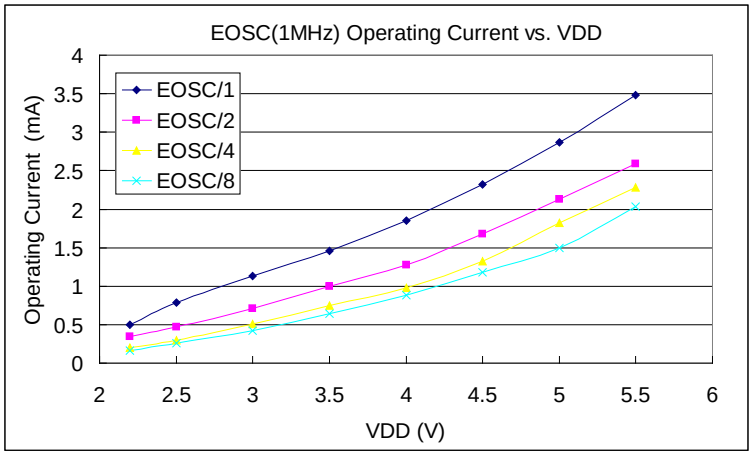
Conditions:

1-FPPA (code option)

ON: EOSC, MISC.6 = 1;

OFF: Band-gap, LVD, IHRC, ILRC, T16, TM2, ADC, PWM, Comparator modules;

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



4-9. Typical Operating Current vs. VDD @CLK=4MHz EOSC/n

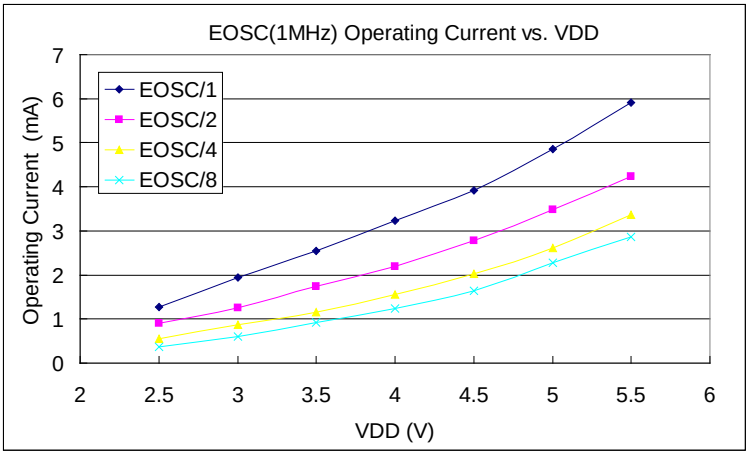
Conditions:

1-FPPA (code option)

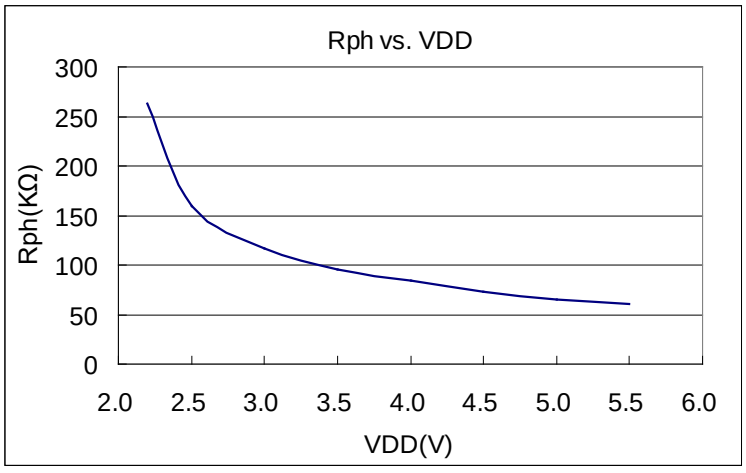
ON: EOSC, MISC.6 = 1;

OFF: Band-gap, LVD, IHRC, ILRC, T16, TM2, ADC, PWM, Comparator modules;

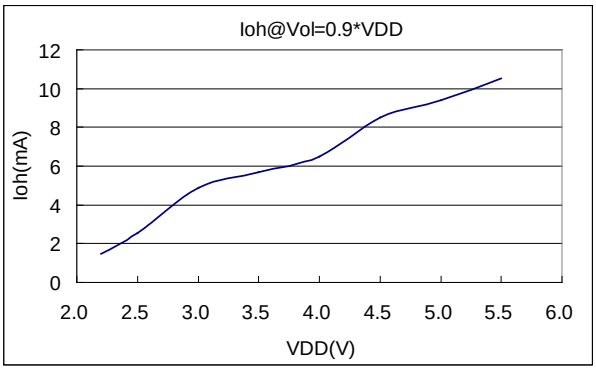
IO: PA0:0.5Hz output toggle and no loading, others: input and no floating

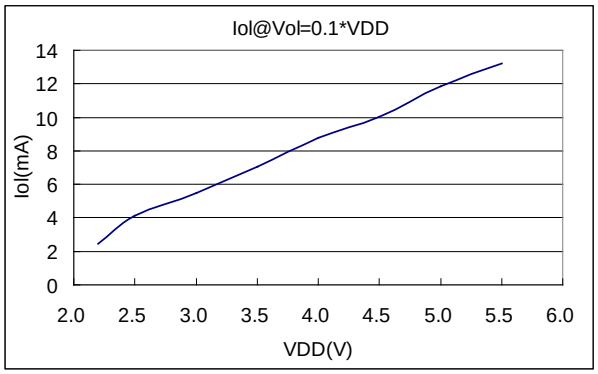


4-10. Typical IO pull high resistance

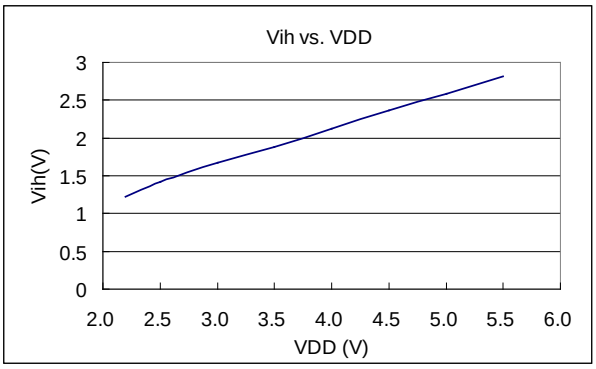


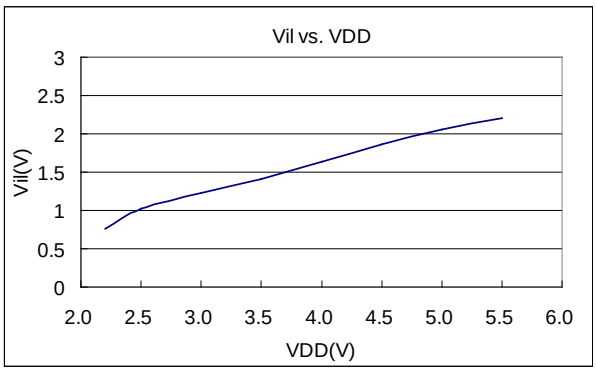
4-11. Typical IO driving current (I_{OH}) and sink current (I_{OL})



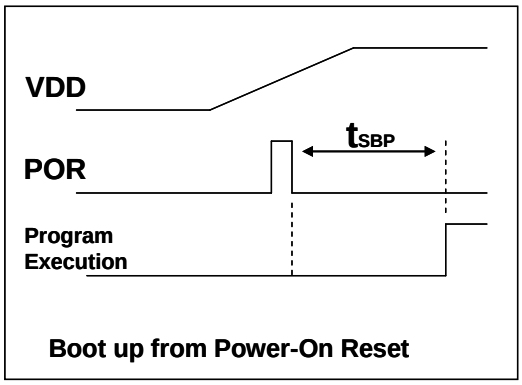


4-12. Typical IO input high/low threshold voltage (V_{IH}/V_{IL})

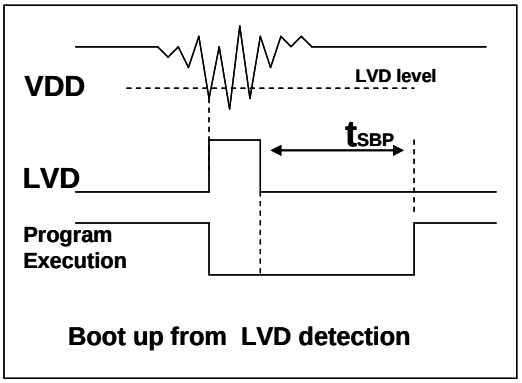




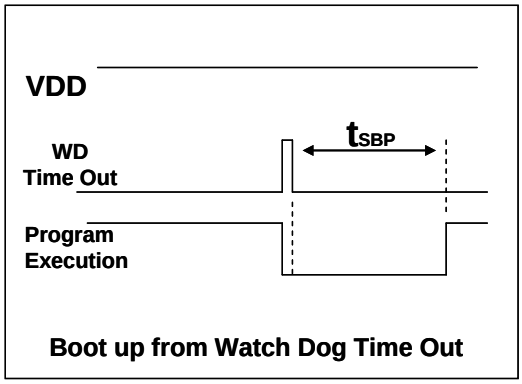
4-13. Timing charts for boot up conditions



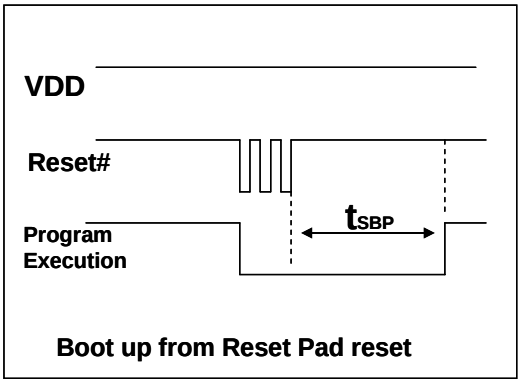
Boot up from Power-On Reset



Boot up from LVD detection



Boot up from Watch Dog Time Out



Boot up from Reset Pad reset

5. Functional Description

5-1. Processing Units

There are eight processing units (FPP unit) inside the MCS11. In every 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 eight FPP units share the same 4Kx16 bits OTP user program memory, 208 bytes data SRAM and all the IO ports, these eight 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 of processing units is illustrated in Fig. 5-1-1.

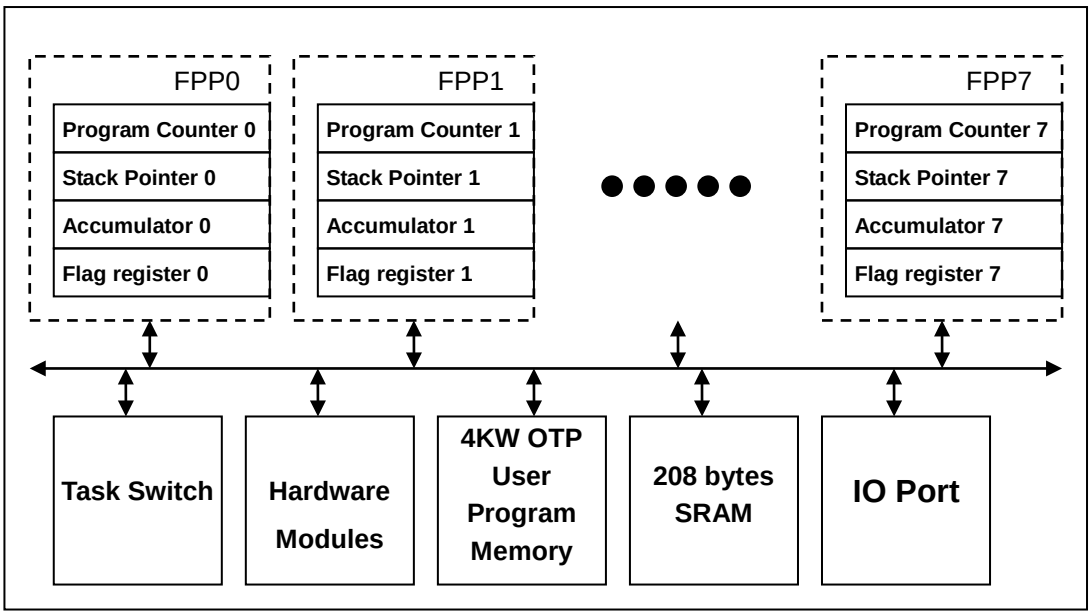


Fig. 5-1-1 Hardware Diagram of Processing units

These eight FPP units are operated at mutual exclusive clock cycles and can be enabled independently. The system performance is shared to the assigned FPP units via ***pmode*** command; please refer to the description of ***pmode*** instruction. The bandwidth assignment is nothing to do with FPP enable, means that the bandwidth is also allocated to the assigned FPP unit even though it is disabled.

Fig. 5-1-2 shows the timing sequence of FPP units for ***pmode***=0 which will assign the bandwidth to two FPP units only. FPP0 and FPP1 each have half computing power of whole system; for ***pmode***=0, FPP0 and FPP1will be operated at 4MHz if system clock is 8MHz. For FPP0 unit, its program will be executed in sequence every other system clock, shown as $(M-1)_{th}$, M_{th}, …. $(M+4)_{th}$ instructions. For FPP1 unit, its program will be also executed in sequence every other system clock, shown as $(N-1)_{th}$, N_{th}, …. $(N+3)_{th}$ instructions.

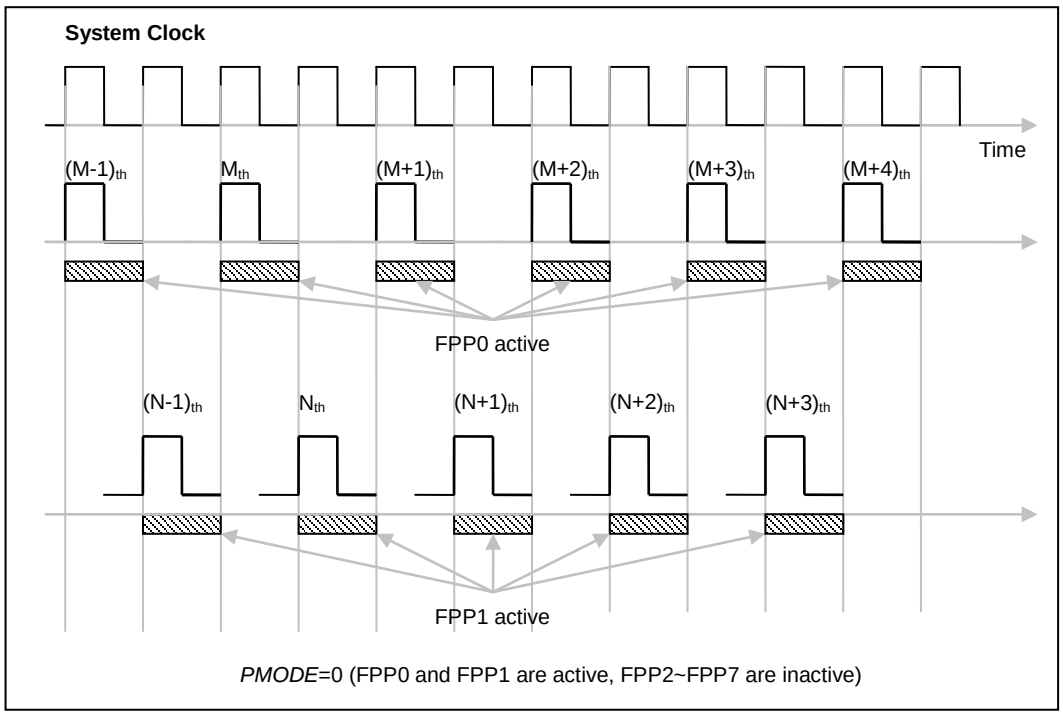


Fig. 5-1-2 Timing Sequence of Processing units for ***pmode***=0

Fig. 5-1-3 shows the timing sequence of FPP units for ***pmode***=6 which will assign the bandwidth to four FPP units (FPP0, FPP1, FPP2, FPP3); for ***pmode***=6, FPP0, FPP1, FPP2 and FPP3 will be operated at 2MHz if system clock is 8MHz, means that each FPP unit has quarter computing power of whole system, however, FPP4, FPP5, FPP6 and FPP7 are inactive; For FPP0 unit, its program will be executed once in sequence every four system clock, shown as $(M-1)_{th}$, M_{th}, …. $(M+4)_{th}$ instructions. For FPP1 unit, its program will be also executed once in sequence every four system clock, shown as $(N-1)_{th}$, N_{th}, …. $(N+3)_{th}$ instructions. For FPP2 unit, its program will be also executed once in sequence every four system clock, shown as $(O-1)_{th}$, O_{th}, …. $(O+3)_{th}$ instructions. For FPP3 unit, its program will be also executed once in sequence every four system clock, shown as $(P-1)_{th}$, P_{th}, …. $(P+3)_{th}$ instructions.

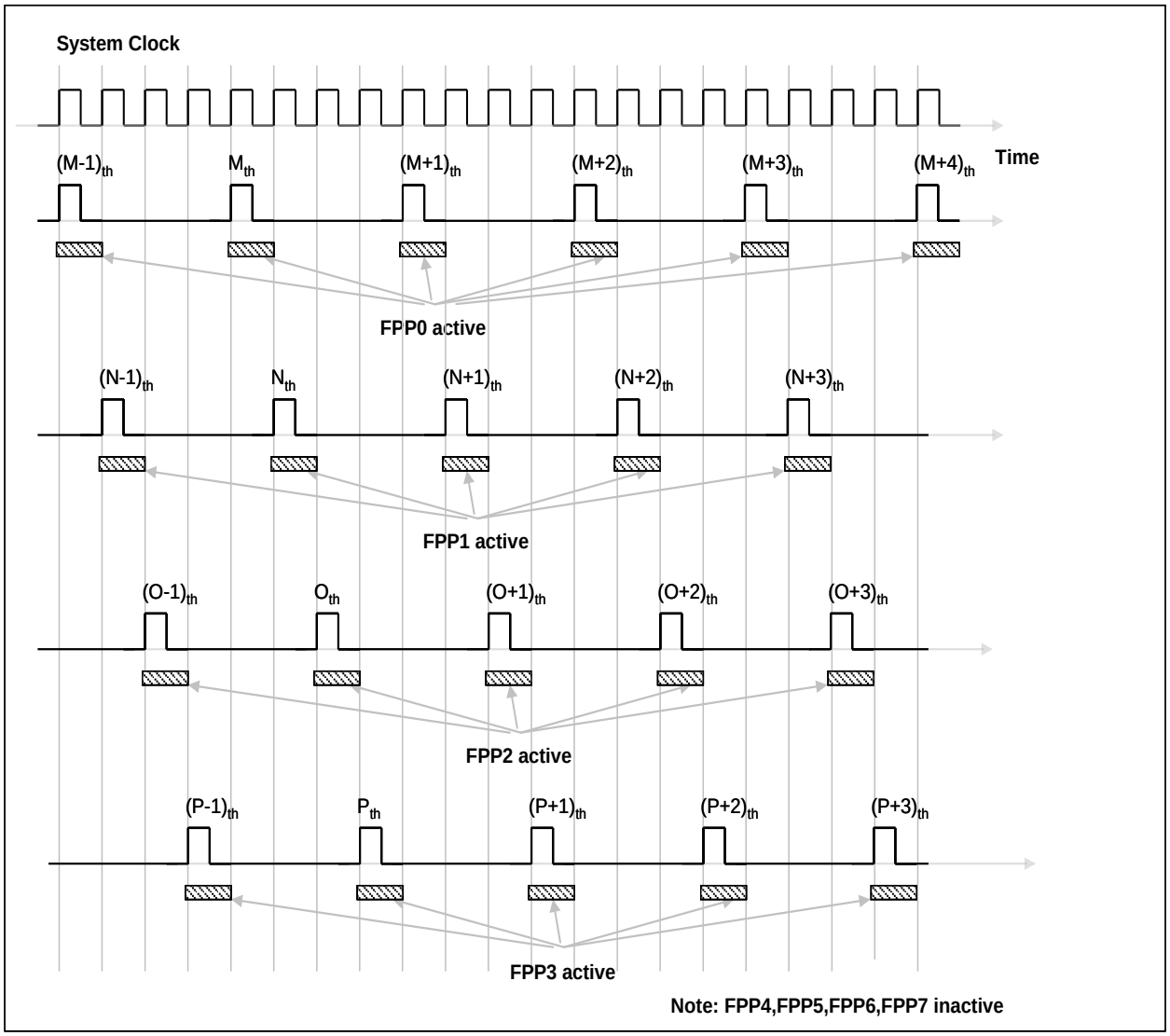


Fig. 5-1-3 Timing Sequence of Processing units for ***pmode***=6

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 other units can be enabled by user's program if necessary. All the FPP units can be enabled or disabled by using any one FPP unit, including it.

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 12 for MCS11. The program counter of FPP0 is 0 after hardware reset, 1 for FPP1, 2 for FPP2, 3 for FPP3, 4 for FPP4, 5 for FPP5, 6 for FPP6 and 7 for FPP7. Whenever interrupt event happens, only FPP0 will be informed and its program counter will jump to 'h10 for interrupt service routine. All the FPP units have 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 and data memory is 208 bytes; therefore, the stack memory should be defined within 208 bytes from 0x00h address. The stack memory of MCS11 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. 5-1-4 shows that the status of stack before FPP0 execution, system has calculated the required stack space and has reserved for the program.

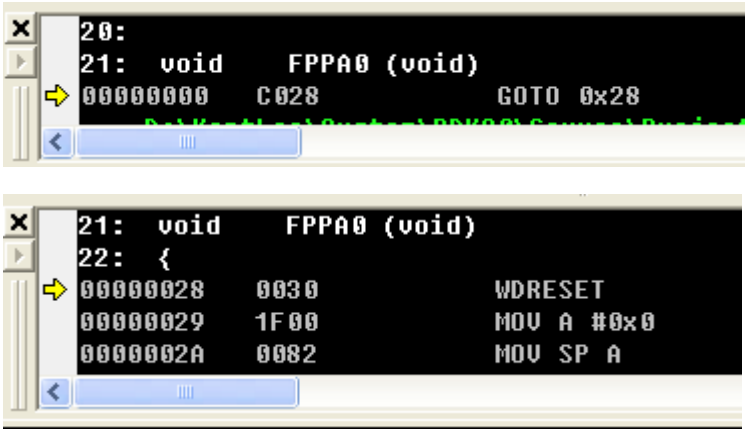


Fig. 5-1-4 Stack Assignment in Mini-C project

5-2. Program Memory – OTP

The OTP (One Time Programmable) program memory is used to store the program instructions to be executed. There are 4KW OTP program inside the MCS11, All the user program codes for all FPP units are stored in this 4KW OTP memory. The OTP program memory may contains the data, tables and interrupt entry. After reset, the initial address for FPP0 is ‘h000, ‘h001 for FPP1, ‘h002 for FPP2, ‘h003 for FPP3, ‘h004 for FPP4, ‘h005 for FPP5, ‘h006 for FPP6 and ‘h007 for FPP7; the interrupt entry is ‘h010 and only FPP0 will be informed. The OTP program memory for MCS11 is partitioned as below.

Address	Function
000	FPP0 reset – goto instruction
001	FPP1 reset – goto instruction
002	FPP2 reset – goto instruction
003	FPP3 reset – goto instruction
004	FPP4 reset – goto instruction
005	FPP5 reset – goto instruction
006	FPP6 reset – goto instruction
007	FPP7 reset – goto instruction
008	User program memory
•	•
00F	User program memory
010	Interrupt entry address
011	User program memory
•	•
•	•
FF7	User program memory
FF8	System using
•	•
FFF	System using

Table 5-2-1: Program Memory Organization

In order to have maximum flexibility for user program using, the user program memory is shared for all FPP units, and the program space allocation is done by program compiler automatically, user does not need to specify the address if not necessary. Table 5-2.2 shows one example of program memory using which two FPP units are used.

Address	Function
000	FPP0 reset – goto instruction (goto ‘h011)
001	FPP1 reset – goto instruction (goto ‘h3A1)
002	Reserved
•	•
00F	Reserved
010	Interrupt entry address
011	Begin of FPP0 user program
•	•
•	•
7A0	End of FPP0 user program
7A1	Begin of FPP1 program
•	•
•	•
F37	End of FPP1 program
F38	Not used
•	•
•	•
FF7	Not used
FF8	System Using
•	•
FFF	System Using

Table 5-2.2: Example of Program Memory Using

5-3. Program Structure

After power-up, the program starting address of FPP0 is 0x000, 0x001 for FPP1, 0x002 for FPP2, 0x003 for FPP3, 0x004 for FPP4, 0x005 for FPP5, 0x006 for FPP6 and 0x007 for FPP7. The 0x010 is the entry address of interrupt service routine, which belongs to FPP0 only. The basic firmware structure for MCS11 is shown as Fig. 5-3-1, it shows that there are four FPP units are used; the program codes of four 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.

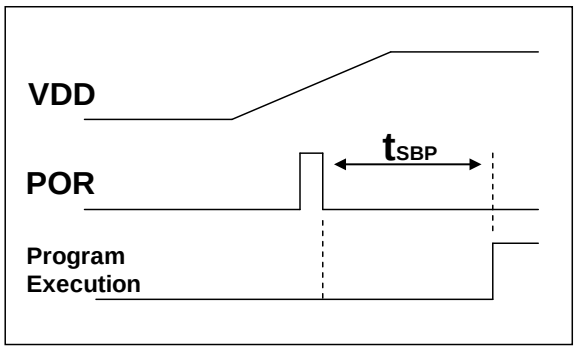
```
// Page 1
.romadr  0x00
// Program Begin
      goto     fpp0Boot;
      goto     fpp1Boot;
      goto     fpp2Boot;
      goto     fpp3Boot;

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

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

```
// Page 2
//------ Begin of FPP1 ----------
fpp1Boot :
    //--- Initialize FPP1 SP and so on
fpp1Loop:
…
   goto fpp1Loop:
//--------- End of FPP1 --------
//------ Begin of FPP2 ----------
fpp2Boot :
    //--- Initialize FPP2 SP and so on
fpp2Loop:
…
   goto fpp2Loop:
//--------- End of FPP2 --------
//------ Begin of FPP3 ----------
fpp3Boot :
    //--- Initialize FPP3 SP and so on…
…
fpp3Loop:
…
   goto fpp3Loop:
//--------- End of FPP3 --------
```

Fig. 5-3-1 Program Structure

5-4. Boot Procedure

POR (Power-On-Reset) is used to reset MCS11 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. 5-4-1. After boot up procedure, the default system clock is ILRC. If user wants to switch the system clock source from ILRC to IHRC or EOSC, user must enable the corresponding oscillator module and make sure clock is already stable.



Boot up from Power-On Reset

Fig. 5-4-1 Power-On Sequence

Fig. 5-4-2 shows the typical program flow after boot up, it shows all the FPP units are used. Please notice that the FPP1~FPP7 are disabled after reset, recommending NOT to enable FPP1~FPP7 before system and FPP0 initialization.

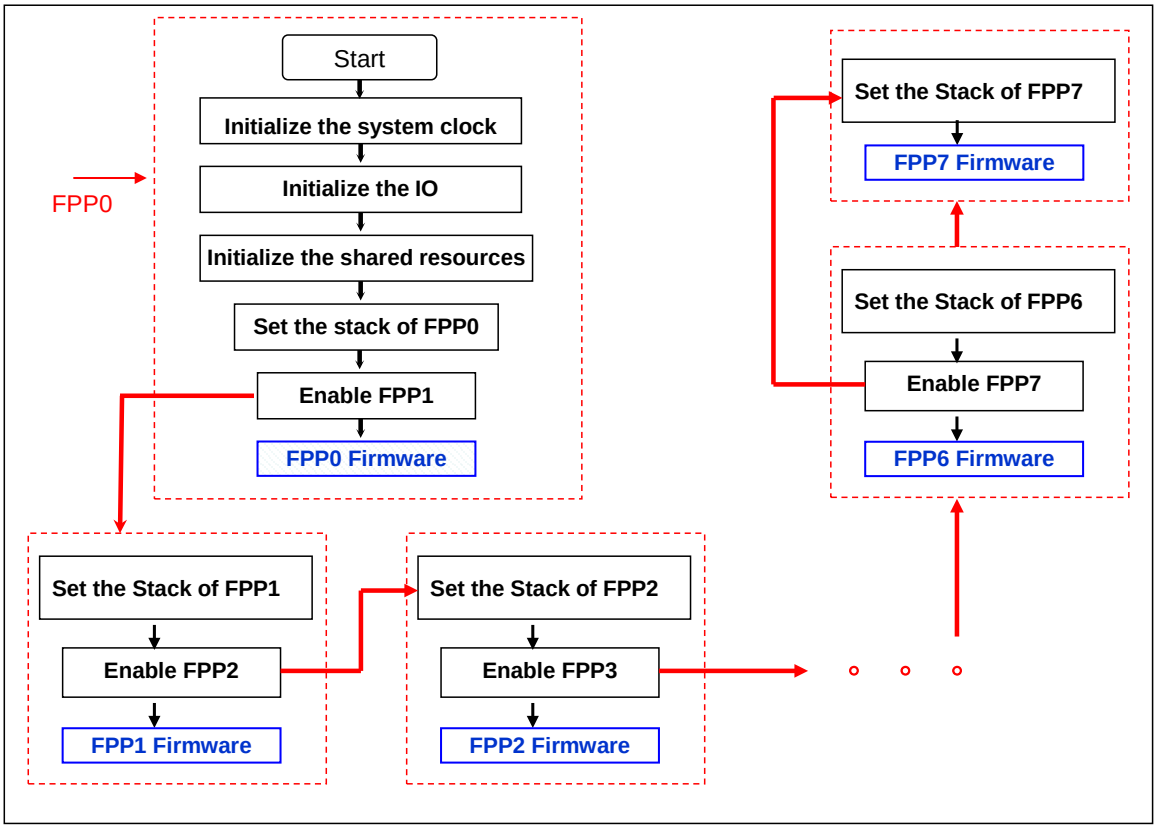


Fig. 5-4-2 Boot Procedure

5-5. Data Memory -- SRAM

Fig. 5-5-1 shows the SRAM data memory organization of MCS11, all the SRAM data memory could be accessed by every FPP unit 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 208 bytes data memory of MCS11 can be accessed by indirect access mechanism.

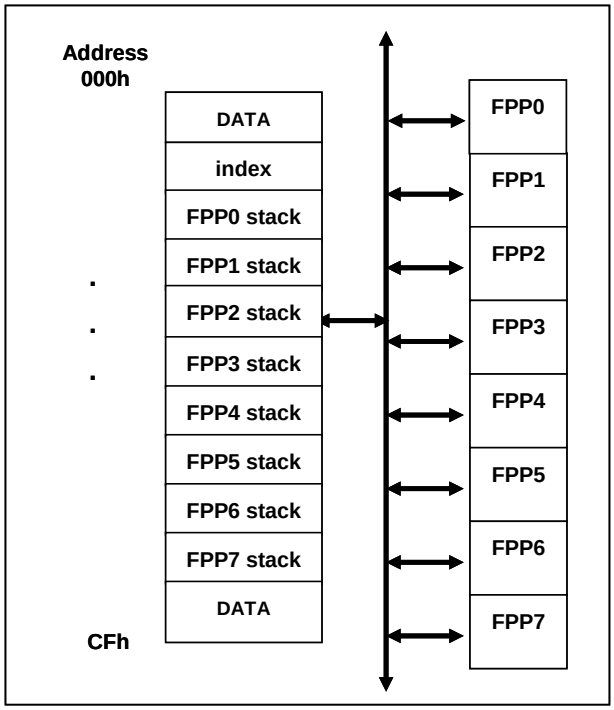


Fig. 5-5-1 Data Memory Organization

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. All the FPP units share the ALU for its corresponding operation.

5-7. Oscillator and clock

There are three oscillator circuits provided by MCS11: 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 1% normally after calibration and it still drifts slightly with supply voltage and operating temperature, the total drift rate is about ±4% for VDD=3.5V~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 24 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. Chip calibration

The IHRC frequency and band-gap reference voltage may be different chip by chip due to manufacturing variation, MCS11 provide both the IHRC frequency calibration and band-gap 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, Band-gap=(**p4**);

Where,

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

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

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

p4= On or Off; Band-gap calibration is On or Off.

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 MCS11 for different option:

(1) .ADJUST_IC SYSCLK=IHRC/2, IHRC=16MHz, VDD=5V, Band-gap =On

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, BG=1.2V

(2) .ADJUST_IC SYSCLK=IHRC/4, IHRC=16MHz, VDD=3.3V, Band-gap =On

After boot, 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, BG=1.2V

(3) .ADJUST_IC SYSCLK=IHRC/8, IHRC=16MHz, VDD=2.5V, Band-gap =On

After boot, 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, BG=1.2V

(4) .ADJUST_IC SYSCLK=IHRC/16, IHRC=16MHz, VDD=2.2V, Band-gap =On

After boot, CLKMD = 0x1C：

- IHRC frequency is calibrated to 16MHz@VDD=2.2V and IHRC module is enabled
- System CLK = IHRC/16 = 1MHz
- Watchdog timer is disabled, ILRC is enabled, PA5 is in input mode, BG=1.2V

(5) .ADJUST_IC SYSCLK=IHRC/32, IHRC=16MHz, VDD=5V, Band-gap =Off

After boot, 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, Band-gap =Off

After boot, 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, 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. Crystal Oscillator

If crystal oscillator is used, a crystal or resonator is required between X1 and X2. Fig. 5-7-1 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.

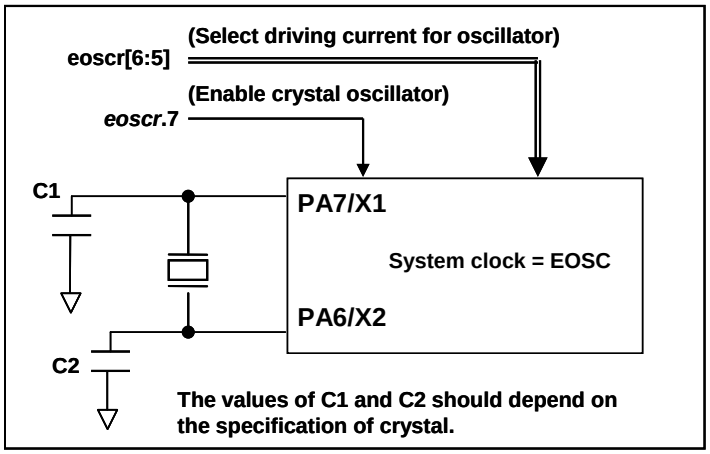


Fig. 5-7-1 Connection of crystal oscillator

Besides crystal, external capacitor and options of MCS11 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;
   stt16count;
   Intrq.T16 =   0;
   wait1     Intrq.T16;              // count fm 0x0000 to 0x2000,
                                        then setINTRQ.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 LVR (Low Voltage Reset) level

The system clock of MCS11 can come from EOSC, IHRC or ILRC, the hardware diagram for system clock in the MCS11 is shown as Fig. 5-7-2. MCS11 can provide the wide range system clock via ***clkmd*** register selection. After power up, the running system clock will be ILRC/1, user can change the system clock any time by setting ***clkmd*** register and the new system clock will be changed into the new one immediately after writing the ***clkmd*** register.

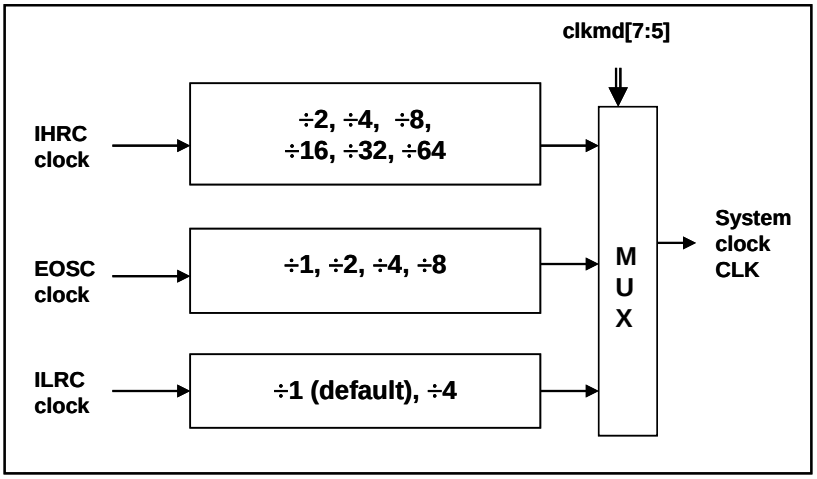


Fig. 5-7-2 Options of System Clock

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

- system clock = 8MHz with LVR=3.5V
- system clock = 4MHz with LVR=2.5V
- system clock = 2MHz with LVR=2.2V

5-8. 16-bit Timer (Timer16)

MCS11 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. 5-8-1.

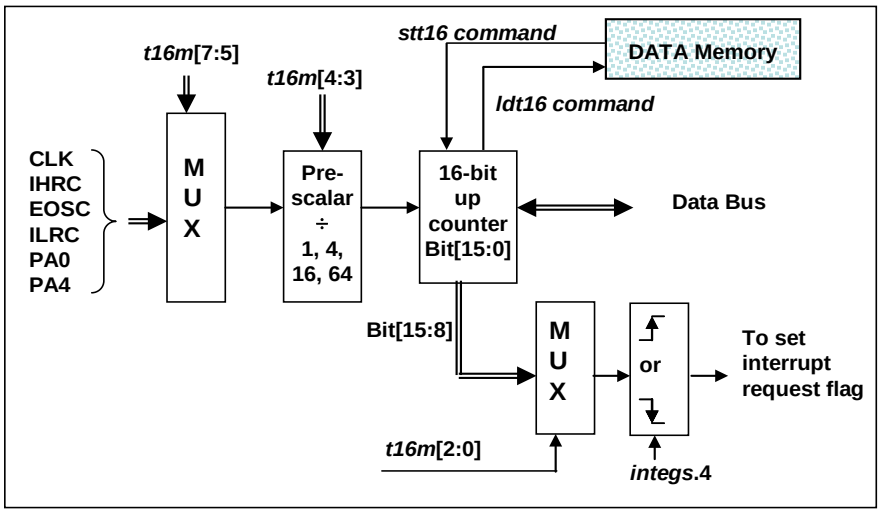


Fig. 5-8-1 Hardware diagram of Timer16

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, PA4, IHRC, EOSC, ILRC, PA0      // 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 of interrupt can be described as below:

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

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

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

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

5-9. 8-bit Timer (Timer2)

One 8-bit hardware timer (Timer2) is implemented in the MCS11; please refer to Fig. 5-9-1, shown its hardware diagram. The clock sources of Timer2 may come from system clock (CLK), internal high RC oscillator (IHRC), internal low RC oscillator (ILRC), PA0, PB0 and PA4, bit [7:4] of register ***tm2c*** is used to select the clock source of Timer2. If IHRC is selected as the clock source, the clock signal sent to Timer2 will keep running when using ICE in halt state. A clock pre-scaling module is provided with divided-by-1, 4, 16, and 64 options, controlled by bit [6:5] of *tm2s* register; one scaling module with divided-by-1~31 is also provided and controlled by bit [4:0] of *tm2s* register. In conjunction of pre-scaling function and scaling function, the frequency of Timer2 clock (TM2_CLK) can be wide range and flexible.

The Timer2 counter performs 8-bit up-counting operation only; the 8-bit counter will be clear to zero and generate interrupt request automatically whenever its values reach to that of bound register *tm2b*, the bound register is used to define the period of Timer2. The timing diagram of Timer2 is shown in Fig.5-9-2.

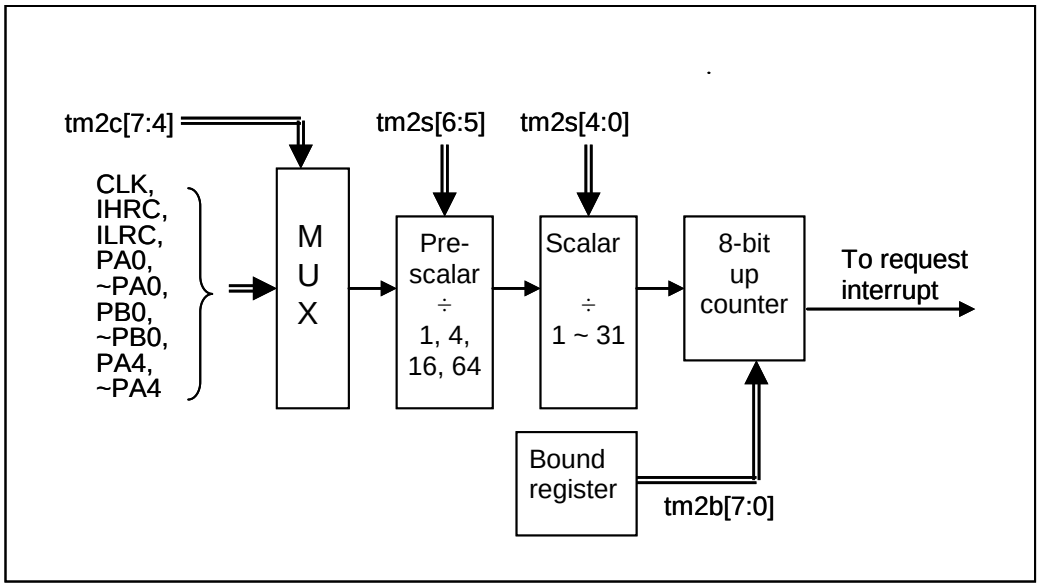


Fig. 5-9-1 Timer2 hardware diagram

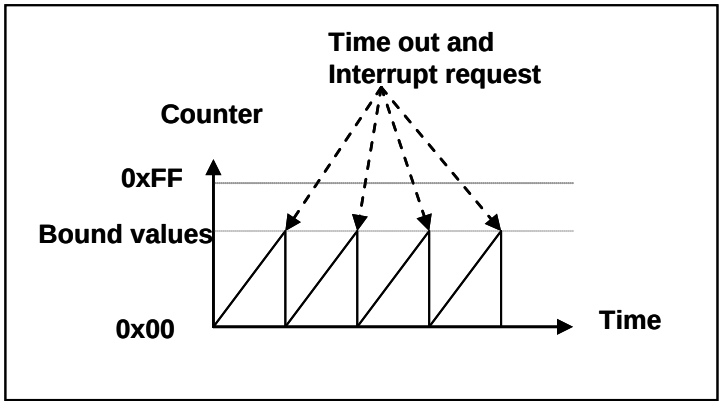


Fig. 5-9-2 Timing diagram of Timer2

5-10. 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 period when misc[1:0]=11
- 16384 ILRC period when misc[1:0]=10
- 4096 ILRC period when misc[1:0]=01
- 2048 ILRC 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 safe operation. WDT can be cleared by power-on-reset or by command ***wdreset*** at any time. When WDT is timeout, MCS11 will be reset to restart the program execution. The relative timing diagram of watchdog timer is shown as Fig.5-10-1. 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.

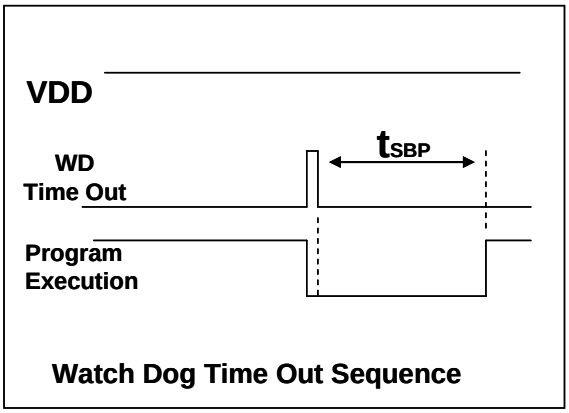


Fig. 5-10-1 Time-Out Sequence of WatchDog Timer

5-11. Interrupt

There are eight interrupt lines for MCS11: four external interrupt lines (PA0 or PA5, PB0 or PB7, the edge type is specified by ***integsr*** register), Timer16 interrupt, Timer2 interrupt, hall comparator interrupt, PWM generator interrupt and ADC 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. 5-11-1. All the interrupt request flags are set by hardware and cleared by software. All the interrupt request lines are also controlled by ***engint*** command (enable global interrupt) to enable interrupt operation and ***disgint*** command (disable global interrupt) to disable it. 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.

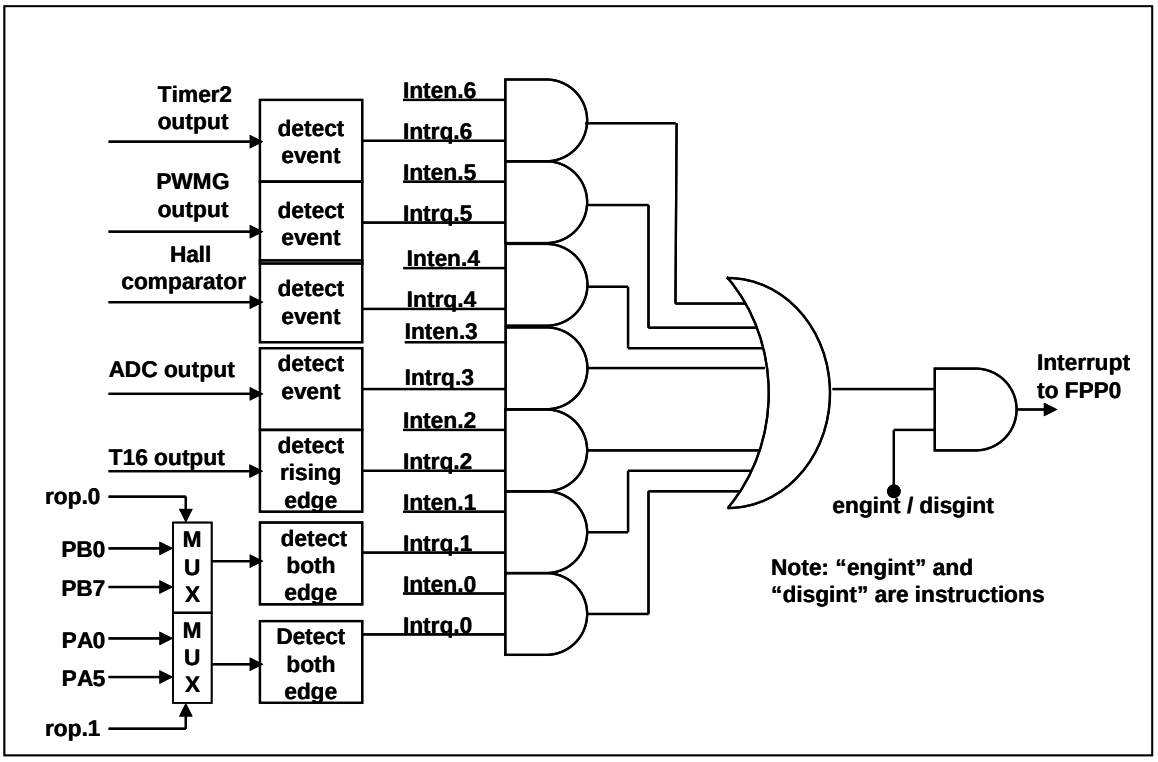


Fig. 5-11-1 Hardware diagram of interrupt controller

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
}
```

5-12. 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. 5-12-1 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. 5-12-1 Differences in oscillator modules between STOPSYS and STOPEXE

5-12-1. Power-Save mode ("*stopexe*")

Using "***stopexe***" instruction to enter the Power-Save mode, only system clock is disabled, remaining all the oscillator modules active. For CPU, it stops executing; however, for Timer16, counter keep counting if its clock source is not the system clock. The wake-up sources for "***stopexe***" can be IO-toggle or Timer16 counts to the set values when clock sources of Timer16 come from 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 shows 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.

5-12-2. 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 MCS11 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:

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

5-12-3. Wake-up

After entering the Power-Down or Power-Save modes, the MCS11 can be resumed to normal operation by toggling IO pins, Timer16 interrupt is available for Power-Save mode ONLY. Fig. 5-12-2 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. 5-12-2 Differences in wake-up sources between Power-Save mode and Power-Down mode

When using the IO pins to wake-up the MCS11, 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 ***mis***c 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 there is no fast wake-up mode whenever EOSC is enabled.

Suspend mode	wake-up mode	system clock source	wake-up time (t_{WUP}) from IO toggle
STOPEXE suspend	fast wake-up	IHRC or 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.
STOPSYS suspend	fast wake-up	ILRC	128 T_{SYS} + T_{SILRC}; Where T_{SILRC} is the stable time of ILRC from power-on.
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
STOPSYS suspend	normal wake-up	Any one	1024 * T_{ILRC}, Where T_{ILRC} is the clock period of ILRC

5-13. 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. 5-13-1.

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

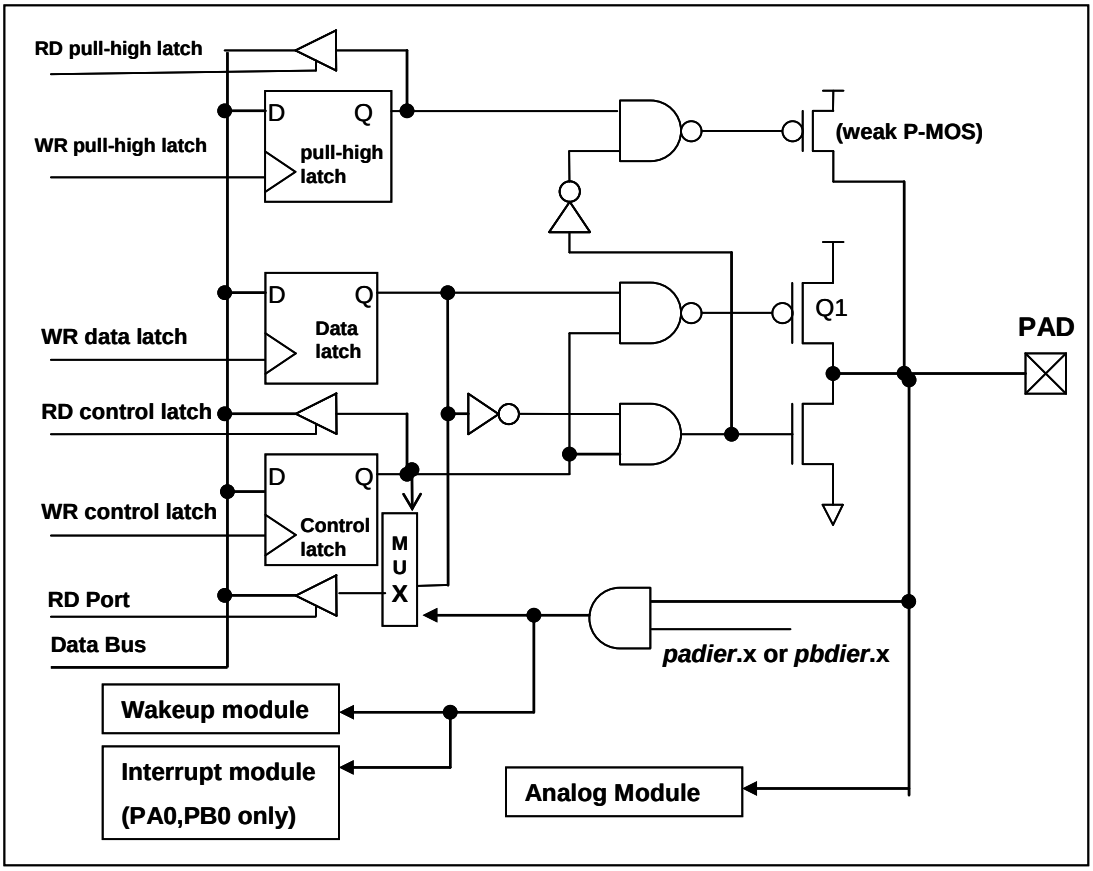


Fig. 5-13-1 Hardware diagram of IO buffer

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 MCS11 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.

5-14. Reset and LVD (Low Voltage Detection)

5-14-1. Reset

There are many causes to reset the MCS11, once reset is asserted, most of all the registers in MCS11 will be set to default values, When reset comes from WDT timeout, ***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.

5-14-2. LVD reset

By code option, there are 8 different levels of LVD for reset ~ 4.0V, 3.5V, 3.0V, 2.75V, 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.

5-15. Hall Comparator

The hall comparator is built by hardware circuitry HC_1 and HC_2. PA4 is the plus input for both HC_1 and HC_2, PA0, PA1 and PA2 are the minus input by option. The hall comparator output (HC_Out) is combined by HC_Out1 and HC_Out2. HC_Out can be output to PA5 and internal used. The hall comparator adjust registers (***hc1a*** and ***hc2a***) are used to control the offset value of HC_1 and HC_2 in order to optimize the system performance.

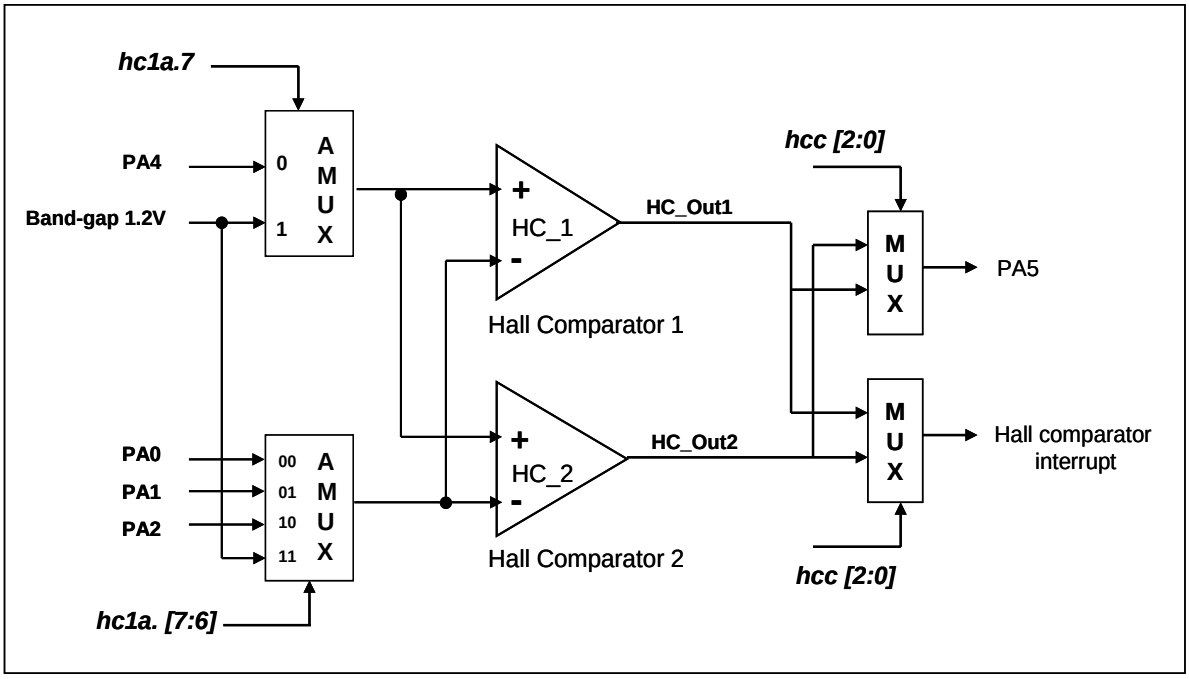


Fig. 5-15-1 The hardware diagram of Hall comparator

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

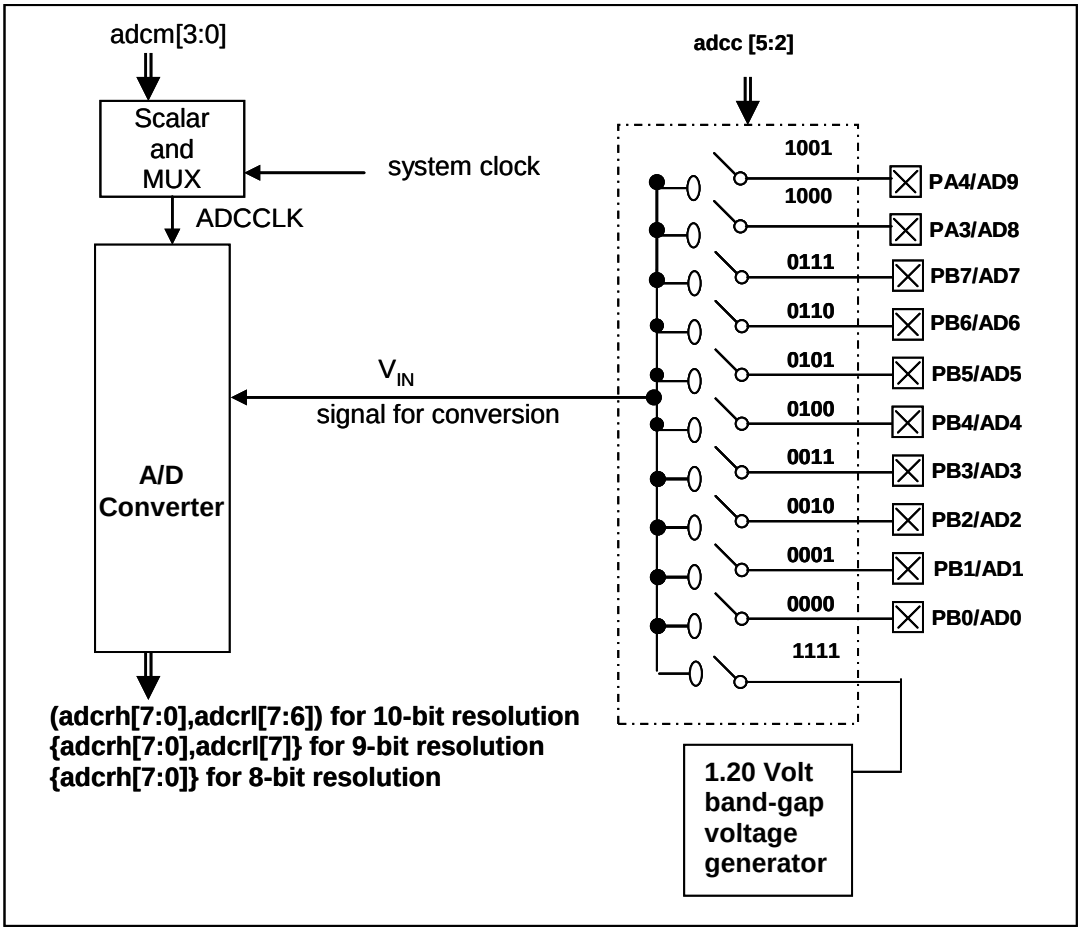


Fig. 5-16-1 ADC Block Diagram

The MCS11 provides one 10-bit resolution analog-to-digital conversion module with 10 external channels and one channel for internal 1.20 volt band-gap reference voltage; it allows the conversion of an analog input signal to a corresponding maximum 10-bit digital number, its block diagram is shown as Fig. 5-16-1. For the conversion process, analog signal will be sampled and held first, then sending into the converter to generate the result via successive approximation. The analog reference high voltage of ADC is the positive supply voltage (VDD) and the reference low is always the GND. Higher than 1uF capacitor is recommended to be placed between VDD and GND to have better AD conversion result.

5-16-1. The input requirement for AD conversion

For the AD conversion to meet its specified accuracy, the charge holding capacitor (C_{HOLD}) must be allowed to fully charge to the voltage reference high level (VDD) and discharge to the voltage reference low level (GND). The analog input model is shown as Fig. 5-16-2, the signal driving source impedance (Rs) and the internal sampling switch impedance (Rss) will affect the required time to charge the capacitor C_{HOLD} directly. The internal sampling switch impedance may vary with ADC supply voltage; the signal driving source impedance will affect accuracy of analog input signal. User must ensure the measured signal is stable before sampling; therefore, the maximum signal driving source impedance is highly dependent on the frequency of signal to be measured. The recommended maximum impedance for analog driving source is about 10KΩ under 500KHz input frequency and 10-bit resolution requirements, and 10MΩ under 500Hz input frequency and 10-bit resolution.

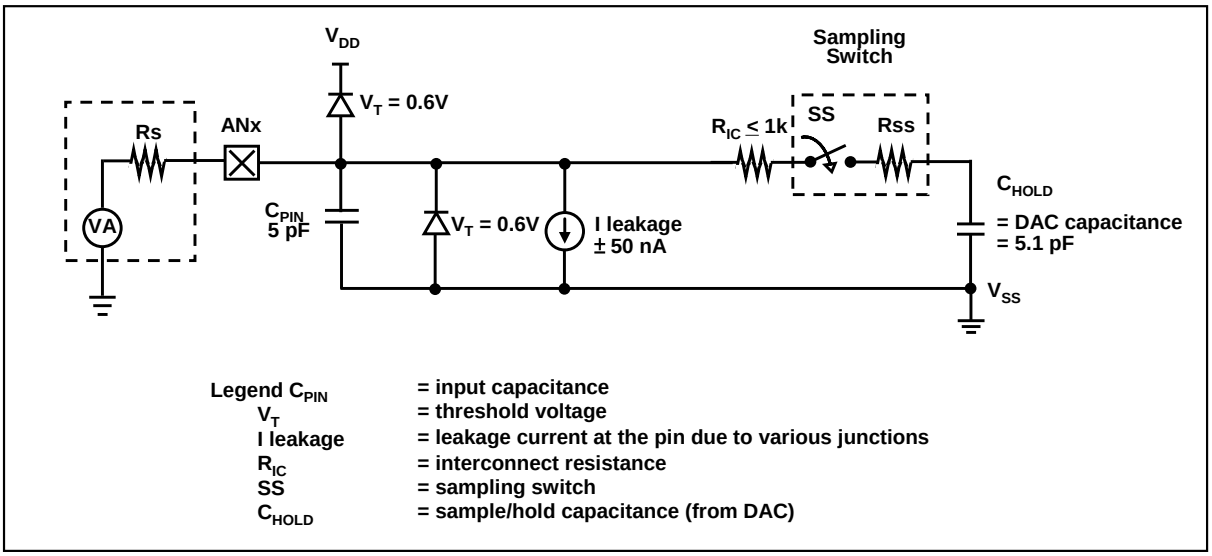


Fig. 5-16-2 Analog input model of ADC

Before starting the AD conversion, the minimum signal acquisition time should be met for the selected analog input signal. The signal acquisition time (T_{ACQ}) of ADC in MCS11 series is fixed to one clock period of ADCLK, the selection of ADCLK must be met the minimum signal acquisition time.

5-16-2. Select the ADC bit resolution

The ADC bit resolution is also selectable from 8-bit to 10-bit, depending on the requirement of customers' application. Higher resolution can detect small signal variation; however, it will take more time to convert the analog signal to digital signal. The selection can be done via ***adcm*** register. The ADC bit resolution should be configured before starting the AD conversion.

5-16-3. ADC clock selection

The clock of ADC module (ADCLK) can be selected by ***adcm*** register; there are eight options for ADCLK from sysclk/1 to sysclk/128. Due to the signal acquisition time T_{ACQ} is one clock period of ADCLK, the ADCLK must meet that requirement. The recommended ADC clock is to operate at 2us.

5-16-4. AD conversion

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

- 8-bit resolution: AD conversion time = 13 T_{ADCLK}
- 9-bit resolution: AD conversion time = 14 T_{ADCLK}
- 10-bit resolution: AD conversion time = 15 T_{ADCLK}

5-16-5. Configure the analog pins

The 10 external analog input signals for ADC shared with PA3, PA4 and PB[7:0]. In order to avoid leakage current at the digital circuit portion, those pins which are defined for analog inputs should disable the digital input function (set the corresponding bit of ***padidr*** or ***pbdidr*** register to be 1). Because the measurement signals of ADC belong to small signal; it should avoid the measured signal to be interfered during the measurement period, the selected pin should (1) be set to input mode (2) turn off weak pull-high resistor (3) set the corresponding pin to analog input by port A/B digital input disable register (***padidr*** / ***pbdidr***).

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

(1) Configure the ADC module:
- Select the ADC input channel by ***adcc*** register
- Select the ADC input channel by ***adcc*** register
- Select the bit resolution of ADC by ***adcm*** register
- Configure the AD conversion clock by ***adcm*** register
- Configure the pin as analog input by ***padidr, pbdidr*** register
- Enable the ADC module by ***adcc*** register

(2) Configure interrupt for ADC: (if desired)
- Clear the ADC interrupt request flag in bit 3 of ***intrq*** register
- Enable the ADC interrupt request in bit 3 of ***inten*** register
- Enable global interrupt by issuing ***engint*** command

(3) Start AD conversion:
- Set ADC process control bit in the ***adcc*** register to start the conversion (set1 ***adcc***.6).

(4) Wait for the completion flag of AD conversion, by either:
- Waiting for the completion flag by using command “wait1 addc.6”; or
- Waiting for the ADC interrupt.

(5) Read the ADC result registers:
- Read ***adcrh*** and ***adcrl*** the result registers

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

5-16-6. Using the ADC

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

First, defining the selected pins:

```
PBC     =  0B_XXXX_0000;        // PB0 ~ PB3 as Input
PBPH    =  0B_XXXX_0000;        // PB0 ~ PB3 without pull-high
PBDIDR  =  0B_XXXX_1111;        // PB0 ~ PB3digital input is disabled
```

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

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

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

```
$  ADCM     10BIT, /32;         // 10-bit, /32
$  ADCM     10BIT, /16;         // 10-bit, /16
$  ADCM     10BIT, /8;          // 10-bit, /8
$  ADCM     8BIT, /8;           //  8-bit, /8
```

Then, start the ADC conversion:

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

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

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

The ADC can be disabled by using the following method:

```
$  ADCC     Disable;
```

or

```
   ADCC     =  0;
```

5-17 10-BIT PWM generator

5-17-1. PWM Waveform

A PWM output waveform (Fig. 5-17-1) has a time-base (T_{Period} = Time of Period) and a time with output high level (Duty Cycle). The frequency of the PWM output is the inverse of the period ($f_{PWM} = 1/T_{Period}$), the resolution of the PWM is the clock count numbers for one period (N bits resolution, $2^N \times T_{clock} = T_{Period}$).

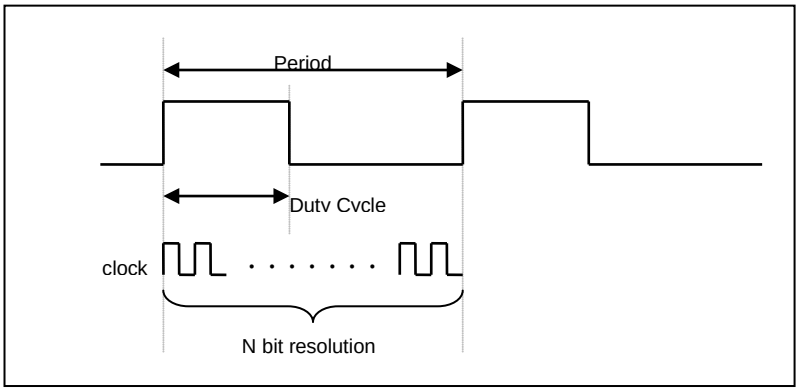


Fig. 5-17-1 PWM Output Waveform

5-17-2. Hardware and Timing Diagram

One 10-bit hardware PWM generator is built inside the MCS11; Fig. 5-17-2 shows its hardware diagram. The clock source can be IHRC or system clock and output pin can be PA2, PA3, PA4, PA6 or PA7 via ***pwmc*** register selection. The period of PWM waveform is defined in the PWM upper bond high and low registers, the duty cycle of PWM waveform is defined in the PWM duty high and low registers.

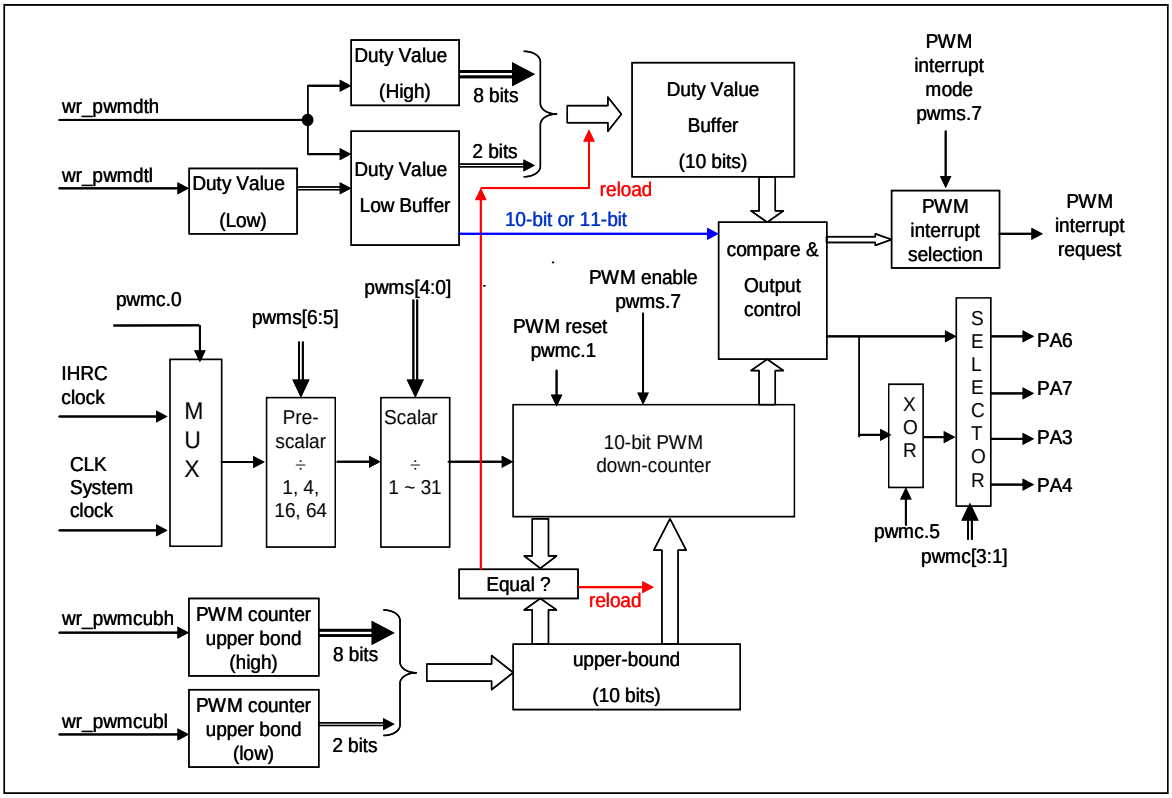


Fig. 5-17-2 Hardware Diagram of 10-bit PWM Generator

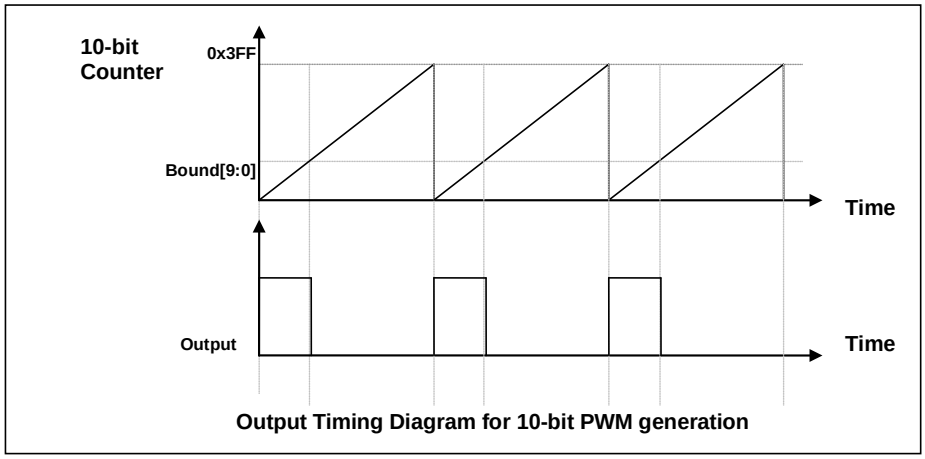


Fig. 5-17-3 Output Timing Diagram of 10-bit PWM Generator

5-17-3. Equations for 10-bit PWM Generator

If F_{IHRC} is the frequency of IHRC oscillator and IHRC is the chosen clock source for 10-bit PWM generator, the PWM frequency and duty cycle in time will be:

Frequency of PWM Output = F_{IHRC} ÷ [P × K × B]

Duty Cycle of PWM Output (in time) = (1/F_{IHRC}) * [DB ÷ CB]

Where, pwms[6:5] = **P** ; pre-scalar

pwms[4:0] = **K** ; scalar

Duty_Bound[9:0] = {pwmdth[7:0],pwmdtl[7:6]} = **DB**; duty bound

Counter_Bount[9:0] = {pwmcubh[7:0], pwmcubl[7:6]} = **CB**; counter bount

5-18 Input Pulse Capture

The feature of input Pulse Capture is useful in applications which requiring frequency and pulse measurement. Fig. 5-18-1 shows the hardware diagram of input Pulse Capture in MCS11, the time base of Pulse Capture module can be system clock CLK, IHRC and EOSC, the input signals for measurement can be comparator output, PA0, PA5, PA6, PB0 or PB7.

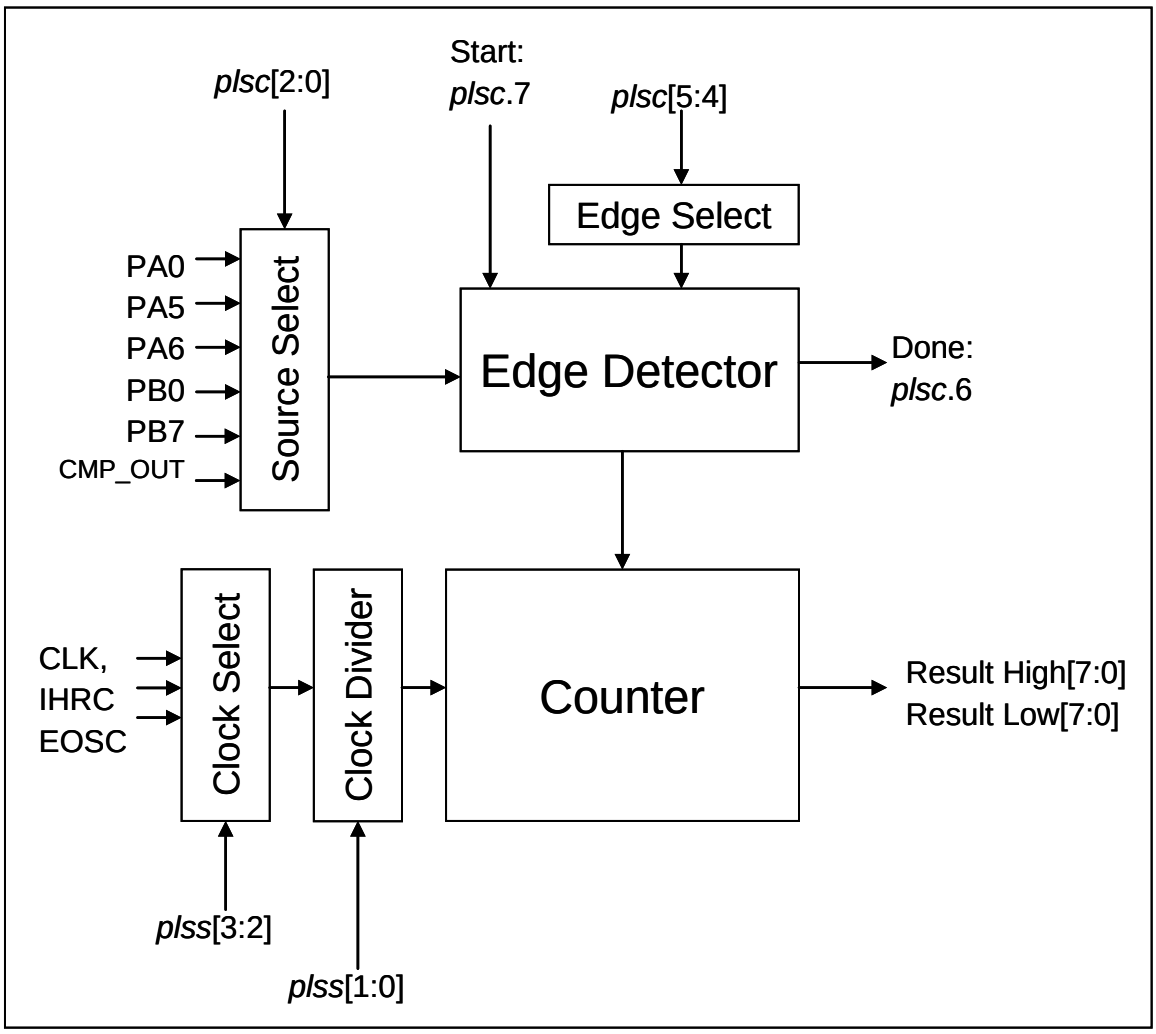


Fig. 5-18-1 Hardware Diagram of Input Pulse Capture

5-19 PWM Protection

For motor application, it's quite important to avoid ON state for both high side and low side simultaneously. MCS11 provides the hardware PWM protection circuit, shown as Fig. 5-19-1; there are two hardware PWM protection modules controlled by *pwmptr0* and *pwmptr1* registers individually to meet single-phase BLDC application. If the low site output is in active state and matches with the state of high site, the PWM generator will be disabled to force output in inactive state.

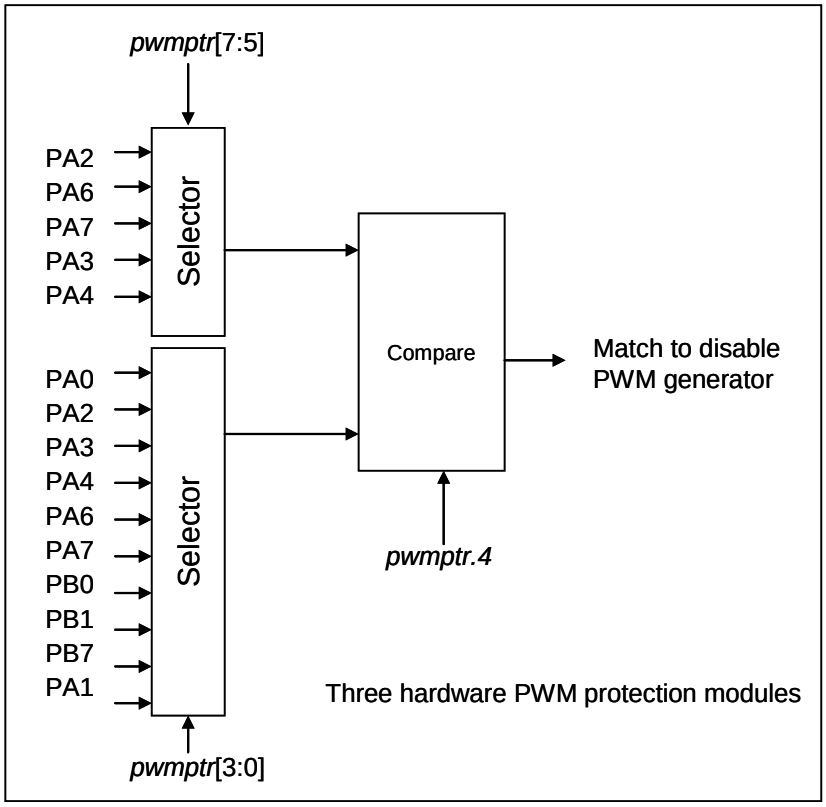


Fig. 5-19-1 Block Diagram of PWM protection

5-20 Multiplier

There is a 8x8 multiplier on-chip to enhance hardware capability in arithmetic function, its multiplication is an 8x8 unsigned operation and can be finished in one clock cycle. Before issuing the *mul* command, both and multiplicand and multiplicator must be put on ACC and register *mulop* (0x08); After *mul* command, the high byte result will be put on register *mulrh* (0x09) and low byte result on ACC. The hardware diagram of this multiplier is shown as Fig. 5-20-1.

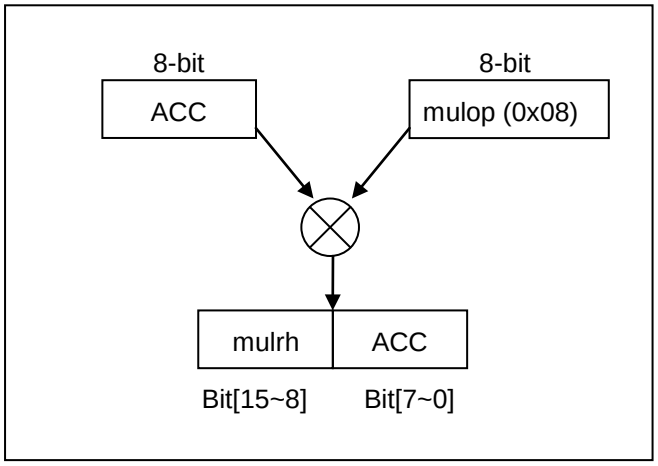


Fig. 5-20-1 Block diagram of hardware multiplier

5-21. General Purpose Comparator

5-21-1. General Purpose Comparator Hardware Diagram

One general purpose comparator is built inside the MCS11; Fig. 5-21-1 shows its hardware diagram. It can compare signals between two pins or with either internal reference voltage $V_{internal\ R}$ or internal band-gap reference voltage. The two signals to be compared, one will be the plus input of comparator and the other one is the minus input of comparator. For the minus input of comparator, it can be PB7, PB0, Internal band-gap 1.20 volt, internal reference voltage $V_{internal\ R}$, PA3 or PA4 selected by bit [3:1] of gpcc register, and for the plus input of comparator, it can be PB0 or $V_{internal\ R}$ selected by bit 0 of gpcc register. The comparator result can be enabled to output to PB0 directly, or sampled by rising edge of Time2 clock (TM2_CLK) which comes from Timer2 module. The output can be optional inversed the polarity by bit 4 of ***gpcc*** register, the comparator output can be used to request interrupt service or read out by gpcc register.

The comparator is disabled after power-on reset and can be enabled by setting gpcc.7=1, the comparator module can be put into power-down mode only when issuing ***stopsys*** command which will put MCS11 into power-down mode.

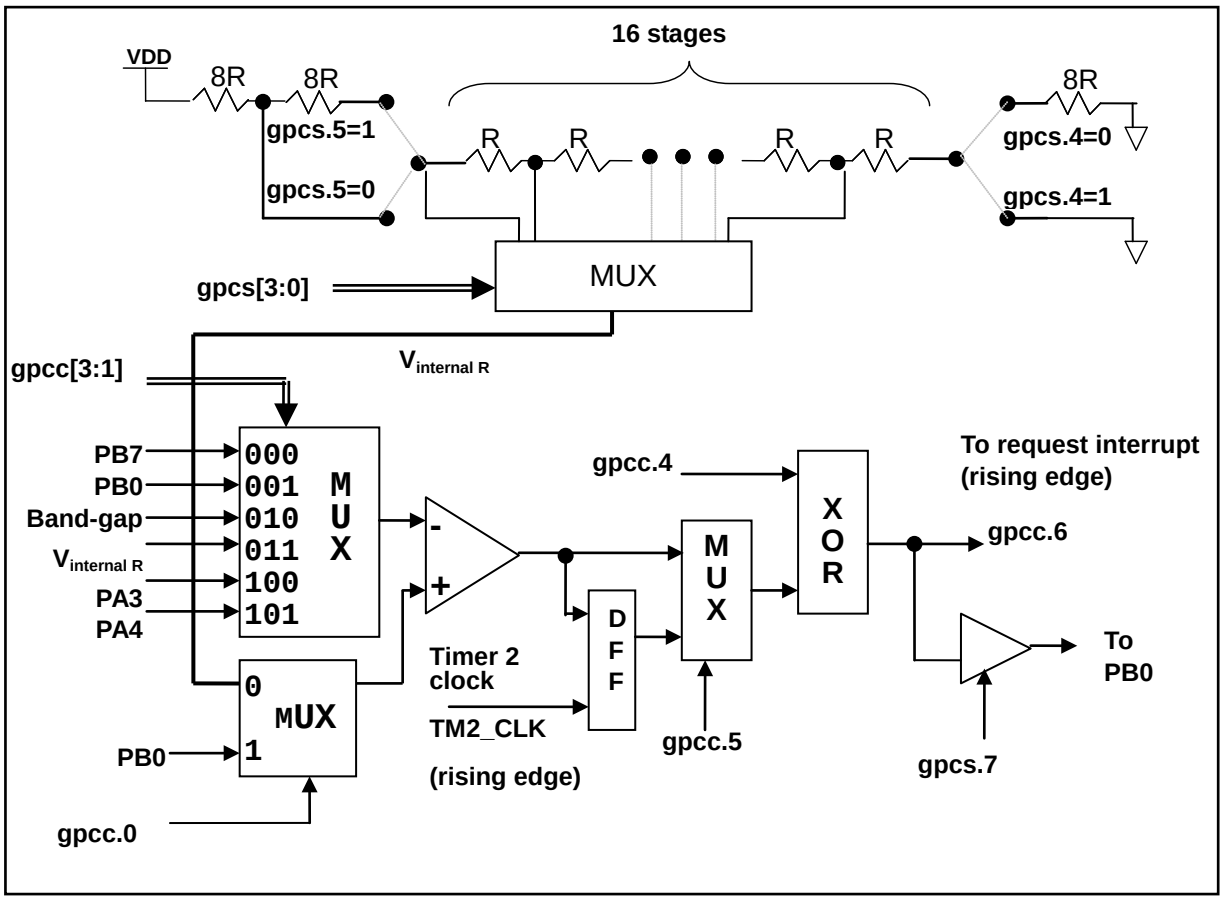


Fig. 5-21-1 Hardware diagram of general purpose comparator

5-21-2. Analog Inputs

A simplified circuit for the analog inputs is shown in the Fig. 5-21-2. All the analog input pins for general purpose comparator are shared function with a digital input which had reverse biased ESD protection diodes to VDD and GND, therefore, the analog input signal must be between VDD and GND.

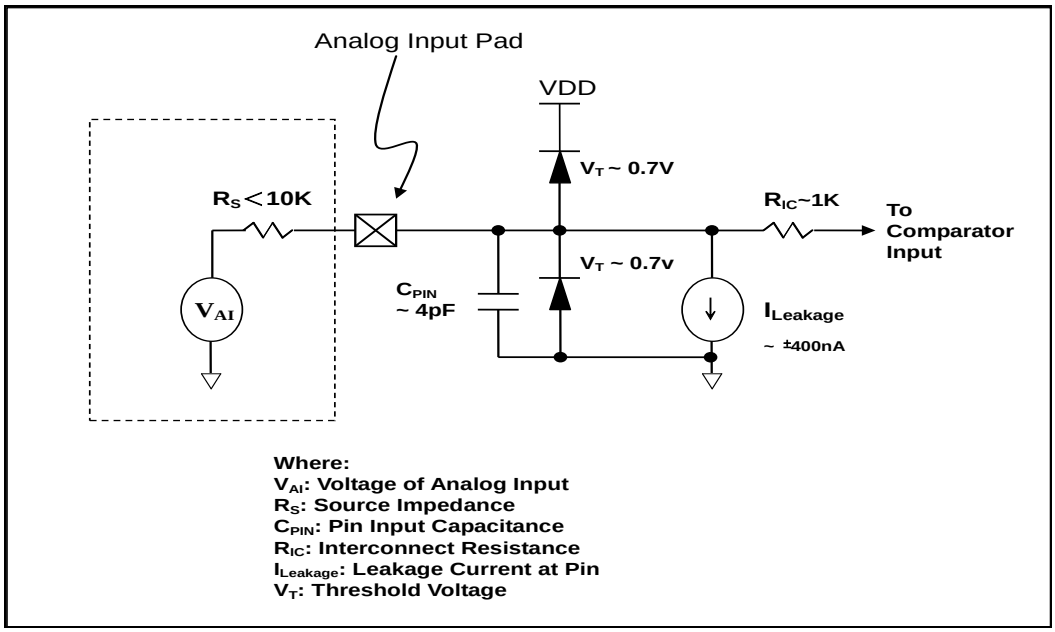


Fig. 5-21-2 Analog Input Model of General Purpose Comparator

5-21-3. Internal reference voltage ($V_{internal\ R}$)

The internal reference voltage $V_{internal\ R}$ is built by series resistance to provide different level of reference voltage, bit 4 and bit 5 of ***gpcs*** register are used to select the maximum and minimum values of $V_{internal\ R}$ and bit [3:0] of ***gpcs*** register are used to select one of the voltage level which is deivided-by-16 from the defined maximum level to minimum level. Fig. 5-21-3 to Fig. 5-21-6 shows four conditions to have different reference voltage $V_{internal\ R}$. By setting the ***gpcs*** register, the internal reference voltage $V_{internal\ R}$ can be ranged from (1/32)*VDD to (3/4)*VDD.

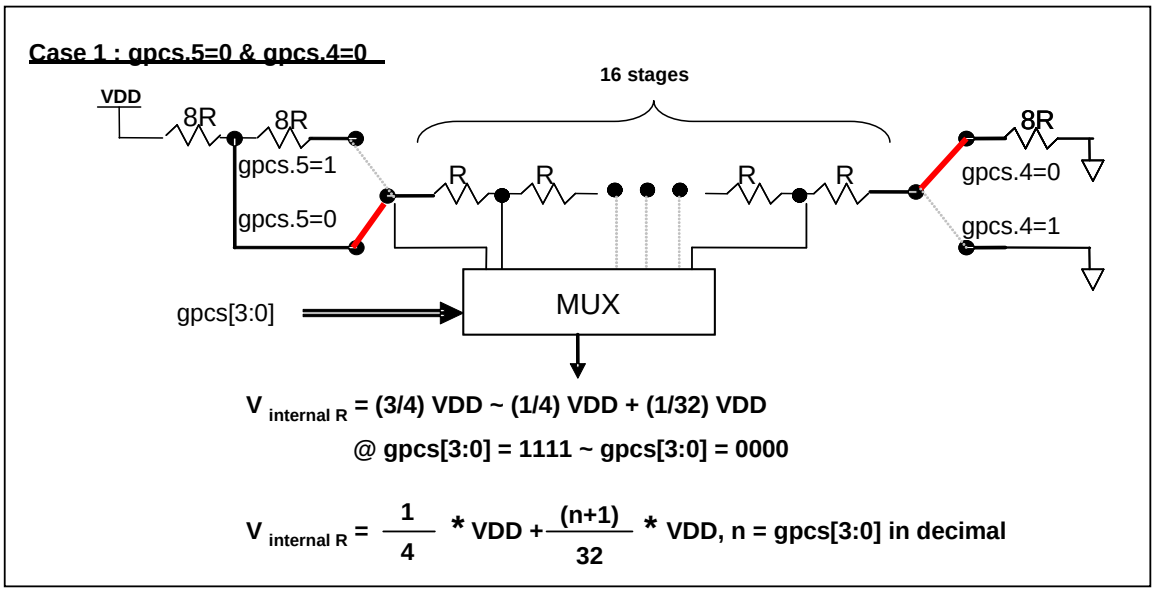


Fig. 5-21-3 $V_{internal\ R}$ hardware connection if gpcs.5=0 and gpcs.4=0

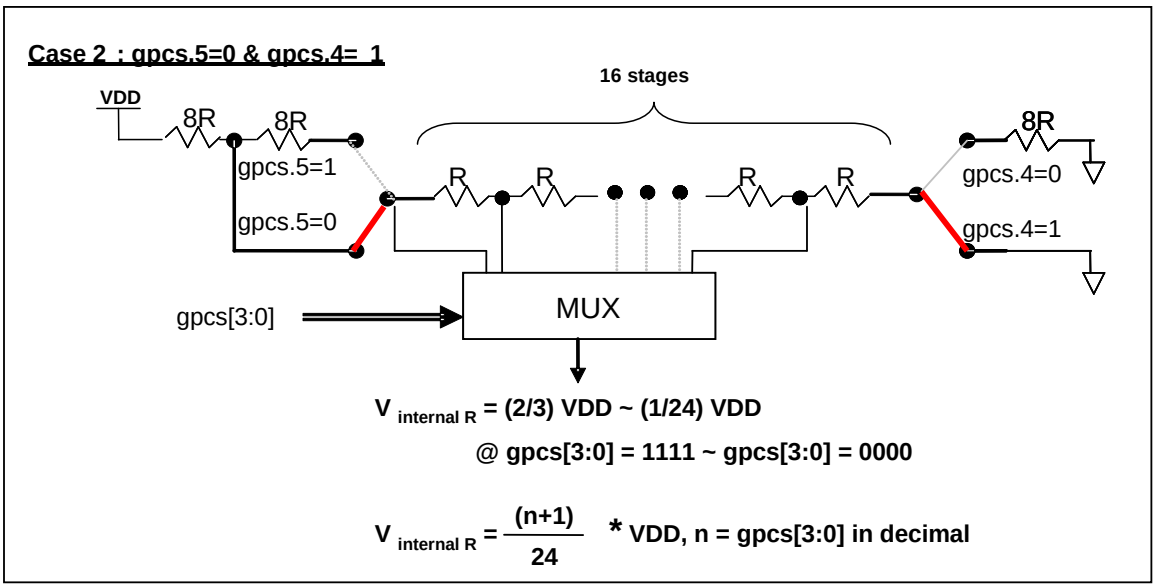


Fig. 5-21-4 $V_{internal\ R}$ hardware connection if gpcs.5=0 and gpcs.4=1

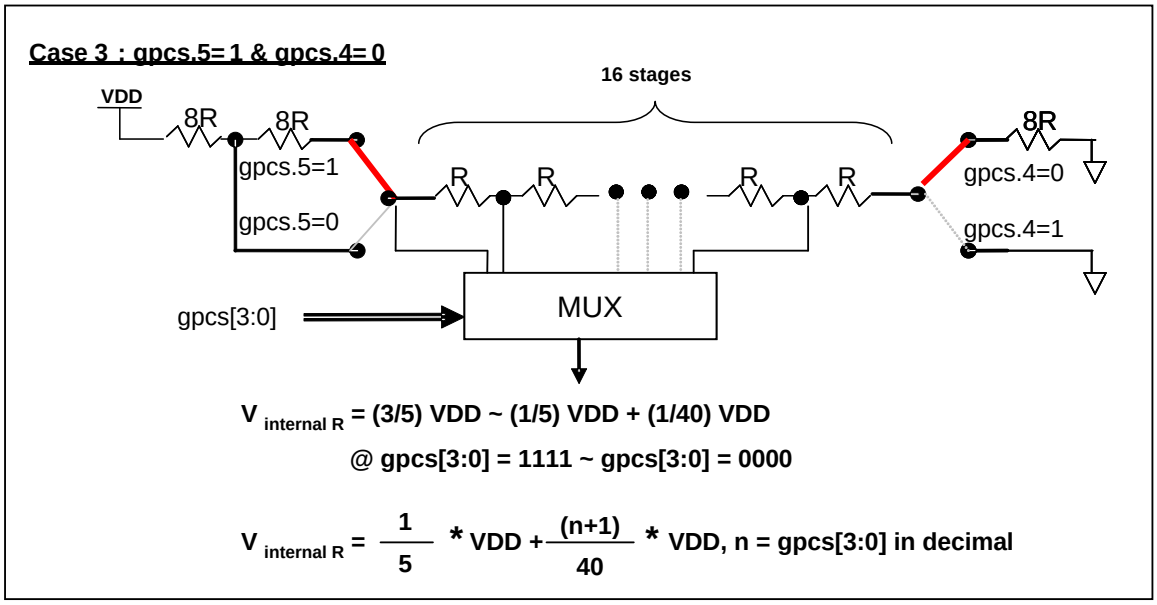


Fig. 5-21-5 $V_{internal\ R}$ hardware connection if gpcs.5=1 and gpcs.4=0

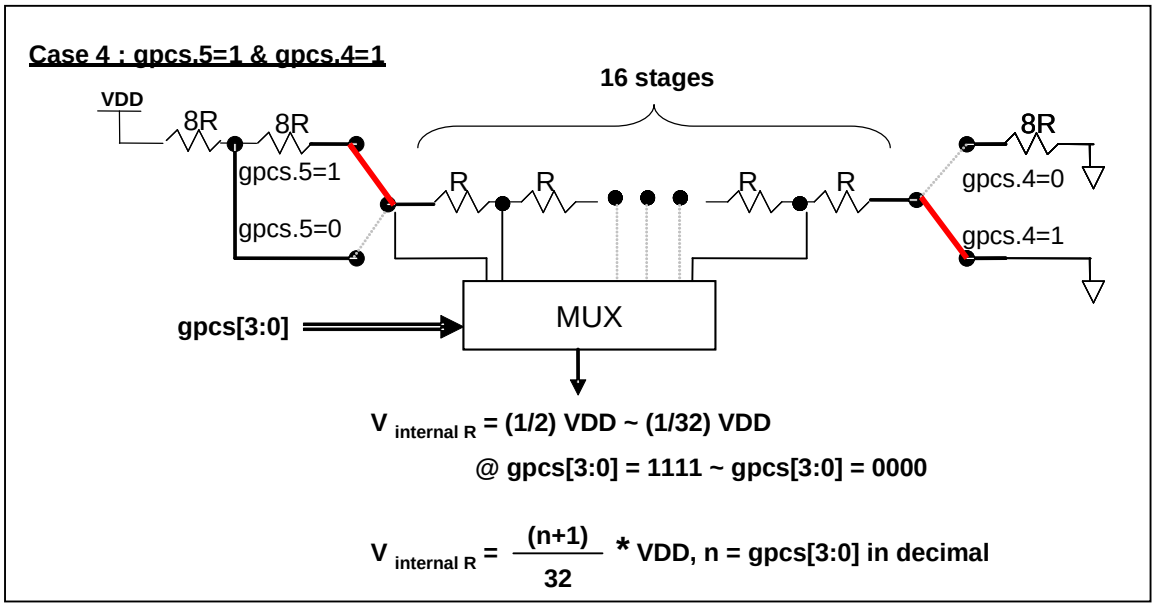


Fig. 5-21-6 $V_{internal\ R}$ hardware connection if gpcs.5=1 and gpcs.4=1

5-21-4. Synchronizing General Purpose Comparator Output to Timer2

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

5-21-5. Using the general purpose comparator

Case I:

Choosing PB7 as minus input and $V_{internal\ R}$ with (18/32)*VDD voltage level as plus input, the comparator result will be output to PB0, the comparator result will be output to PB0. $V_{internal\ R}$ is configured as Fig. 5-21-3 and gpcs [3:0] = 4b'1001 (n=9) to have $V_{internal\ R}$ = (1/4)*VDD + [(9+1)/32]*VDD = (18/32)*VDD.

```
gpcs    = 0b1_0_00_1001;    // output to PB0, V_internal R = VDD*(18/32)
gpcc    = 0b1_0_0_0_000_0;  // enable comp, - input: PB7, + input: V_internal R
pbdier  = 0b01111111;       // disable PB7 digital input to prevent leakage current
```

Case 2:

Choosing $V_{internal\ R}$ as minus input with (14/32)*VDD voltage level and PB0 as plus input, the comparator result will be inversed and without output to PB0. $V_{internal\ R}$ is configured as Fig. 5-21-6 and gpcs [3:0] = 4b'1101 (n=13) to have $V_{internal\ R}$ = [(13+1)/32]*VDD = (14/32)*VDD.

```
gpcs    = 0b0_1_1_1_1101;   // V_internal R = VDD*(14/32)
gpcc    = 0b1_0_0_1_011_1;  // Inverse output, - input: V_internal R, + input: PB0
pbdier  = 0b11111110;       // disable PB0 digital input to prevent leakage current
```

5-21-6. Using the comparator and band-gap 1.20V

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

.

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

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

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

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

Please refer to IDE utility for more information and sample code.

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

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

Bit	Reset	R/W	Description
7	0	R/W	FPP7 enable. This bit is used to enable FPP7. 0 / 1: disable / enable
6	0	R/W	FPP6 enable. This bit is used to enable FPP6. 0 / 1: disable / enable
5	0	R/W	FPP5 enable. This bit is used to enable FPP5. 0 / 1: disable / enable
4	0	R/W	FPP4 enable. This bit is used to enable FPP4. 0 / 1: disable / enable
3	0	R/W	FPP3 enable. This bit is used to enable FPP3. 0 / 1: disable / enable
2	0	R/W	FPP2 enable. This bit is used to enable FPP2. 0 / 1: disable / enable
1	0	R/W	FPP1 enable. This bit is used to enable FPP1. 0 / 1: disable / enable
0	1	R/W	FPP0 enable. This bit is used to enable FPP0. 0 / 1: disable / enable

6-3. Stack Pointer Register (*sp*), IO address = 0x02

Bit	Reset	R/W	Description
7 - 0	-	R/W	Stack Pointer Register. Read out the current stack pointer, or write to change the stack pointer.

6-4. 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	
1	1	R/W	Watch Dog Enable. 0 / 1: disable / enable	
0	0	R/W	Pin PA5/RESET# function. 0 / 1: PA5 / RESET#.	

6-5 Register Option Register (rop), IO address = 0x3e

Bit	Reset	R/W	Description
7 – 4	-	-	Reserved.
3	0	WO	PWMG resolution selection 0: 11-bit 1: 10-bit
2	0	WO	Option for Timer16 clock pre-divider selection. Please see the t16m register.
1	0	WO	Option for external interrupt 1 pin selection.
0	0	WO	Option for external interrupt 0 pin selection.

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

Bit	Reset	R/W	Description	
7	-	-	Reserved.	
6	0	R/W	Enable interrupt from Timer2. 0 / 1: disable / enable.	
5	0	R/W	Enable interrupt from PWM generator. 0 / 1: disable / enable.	
4	0	R/W	Enable Hall comparator interrupt	
3	0	R/W	Enable interrupt from ADC. 0 / 1: disable / enable.	
2	0	R/W	Enable interrupt from Timer16 overflow. 0 / 1: disable / enable.	
1	0	R/W	*rop*.0=0	Enable interrupt from PB0 pin. 0 / 1: disable / enable.
			rop.0=1	Enable interrupt from PB7 pin. 0 / 1: disable / enable.
0	0	R/W	*rop*.1=0	Enable interrupt from PA0 pin. 0 / 1: disable / enable.
			rop.1=1	Enable interrupt from PA5 pin. 0 / 1: disable / enable.
Where, *rop* is the 0x3e register optional register.				

6-7. Interrupt Request Register (*intrq*), IO address = 0x05

Bit	Reset	R/W	Description	
7	-	-	Reserved.	
6	-	R/W	Interrupt Request from Timer2, this bit is set by hardware and cleared by software. 0 / 1: No request / Request	
5	-	R/W	Interrupt Request from PWM generator, this bit is set by hardware and cleared by software. 0 / 1: No request / Request	
4	-	R/W	Interrupt Request from Hall comparator, this bit is set by hardware and cleared by software. 0 / 1: No request / Request	
3	-	R/W	Interrupt Request from ADC, this bit is set by hardware and cleared by software. 0 / 1: No request / Request	
2	-	R/W	Interrupt Request from Timer16, this bit is set by hardware and cleared by software. 0 / 1: No request / Request	
1	-	R/W	*rop*.0=0	Interrupt Request from PB0 pin, this bit is set by hardware and cleared by software. 0 / 1: No request / Request
			rop.0=1	Interrupt Request from PB7 pin, this bit is set by hardware and cleared by software. 0 / 1: No request / Request
0	-	R/W	*rop*.1=0	Interrupt Request from PA0 pin, this bit is set by hardware and cleared by software. 0 / 1: No Request / request
			rop.1=1	Interrupt Request from PA5 pin, this bit is set by hardware and cleared by software. 0 / 1: No Request / request
Where, *rop* is the 0x3e register optional register.				

6-8. Timer16 mode Register (*t16m*), IO address = 0x06

Bit	Reset	R/W	Description
7 - 5	000	R/W	Timer Clock source selection. 000: Timer16 is disabled. 001: CLK (system clock) 010: reserved 011: PA4 100: IHRC 101: EOSC 110: ILRC 111: PA0 Others: Timer16 is disabled.
4 – 3	00	R/W	*rop*.2=0 Timer16 clock pre-divider. 00: /1 01: /4 10: /16 11: /64 *rop*.2=1 Timer16 clock pre-divider. 00: /2 01: /8 10: /32 11: /128
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
Where, *rop* is the 0x3e register optional register.			

6-9. General Data register for IO (*gdio*), IO address = 0x07

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

6-10. Multiplier Operand Register (*mulop*), IO address = 0x08

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

6-11. Multiplier Result High Byte Register (*mulrh*), IO address = 0x09

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

6-12. External Oscillator setting Register (*eoscr*), IO address = 0x0a

Bit	Reset	R/W	Description
7	0	WO	Enable crystal oscillator. 0 / 1 : Disable / Enable
6 – 5	00	WO	External oscillator selection. 00 : reserved 01 : Low driving current, for lower frequency, ex: 32KHz crystal oscillator 10 : Middle driving current, for middle frequency, ex: 1MHz crystal oscillator 11 : High driving current, for higher frequency, ex: 4MHz crystal oscillator
4 – 1	-	-	Reserved. Please keep 0.
0	0	WO	Power down both band-gap and LVD hardware modules. 0 / 1 : Normal / Power-down

6-13. Internal High RC oscillator control Register (*ihrcr*), IO address = 0x0b

Bit	Reset	R/W	Description
7 – 0	00	WO	Bit [7:0] for frequency calibration of IHRC. This register is for system using only, please do NOT write this register.

6-14. Interrupt Edge Select Register (*integs*), IO address = 0x0c

Bit	Reset	R/W	Description
7 – 5	-	-	Reserved.
4	0	WO	Timer16 edge selection. 0 : rising edge of the selected bit to trigger interrupt 1 : falling edge of the selected bit to trigger interrupt
3 – 2	00	WO	PB0 or PB7 interrupt edge selection. 00 : both rising edge and falling edge of the selected bit to trigger interrupt 01 : rising edge of the selected bit to trigger interrupt 10 : falling edge of the selected bit to trigger interrupt 11 : reserved. Note: If rop.0=0, PB0 is selected; If rop.0=1, PB7 is selected.
1 – 0	00	WO	PA0 or PA5 interrupt edge selection. 00 : both rising edge and falling edge of the selected bit to trigger interrupt 01 : rising edge of the selected bit to trigger interrupt 10 : falling edge of the selected bit to trigger interrupt 11 : reserved. Note: If rop.1=0, PA0 is selected; If rop.1=1, PA5 is selected.

6-15. 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 low to prevent leakage current when external crystal oscillator is used. If this bit is set to low, PA7 can NOT be used to wake-up the system. 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 low to prevent leakage current when external crystal oscillator is used. If this bit is set to low, PA6 can NOT be used to wake-up the system. 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, wakeup is disabled when this bit is “1” and “0” is enabled.
4	1	WO	Enable PA4 digital input and wake-up event. 1 / 0 : enable / disable. This bit should be set to low when PA4 is assigned as AD input to prevent leakage current. 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 digital input and wake-up event. 1 / 0 : enable / disable. This bit should be set to low when PA3 is assigned as AD input to prevent leakage current. If this bit is set to low, PA3 can NOT be used to wake-up the system. Note: For ICE emulation, the function is disabled when this bit is “1” and “0” is enabled.
2	1	WO	Enable PA2 wake-up event. 1 / 0 : enable / disable. This bit can be set to low to disable wake-up from PA2 toggling. Note: For ICE emulation, wakeup is disabled when this bit is “1” and “0” is enabled.
1	1	WO	Enable PA1 wake-up event. 1 / 0 : enable / disable. This bit can be set to low to disable wake-up from PA1 toggling. Note: For ICE emulation, wakeup is disabled when this bit is “1” and “0” is enabled.
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.

6-16. Port B Digital Input Enable Register (*pbdier*), IO address = 0x0e

Bit	Reset	R/W	Description
7	1	WO	Enable PB7 digital input and wake-up event. 1 / 0 : enable / disable. This bit should be set to low when PB7 is assigned as AD input to prevent leakage current. 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 digital input and wake-up event. 1 / 0 : enable / disable. This bit should be set to low when PB6 is assigned as AD input to prevent leakage current. If this bit is set to low, PB6 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 PB5 digital input and wake-up event. 1 / 0 : enable / disable. This bit should be set to low when PB5 is assigned as AD input to prevent leakage current. If this bit is set to low, PB5 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.
4	1	WO	Enable PB4 digital input and wake-up event. 1 / 0 : enable / disable. This bit should be set to low when PB4 is assigned as AD input to prevent leakage current. If this bit is set to low, PB4 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 PB3 digital input and wake-up event. 1 / 0 : enable / disable. This bit should be set to low when PB3 is assigned as AD input to prevent leakage current. If this bit is set to low, PB3 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	WO	Enable PB2 digital input and wake-up event. 1 / 0 : enable / disable. This bit should be set to low when PB2 is assigned as AD input to prevent leakage current. 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 digital input and wake-up event. 1 / 0 : enable / disable. This bit should be set to low when PB1 is assigned as AD input to prevent leakage current. 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	Disable PB0 digital input, external interrupt and wake up event. 1 / 0 : enable / disable. This bit should be set to low when PB0 is assigned as AD input to prevent leakage current. If this bit is set to low, PB0 can NOT be used as external interrupt pin and to wake up the system during power-down mode. 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.

6-17. Port A Data Register (*pa*), IO address = 0x10

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

6-18. Port A Control Register (*pac*), IO address = 0x11

Bit	Reset	R/W	Description
7 - 0	8'h00	R/W	Port A control registers. This register is used to define input mode or output mode for each corresponding pin of port A. 0 / 1: input / output Please note that PA5 can be INPUT or OUTPUT LOW ONLY, the output state will be tri-state when PA5 is programmed into output mode with data 1.

6-19. Port A Pull-High Register (*paph*), IO address = 0x12

Bit	Reset	R/W	Description
7 - 0	8'h00	R/W	Port A pull-high register. This register is used to enable the internal pull-high device on each corresponding pin of port A and this pull high function is active only for input mode. 0 / 1 : disable / enable Please note that PA5 does NOT have pull-up resistor.

6-20. Port B Data Register (*pb*), IO address = 0x14

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

6-21. Port B Control Register (*pbc*), IO address = 0x15

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

6-22. Port B Pull-High Register (*pbph*), IO address = 0x16

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

6-23. ADC Control Register (*adcc*), IO address = 0x20

Bit	Reset	R/W	Description
7	0	R/W	Enable ADC function. 0/1: Disable/Enable.
6	-	R/W	ADC process control bit. Write “1” to start AD conversion, and the flag is cleared automatically when starting the AD conversion ; Read “1” to indicate the completion of AD conversion and “0” is in progressing.
5 – 2	0000	R/W	Channel selector. These four bits are used to select input signal for AD conversion. 0000: PB0/AD0, 0001: PB1/AD1, 0010: PB2/AD2, 0011: PB3/AD3, 0100: PB4/AD4, 0101: PB5/AD5, 0110: PB6/AD6, 0111: PB7/AD7 1000: PA3/AD8 1001: PA4/AD9 1111: Band-gap 1.20 volt reference voltage Others: reserved
1 - 0	-	-	Reserved. Please keep 0.

6-24. ADC Mode Register (*adcm*), IO address = 0x21

Bit	Reset	R/W	Description
7 – 5	000	WO	Bit Resolution of ADC. 000: 8-bit, AD 8-bit result [7:0] = ***adcrh***[7:0]. 001: 9-bit, AD 9-bit result [8:0] = { ***adcrh***[7:0], ***adcrl***[7] }. 010: 10-bit, AD 10-bit result [9:0] = { ***adcrh***[7:0], ***adcrl***[7:6] }. others: reserved,
4 – 1	000	WO	ADC clock source selection. 0000: sysclk/1, 0001: sysclk/2, 0010: sysclk/4, 0011: sysclk/8, 0100: sysclk/16, 0101: sysclk/32, 0110: sysclk/64, 0111: sysclk/128, Others: reserved.
0	-	-	Reserved

6-25. ADC Result High Register (*adcrh*), IO address = 0x22

Bit	Reset	R/W	Description
7 – 0	-	RO	These eight read-only bits will be the bit [9:2] of AD conversion result. The bit 7 of this register is the MSB of ADC result for any resolution.

6-26. ADC Result Low Register (*adcrl*), IO address = 0x23

Bit	Reset	R/W	Description
7 – 6	-	RO	These two bits will be the bit [1:0] of AD conversion result.
5 – 0	-	-	Reserved

6-27. Timer2 Control Register (*tm2c*), IO address = 0x3c

Bit	Reset	R/W	Description
7 – 4	0000	R/W	Timer2 clock selection. 0000 : disable 0001 : system clock 0010 : internal high RC oscillator (IHRC) 0011 : reserved 0100 : ILRC 0101 : reserved 011x : reserved 1000 : PA0 (rising edge) 1001 : ~PA0 (falling edge) 1010 : PB0 (rising edge) 1011 : ~PB0 (falling edge) 1100 : PA4 (rising edge) 1101 : ~PA4 (falling edge) **Notice:** In ICE mode and IHRC is selected for Timer2 clock, the clock sent to Timer2 does NOT be stopped, Timer2 will keep counting when ICE is in halt state.
3 – 0	-	-	Reserved, please keep 0.

6-28. Timer2 Counter Register (*tm2ct*), IO address = 0x3d

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

6-29. Timer2 Scalar Register (*tm2s*), IO address = 0x37

Bit	Reset	R/W	Description
7	-	-	Reserved.
6 – 5	00	WO	Timer2 clock pre-scalar. 00 : /1 01 : /4 10 : /16 11 : /64
4 – 0	00000	WO	Timer2 clock scalar.

6-30. Timer2 Bound Register (*tm2b*), IO address = 0x09

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

6-31. Hall Comparator Control Register (*hcc*), IO address = 0x2a

Bit	Reset	R/W	Description
7	0	R/W	Enable Hall comparator. 0 / 1 : disable / enable When this bit is set to enable, please also set the corresponding analog input pins to be digital disable to prevent IO leakage.
6	-	RO	Comparison result of Hall comparator 1 (HC_1), output: HC_Out1.
5	-	RO	Comparison result of Hall comparator 2 (HC_2), output HC_Out2.
4	-	RO	HC_Out
3	-	-	Reversed
2 – 0	0	R/W	Output to PA5 and interrupt source of the hall comparator 0x00: None output to PA5, interrupt source = HC_Out1 0x01: None output to PA5, interrupt source = HC_Out2 0x02: None output to PA5, interrupt source = HC_Out 0x03: HC_Out1 to PA5, interrupt source = HC_Out1 0x04: HC_Out2 to PA5, interrupt source = HC_Out2 0x05: HC_Out to PA5, interrupt source = HC_Out Others: reversed

6-32. Hall Comparator 1 Adjust Register (HC1A), IO address = 0x2b

Bit	Reset	R/W	Description
7	0	R/W	Hall comparator positive pin selection. 0: PA4 1: 1.2V Band-gap
6 – 5	00	R/W	Hall comparator negative pin selection. 00: PA0 01: PA1 10: PA2 11: 1.2V Band-gap
4 – 0	5'h00	R/W	Hall comparator 1 (HC_1) adjust bits.

6-33. Hall Comparator 2 Adjust Register (HC2A), IO address = 0x2c

Bit	Reset	R/W	Description
7 – 5	3'h00	R/W	Reversed
4 – 0	5'h00	R/W	Hall comparator 2 (HC_2) adjust bits.

6-34 PWM Generator control Register (*pwmc*), IO address = 0x30

Bit	Reset	R/W	Description
7	0	R/W	Enable PWM generator. 0 / 1 : disable / enable.
6	-	RO	Output of PWM generator.
5	0	R/W	Enable to inverse the polarity of PWM generator output. 0 / 1 : disable / enable.
4	0	R/W	PWM counter reset. Writing "1" to clear PWM counter and this bit will be self clear to 0 after counter reset.
3 – 1	0	R/W	Select PWM output pin. 000: disable 010: PA6 011: PA7 100: PA2 101: PA3 110: PA4 Others: reserved
0	0	R/W	Clock source of PWM generator. 0: system clock , 1: IHRC

6-35 PWM Generator Scalar Register (*pwms*), IO address = 0x31

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

6-36 PWM Counter Upper Bound High Register (*pwmcubh*), IO address = 0x1a

Bit	Reset	R/W	Description
7 - 0	8'h00	WO	Bit[10:3] of PWM counter upper bound.

6-37 PWM Counter Upper Bound Low Register (*pwmcubl*), IO address = 0x1b

Bit	Reset	R/W	Description
7 – 5	000	WO	Bit[2:0] of PWM counter upper bound.
4 – 0	-	-	Reserved

6-38 PWM Duty Value High Register (*pwmdth*), IO address = 0x32

Bit	Reset	R/W	Description
7 - 0	8'h00	WO	Duty values bit[10:3] of PWM generator.

6-39 PWM Duty Value Low Register (*pwmdtl*), IO address = 0x33

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

6-40 Pulse Capture Control Register (*plscc*), IO address = 0x34

Bit	Reset	R/W	Description
7	0	R/W	Start to do Pulse Capture. After writing this bit "1". It starts the Pulse Capture operation and clears this bit automatically after finishing the Pulse Capture operation
6	0	R/W	Pulse Capture overflow. This bit is set to "1" when the Pulse Capture counter overflows.
5	0	R/W	Pulse Capture front edge selection. 0: rising edge; 1:falling edge.
4	0	R/W	Pulse Capture back edge selection. 0: rising edge; 1:falling edge.
3	-	-	Reserved.
2 – 0	000	R/W	Sources for Pulse Capture. 000: PA0 001: PA5 010: PA6 011: PB0 100: PB7 101: comparator output Others: reserved.

6-41 Pulse Capture Scalar Register (*plscs*), IO address = 0x35

Bit	Reset	R/W	Description
7 – 4	-	-	Reserved.
3 – 2	00	WO	Pulse Capture clock source. 00: system clock 01: IHRC 10: EOSC (external oscillator clock)
1 – 0	00	WO	Pulse Capture clock divider. 00 : /1 01 : /4 10 : /16 11 : /64

6-42 Pulse Capture Result High Register (*plsrh*), IO address = 0x36

Bit	Reset	R/W	Description
7 – 0	-	RO	Pulse Capture result high.

6-43 Pulse Capture Result Low Register (*plsrl*), IO address = 0x37

Bit	Reset	R/W	Description
7 – 0	-	RO	Pulse Capture result low.

6-44 RESET Status Register (*rstst*), IO address = 0x25

Bit	Reset (FOR only)	R/W	Description
7	0	R/W	MCU had been reset by Watch-Dog time-out? This bit is set to high whenever reset occurs from watch-dog time-out, and reset only when writing “0” to clear this bit or POR (power-on-reset) happens. 0 / 1 : No / Yes..
6	0	R/W	MCU had been reset by invalid code? This bit is set to high whenever reset occurs from invalid instruction code, and reset only when writing “0” to clear this bit or POR (power-on-reset) happens. 0 / 1 : No / Yes..
5	0	-	Reserved. Please keep 0.
4	-	-	Reserved. Please keep 1.
3	-	R/W	MCU reset from external reset pin (PA5)? This bit is set to high whenever reset occurs from PA5 pin, and reset only when writing “0” to clear this bit or POR (power-on-reset) happens. 0 / 1 : No / Yes.
2	-	R/W	VDD had been lower than 4V? This bit is set to high whenever VDD under 4V and reset only when writing “0” to clear this bit or POR (power-on-reset) happens. 0 / 1 : No / Yes.
1	-	R/W	VDD had been lower than 3V? This bit is set to high whenever VDD under 3V and reset only when writing “0” to clear this bit or POR (power-on-reset) happens. 0 / 1 : No / Yes.
0	-	R/W	VDD had been lower than 2V? This bit is set to high whenever VDD under 2V and reset only when writing “0” to clear this bit or POR (power-on-reset) happens. 0 / 1 : No / Yes.

6-45 PWM Protect Register 0 (*pwmptr0*), IO address = 0x27 (Write once)

Bit	Reset	R/W	Description
7:5	000	R/W	PWM low site output destination. 000: disable 010: PA6 011: PA7 100: PA2 101: PA3 110: PA4 Others: reserved
4	0	R/W	PWM protect polarity. 0 / 1 : low / high.
3 - 0	0000	R/W	PWM high site selected pin. 0000: PA0 0001: PA2 0010: PA3 0011: PA4 0100: PA6 0101: PA7 0110: PB0 0111: PB1 1101: PB7 1111: PA1 Others: reserved

6-46 PWM Protect Register 1 (*pwmptr1*), IO address = 0x28 (Write once)

Bit	Reset	R/W	Description
7:5	000	R/W	PWM low site output destination. 000: disable 010: PA6 011: PA7 100: PA2 101: PA3 110: PA4 Others: reserved
4	0	R/W	PWM protect polarity. 0 / 1 : low / high.
3 - 0	0000	R/W	PWM high site selected pin. 0000: PA0 0001: PA2 0010: PA3 0011: PA4 0100: PA6 0101: PA7 0110: PB0 0111: PB1 1101: PB7 1111: PA1 Others: reserved

6-47 General Purpose Comparator Control Register (*gpcc*), IO address = 0x3e

Bit	Reset	R/W	Description
7	0	R/W	Enable general purpose comparator. 0 / 1 : disable / enable When this bit is set to enable, please also set the corresponding analog input pins to be digital disable to prevent IO leakage.
6	-	RO	Comparator result of general purpose comparator. 0: plus input < minus input 1: plus input > minus input
5	0	R/W	Selection the sampled source of comparator result 0: result output NOT sampled by TM2_CLK 1: result output sampled by TM2_CLK
4	0	R/W	Inverse the polarity of the comparator result output 0: polarity is NOT inversed 1: polarity is inversed
3 – 1	000	R/W	Selection the negative source of general purpose comparator. 000 : PB7 001 : PB0 010 : band-gap output 011 : internal R 100 : PA3 101 : PA4
0	0	R/W	Selection the positive source of general purpose comparator. 0 : internal R 1 : PB0

6-48 General Purpose Comparator Selection Register (*gpcs*), IO address = 0x22

Bit	Reset	R/W	Description
7	0	WO	General purpose comparator output enable (to PB0). 0 / 1 : disable / enable
6	0	WO	General purpose comparator enables to wake up system. 0 / 1 : disable / enable The system will be wake up from power down mode if the result goes high.
5	0	WO	Selection of high range of general purpose comparator.
4	0	WO	Selection of low range of general purpose comparator.
3 – 0	0000	WO	Selection the voltage level of general purpose comparator. 0000 (lowest) ~ 1111 (highest)

6-49 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. Fast wake-up is NOT supported when EOSC is enabled. 0: Normal wake-up. The wake-up time is 1024 ILRC clocks 1: Fast wake-up. (for The wake-up time is 128 CLKs (system clock) + oscillator stable time. If wake-up from STOPEXE suspend, there is no oscillator stable time; If wake-up from STOPSYS suspend, it will be IHRC or ILRC stable time from power-on. 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.
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.
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

6-50 FPPA Reset Register, IO address = 0x3f

Bit	Reset	R/W	Description
7	0	WO	Reset FPP7. This bit will be cleared automatically after resetting FPP7.
6	0	WO	Reset FPP6. This bit will be cleared automatically after resetting FPP6.
5	0	WO	Reset FPP5. This bit will be cleared automatically after resetting FPP5.
4	0	WO	Reset FPP4. This bit will be cleared automatically after resetting FPP4.
3	0	WO	Reset FPP3. This bit will be cleared automatically after resetting FPP3.
2	0	WO	Reset FPP2. This bit will be cleared automatically after resetting FPP2.
1	0	WO	Reset FPP1. This bit will be cleared automatically after resetting FPP1.
0	0	WO	Reset FPP0. This bit will be cleared automatically after resetting FPP0.

7. Instructions

Symbol	Description
ACC	Accumulator
a	Accumulator
sp	Stack pointer
flag	ACC status flag register
I	Immediate data
&	Logical AND
|	Logical OR
←	Movement
^	Exclusive logic OR
+	Add
−	Subtraction
~	NOT (logical complement, 1's complement)
∓	NEG (2's complement)
OV	Overflow (The operational result is out of range in signed 2's complement number system)
Z	Zero (If the result of ALU operation is zero, this bit is set to 1)
C	Carry (The operational result is to have carry out for addition or to borrow carry for subtraction in unsigned number system)
AC	Auxiliary Carry (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
pc2	Program counter for FPP2
pc3	Program counter for FPP3
pc4	Program counter for FPP4
pc5	Program counter for FPP5
pc6	Program counter for FPP6
pc7	Program counter for FPP7

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 ---

pushw word	Move the source data from the memory in *word* to memory that address specified in the stack pointer (*pushw word*). It needs 2T to execute this instruction. Example: *pushw word*; Result: [sp] ← *word* ; sp ← sp + 2 ; Affected flags: 『N』Z 『N』C 『N』AC 『N』OV Application Example: --- word data ; // declare data in RAM … *mov* a, 0x55 ; *mov* ld@data, a ; // move 0x55 to data (LSB) *mov* a, 0xaa ; *mov* hd@data, a ; // move 0xaa to data (MSB) *pushw* data ; // move (0xaa, 0x55) to stack memory … ---
pushw pcN	Store the program counter of Nth FPP unit to the memory which address is specified in the stack pointer of current executing FPP unit (*pushw* pcN). It needs 2T to execute this instruction. Example: *pushw* pc3; Result: [sp] of FPP0 ← pc of FPP3 ; sp of FPP0 ← sp of FPP0 + 2 ; Affected flags: 『N』Z 『N』C 『N』AC 『N』OV Application Example: --- fpp0_loop: … *pushw* pc1 ; // store PC1 to stack memory … *goto* fpp0_loop ; fpp1_loop: … … *goto* fpp1_loop ; ---

popw word	Move the memory data from the address specified in the stack pointer to the *word* memory (*popw word*). It needs 2T to execute this instruction. Example: *popw word*; Result: sp ← sp - 2 ; word ← [sp] ; Affected flags: 『N』Z 『N』C 『N』AC 『N』OV Application Example: --- word data0 ; // declare data0 in RAM word data1 ; // declare data1 in RAM … *mov* a, 0x55 ; *mov* ld@data0, a ; // move 0x55 to data0 (LSB) *mov* a, 0xaa ; *mov* hd@data0, a ; // move 0xaa to data0 (MSB) *pushw* data0 ; // move data (0xaa, 0x55) to stack memory (word) *popw* data1 ; // move (0xaa, 0x55) in stack memory to data1 (word) *mov* a, ld@ptr1 ; // ACC=0x55 *mov* a, hd@ptr1 ; // ACC=0xaa ---
popw pcN	Restore the program counter of the Nth FPP unit from the memory which address is specified in the stack pointer of current executing FPP unit (*popw* pcN). It needs 2T to execute this instruction. Example: *popw* pc3; Result: sp of FPP0 ← sp of FPP0 - 2 ; pc of FPP3 ← [sp] of FPP0 ; Affected flags: 『N』Z 『N』C 『N』AC 『N』OV Example : --- fpp0_loop: … *pushw* pc1 ; // store PC1 to stack memory *nop;* … *popw* pc1 ; // restore PC1 from stack memory … *goto* fpp0_loop ; fpp1_loop: … … *goto* fpp1_loop ; ---

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

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

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

7-5. Bit Operation Instructions

set0 IO.n	Set bit n of IO port to low Example: *set0* pa.5 ; Result: set bit 5 of port A to low Affected flags: 『N』Z 『N』C 『N』AC 『N』OV
set1 IO.n	Set bit n of IO port to high Example: *set1* pb.5 ; Result: set bit 5 of port B to high Affected flags: 『N』Z 『N』C 『N』AC 『N』OV
tog IO.n	Toggle bit state of bit n of IO port Example: *tog* pa.5 ; Result: toggle bit 5 of port A Affected flags: 『N』Z 『N』C 『N』AC 『N』OV
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
delay I	Delay the (N＋1) cycles which N is specified by the immediate data, the timing is based on the executing FPP unit. After the *delay* instruction is executed, the ACC will be zero. Example: *delay* 0x05; Result: Delay 6 cycles here Affected flags: 『N』 Z 『N』 C 『N』 AC 『N』 OV **Notes:** (1) Because ACC is the temporarily buffer for counting, please make sure that it will not be interrupted when executing this instruction. Otherwise, it may be not the expected delay time. (2) <u>This instruction is not supported in single FPP mode.</u>
delay a	Delay the (N＋1) cycles which N is specified by the content of ACC, the timing is based on the executing FPP unit. After the *delay* instruction is executed, the ACC will be zero. Example: *delay* a; Result: Delay 16 cycles here if ACC=0fh Affected flags: 『N』 Z 『N』 C 『N』 AC 『N』 OV **Notes:** (1) Because ACC is the temporarily buffer for counting, please make sure that it will not be interrupted when executing this instruction. Otherwise, it may be not the expected delay time. (2) <u>This instruction is not supported in single FPP mode.</u>
delay M	Delay the (N＋1) cycles which N is specified by the content of memory, the timing is based on the executing FPP unit. After the *delay* instruction is executed, the ACC will be zero. Example: *delay* M; Result: Delay 256 cycles here if M=ffh Affected flags: 『N』 Z 『N』 C 『N』 AC 『N』 OV Notes: (1) Because ACC is the temporarily buffer for counting, please make sure that it will not be interrupted when executing this instruction. Otherwise, it may be not the expected delay time. (2) <u>This instruction is not supported in single FPP mode.</u>
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
pmode n	Operational mode selection for each FPP unit Example: *pmode* 0; Result: FPP units bandwidth sharing is set to mode 0 Mode FPP0 ~ FPP7 bandwidth sharing 0: /2, /2 1: /2, /4, /4 2: /4, /2, /4 3: /2, /4, /8, /8 4: /4, /2, /8, /8 5: /8, /2, /4, /8 6: /4, /4, /4, /4 7: /8, /4, /4, /4, /8 8: /2, /8, /8, /8, /8 9: /4, /4, /4, /8, /8 10: /8, /2, /8, /8, /8 11: /2, /8, /8, /8, /16, /16 12: /16, /2, /8, /8, /8, /16 13: /4, /4, /8, /8, /8, /8 14: /8, /4, /4, /8, /8, /8 15: /4, /4, /4, /8, /16, /16 16: /8, /4, /4, /4, /16, /16 17: /16, /4, /4, /4, /8, /16 18: /2, /8, /8, /16, /16, /16, /16 19: /8, /2, /8, /16, /16, /16, /16 20: /16, /2, /8, /8, /16, /16, /16 21: /4 , /4, /4, /16, /16, /16, /16 22: /16, /4, /4, /4, /16, /16, /16 23: /4, /8, /8, /8, /8, /8, /8 24: /8, /2, /16, /16, /16, /16, /16, /16 25: /4, /8, /4, /8, /16, /16, /16, /16

	26: /8, /4, /4, /8, /16, /16, /16, /16 27: /2, /8, /16, /16, /16, /16, /16, /16 28: /4, /4, /8, /8, /16, /16, /16, /16 29: /16, /2, /8, /16, /16, /16, /16, /16 30: /8, /4, /4, /8, /16, /16, /16, /16 31: /8, /8, /8, /8, /8, /8, /8, /8 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

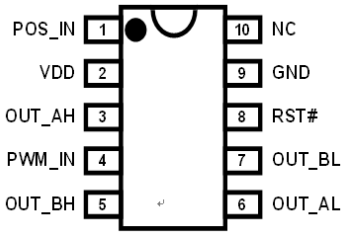
Instruction	Z	C	AC	OV	Instruction	Z	C	AC	OV	Instruction	Z	C	AC	OV
mov a, I	-	-	-	-	*mov* M, a	-	-	-	-	*mov* a, M	Y	-	-	-
mov a, IO	Y	-	-	-	*mov* IO, a	-	-	-	-	*nmov* M, a	-	-	-	-
nmov a, M	Y	-	-	-	*pushw word*	-	-	-	-	*pushw pcN*	-	-	-	-
popw word	-	-	-	-	*popw pcN*	-	-	-	-	*ldtabh* index	-	-	-	-
ldtabl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

8. Servo DC Motor Application

8-1. Key Features

- The detectable resolution of PWM input is adjustable to fit different application
- The minimum detectable resolution of PWM input is 3us
- Adjustable dead-band to fit different servo conformation
- Adjustable rotational angle to fit different servo conformation
- Adjustable PID parameters to fit different servo conformation
- 1-Channel A/D input port to read servo position
- The precision of servo position resolution is below one degree
- 3 pins servo connectors: VDD, GND, PWM input
- Operating Voltage: 3.3V ~ 5.5V

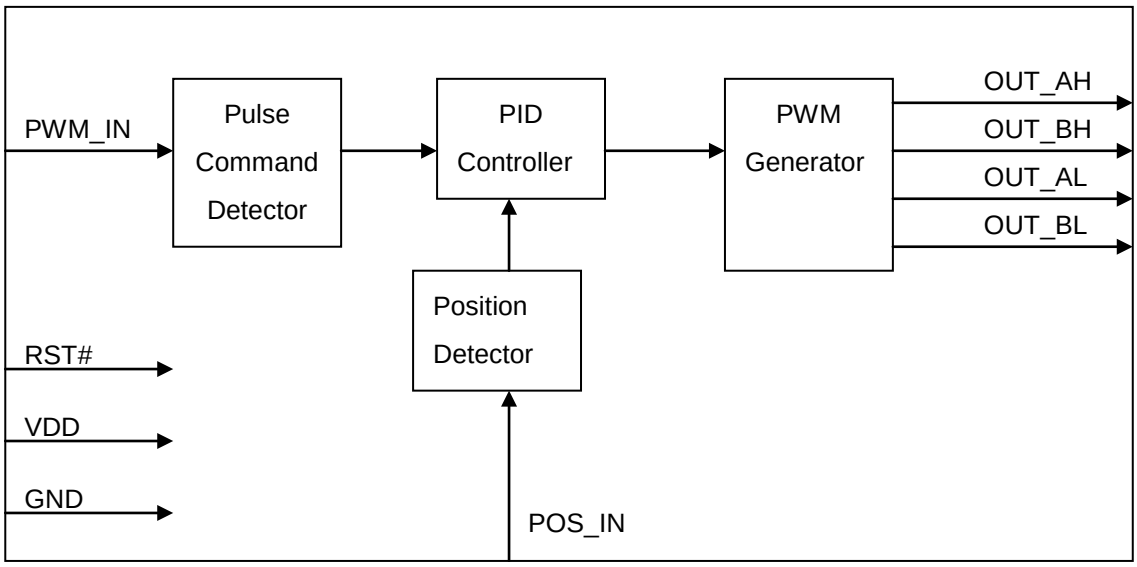
8-2. Pin Diagram and Pin Description for Servo Motor Application



MCS11 (MSOP10-118mil)

Pin No.	Pin Name	I/O	Description
1	POS_IN	Input	AD input to sense position
2	VDD	Input	5 Volt supply voltage
3	OUT_AH	Output	A output signal to control the high side of motor driver
4	PWM_IN	Input	PWM input control signal
5	OUT_BH	Output	B output signal to control the high side of motor driver
6	OUT_AL	Output	A output signal to control the low side of motor driver
7	OUT_BL	Output	B output signal to control the low side of motor driver
8	RST#	Input	RST# pin, must be tied VDD for normal operation
9	GND	Input	Ground
10	NC	--	NC

8-3. Block Diagram



8-4. Functional Description:

MCS11 is a servo motor controller and implemented in firmware, based on Proportional, Integral and Derivative (PID) control mechanism. By tuning the parameters of Kp, Ki and Kd, the MCS11 is able to fit different systems, thus it has the flexibility of adapting to different systems.

(1) Pulse command detector

This module is used to detect the external pulse width command PWM_IN for target position calculation. For RC type servo motor, the PWM_IN pulse width from 900us to 2100us corresponds to user adjustable rotational angle. This resolution of detectable PWM_IN pulse width is 3 usec. After detecting the pulse command the servo will drive the motor to the target position.

(2) Position detector

This module is used to detect the exact position of current motor position. This signal is used as feedback signal for PID controller. Mostly, it is a voltage signal with value decided by the resistance value of rotational resistor that is attached on the shaft of the motor gear.

(3) PID controller

The PID controller that is used in industry application popularly is implemented in this module. The controlled parameter is the motor position. The error between position command which is calculated by “pulse command detector” module and position feedback which is get by “position detector” module is used as input signal to the controller. The benefit of the PID controller is that the changed in system response can be observed by adjusting on or more gain values empirically. The dead-band is also adjustable to fit different conformation.

(4) PWM width Modulator

The pulse width modulator provides an energy efficient method by varying the average voltage applied to the motor winding by controlling the pulse width in every PWM cycle. This module output the four driving signals OUT_AH, OUT_BH, OUT_AL and OUT_BL that can be directly connected to the power stage of H-bridge.

9. POR for Servo Application

As shown in Fig 9-1, MCS11 generates a good power on reset (POR) signal when VDD rises from 0V to 5V in T_{POR} (max. 50ms) and rises through the voltage range of 0.7V to 1.6V in T_{FSV} (max. 10ms).

If there are a lot of abnormal power noises in VDD power-on time period and T_{POR} and T_{FSV} do not meet the specifications, MCS11 is not able to guarantee circuit initialization and may cause a malfunction.

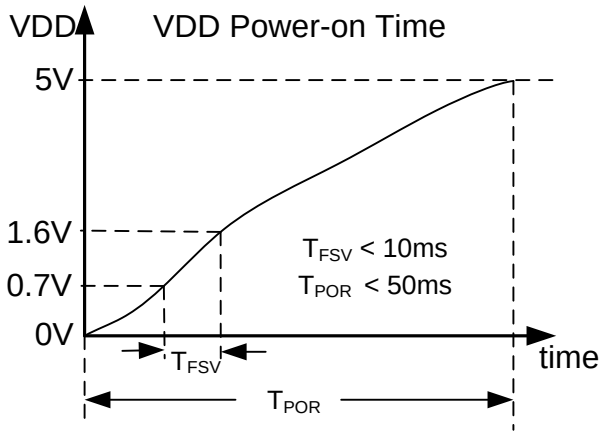


Fig. 9-1

During power-off as shown in Fig 9-2 and Fig 9-3, VDD has to be discharged to V_{PDRV} (max. is 0.7V) for the next power-on. In case VDD is more than V_{PDRV}, it is not recognized to do the next power-on.

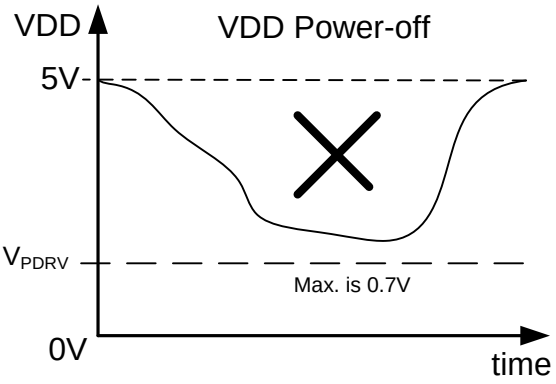


Fig. 9-2

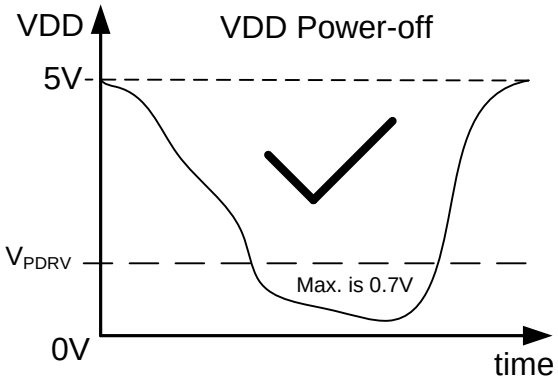


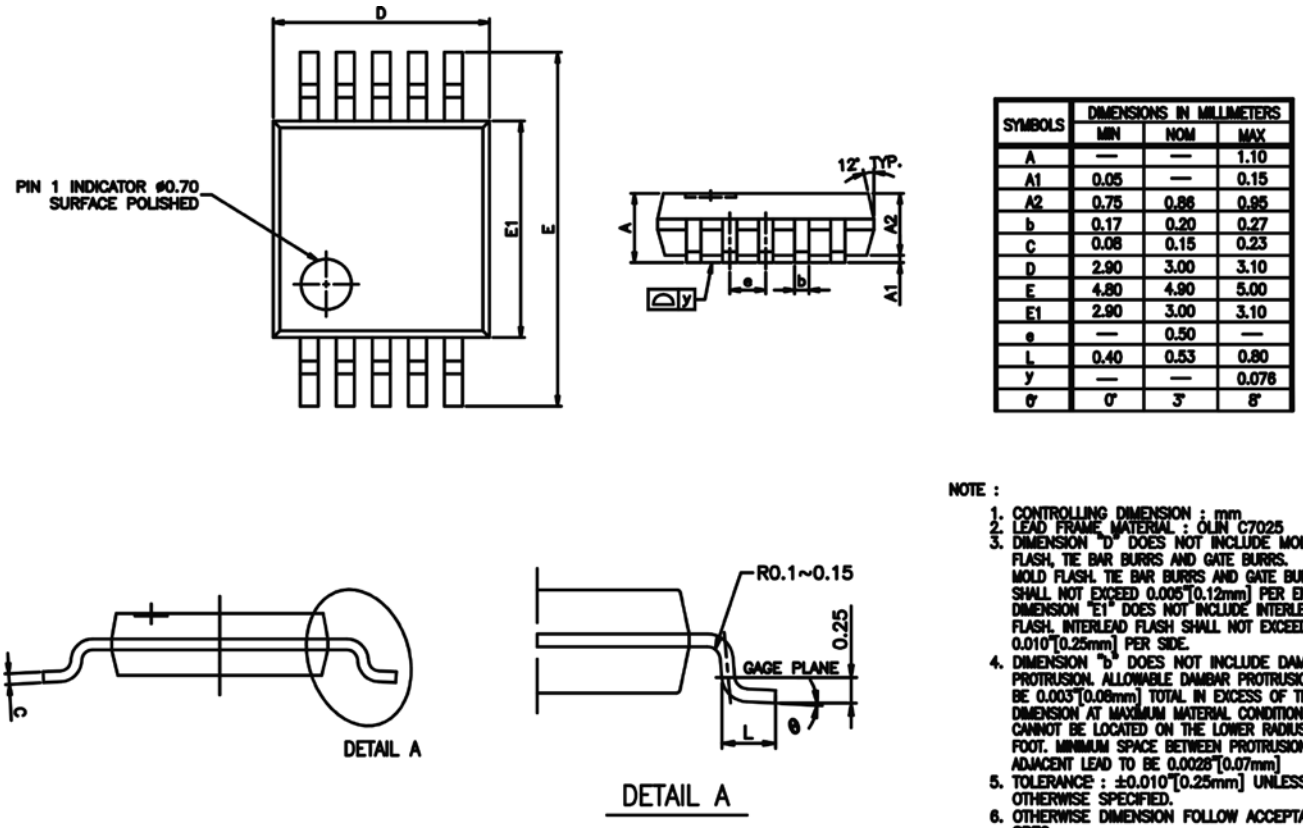
Fig. 9-3

NOTICE:

It may cause MCS11 malfunction, fail or crash when power-on and power-off do not meet the specifications. Another proper POR is needed in order to get rid of such status.

10. Package Information

10-1. MSOP10



SYMBOLS	DIMENSIONS IN MILLIMETERS		
	MIN	NOM	MAX
A	—	—	1.10
A1	0.05	—	0.15
A2	0.75	0.86	0.95
b	0.17	0.20	0.27
c	0.08	0.15	0.23
D	2.90	3.00	3.10
E	4.80	4.90	5.00
E1	2.90	3.00	3.10
e	—	0.50	—
L	0.40	0.53	0.80
y	—	—	0.076
θ	0°	3°	8°

NOTE :
1. CONTROLLING DIMENSION : mm
2. LEAD FRAME MATERIAL : OLIN C7025
3. DIMENSION "D" DOES NOT INCLUDE MOLD FLASH, TIE BAR BURRS AND GATE BURRS. MOLD FLASH, TIE BAR BURRS AND GATE BURRS SHALL NOT EXCEED 0.005"[0.12mm] PER END DIMENSION "E1" DOES NOT INCLUDE INTERLEAD FLASH. INTERLEAD FLASH SHALL NOT EXCEED 0.010"[0.25mm] PER SIDE.
4. DIMENSION "b" DOES NOT INCLUDE DAMBAR PROTRUSION. ALLOWABLE DAMBAR PROTRUSION SHALL BE 0.003"[0.08mm] TOTAL IN EXCESS OF THE "b" DIMENSION AT MAXIMUM MATERIAL CONDITION. DAMBAR CANNOT BE LOCATED ON THE LOWER RADIUS OR THE FOOT. MINIMUM SPACE BETWEEN PROTRUSION AND AN ADJACENT LEAD TO BE 0.0028"[0.07mm]
5. TOLERANCE : ±0.010"[0.25mm] UNLESS OTHERWISE SPECIFIED.
6. OTHERWISE DIMENSION FOLLOW ACCEPTABLE SPEC.
7. REFERENCE DOCUMENT : JEDEC SPEC MO-187