MCU401

Touch Controller

Data Sheet

Version 0.00 Apr. 20, 2018

Table of Contents

Revision History:

Revision	Date	Description
0.00	2018/04/20	Preliminary version

Features

System Features

- Clock sources: internal high RC oscillator and internal low RC oscillator
- Band-gap circuit to provide 1.20V Band-gap voltage
- One hardware 16-bit timer (TIMER16)
- One hardware 8-bit timer with PWM generator (6/8 bit mode, TIMER2)
- Support fast wake-up
- Eight Levels of LVR reset: 4.0V, 3.5V, 3.0V, 2.75V, 2.5V, 2.2V, 2.0V, 1.8V
- 15 IO pins with optional pull-high resistor
- 11 IO pins can be selected as touch pad individually.
- There are 6 interrupt sources：
  - ✧ Two external interrupt pins INT0 and INT1
  - ✧ Two interrupt lines for TIMER16 and TIMER2 individually
  - ✧ Two interrupt lines for touch function (TK_OV and TK_END)
- Operating voltage range：2.3V ~ 5.5V

CPU Features

- Operating modes：One processing unit mode
- 1.5KW OTP program memory
- 96 Bytes data RAM
- Most instructions are 1T execution cycle
- Programmable stack pointer to provide adjustable stack level (Using 2 bytes SRAM for one stack level)
- Direct and indirect addressing modes for data and instructions
- All data memories are available for use as an index pointer
- Separated IO and memory space

General Description and Block Diagram

The MCU401 is a fully static, OTP-based touch controller; it employs RISC architecture and most the instructions are executed in one cycle except that few instructions are two cycles that handle indirect memory access. Besides touch controller, 1.5KW bits OTP program memory and 96 bytes data SRAM are inside, one hardware 16-bit timer and one hardware 8-bit Timer2 with PWM generation are also provided in the MCU401.

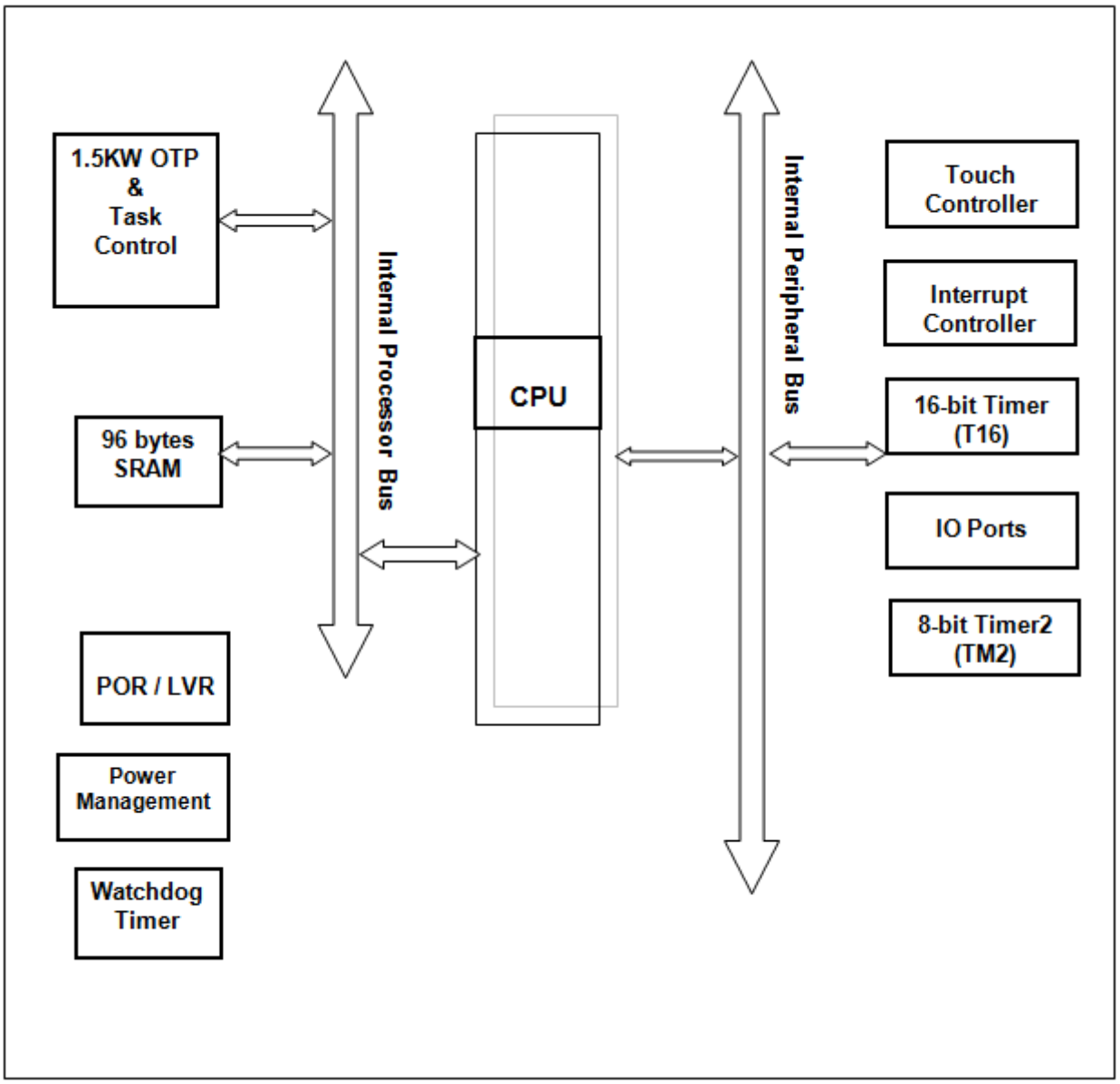
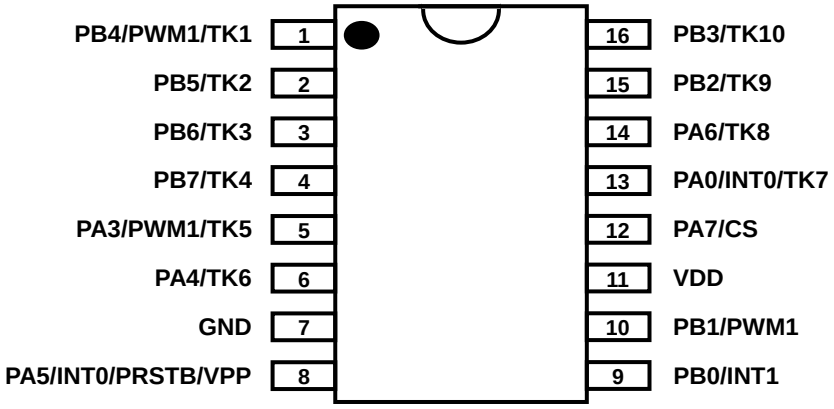


Fig. 1: MCU401 Block Diagram

Pin Definition and Functional Description



MCU401-S16 (SOP16-150mil)

Pin Name	Pin & Buffer Type	Description
PA6 / TK8	IO ST / CMOS / Analog	This pin can be used as: (1) Bit 6 of port A. It can be configured as digital input, two-state output with pull-up resistor by software independently (2) Touch Key 8 When this pin is configured as analog input, please use bit 6 of register ***padier*** to disable the digital input to prevent leakage current.
PA5 / INT0 / PRSTB / VPP	IO ST / CMOS	This pin can be used as: (1) Bit 5 of port A. It can be configured as input with optional pull-up resistor or open-drain output pin. (2) Optional external interrupt line 0. Both rising edge and falling edge are accepted to request interrupt service. (3) External reset pin. (4) VPP for OTP programming.
PA4 / TK6	IO ST / CMOS / Analog	This pin can be used as: (1) Bit 4 of port A. It can be configured as digital input, two-state output with pull-up resistor by software independently (2) Touch Key 6 When this pin is configured as analog input, please use bit 4 of register ***padier*** to disable the digital input to prevent leakage current.
PA3 / PWM1 / TK5	IO ST / CMOS / Analog	This pin can be used as: (1) Bit 3 of port A. It can be configured as digital input, two-state output with pull-up resistor by software independently (2) PWM output of Timer2. (3) Touch Key 5 When this pin is configured as analog input, please use bit 3 of register ***padier*** to disable the digital input to prevent leakage current.

Pin Name	Pin & Buffer Type	Description
PA0 / INT0 / TK7	IO ST / CMOS / Analog	This pin can be used as: (1) Bit 0 of port A. It can be configured as digital input, two-state output with pull-up resistor by software independently (2) Optional external interrupt line 0. Both rising edge and falling edge are accepted to request interrupt service. (3) Touch Key 7 When this pin is configured as analog input, please use bit 0 of register ***padier*** to disable the digital input to current leakage current.
PB0 / INT1	IO ST / CMOS	This pin can be used as: (1) Bit 0 of port B. It can be configured as digital input, two-state output with pull-up resistor by software independently (2) External interrupt line 1. Both rising edge and falling edge are accepted to request interrupt service.
PB1 / PWM1	IO ST / CMOS	This pin can be used as: (1) Bit 1 of port B. It can be configured as digital input, two-state output with pull-up resistor by software independently (2) PWM output of Timer2
PB2 / TK9	IO ST / CMOS / Analog	This pin can be used as: (1) Bit 2 of port B. It can be configured as digital input, two-state output with pull-up resistor by software independently (2) Touch Key 9 When this pin is configured as analog input, please use bit 2 of register ***pbdier*** to disable the digital input to prevent leakage current.
PB3 / TK10	IO ST / CMOS / Analog	This pin can be used as: (1) Bit 3 of port B. It can be configured as digital input, two-state output with pull-up resistor by software independently (2) Touch Key 10 When this pin is configured as analog input, please use bit 3 of register ***pbdier*** to disable the digital input to prevent leakage current.
PB4 / PWM1 / TK1	IO ST / CMOS / Analog	This pin can be used as: (1) Bit 4 of port B. It can be configured as digital input, two-state output with pull-up resistor by software independently (2) PWM output of Timer2. (3) Touch Key 1 When this pin is configured as analog input, please use bit 4 of register ***pbdier*** to disable the digital input to prevent leakage current.

Pin Name	Pin & Buffer Type	Description
PB5 / TK2	IO ST / CMOS / Analog	This pin can be used as: (1) Bit 5 of port B. It can be configured as digital input, two-state output with pull-up resistor by software independently (2) Touch Key 2 When this pin is configured as analog input, please use bit 5 of register pbdier to disable the digital input to prevent leakage current.
PB6 / TK3	IO ST / CMOS / Analog	This pin can be used as: (1) Bit 6 of port B. It can be configured as digital input, two-state output with pull-up resistor by software independently (2) Touch Key 3 When this pin is configured as analog input, please use bit 6 of register pbdier to disable the digital input to prevent leakage current.
PB7 / TK4	IO ST / CMOS / Analog	This pin can be used as: (1) Bit 7 of port B. It can be configured as digital input, two-state output with pull-up resistor by software independently (2) Touch Key 4 When this pin is configured as analog input, please use bit 7 of register pbdier to disable the digital input to prevent leakage current.
PA7 / CS	IO ST / CMOS / Analog	(1) Bit 7 of port A. It can be configured as digital input, two-state output with pull-up resistor by software independently (2) External Capacitor When this pin is configured as CS, the input function of this pin is disabled to prevent leakage current regardless of the setting of the bit 7 of register padier.
VDD		Positive power
GND		Ground
Notes: IO: Input/Output; ST: Schmitt Trigger input; Analog: Analog input pin; CMOS: CMOS voltage level		

Device Characteristics

DC/AC Characteristics

Symbol	Description	Min.	Typ	Max.	Unit	Conditions
V_{DD}	Operating Voltage	2.3		5.5	V	
f_{SYS}	System clock (CLK)* =			8M		
	IHRC/2	0		4M	Hz	$V_{DD} \geqq$ 3.5V
	IHRC/4	0				$V_{DD} \geqq$ 2.5V
	ILRC		58K			V_{DD} =3V
I_{OP}	Operating Current		0.6	2	mA	f_{SYS}=4MHz, 5V
I_{PD}	Power Down Current		1.4		uA	
	(by stopsys command)		1.0			V_{DD} =5V
						V_{DD} =3.3V
I_{PS}	Power Save Current		5		uA	
	(by stopexe command)					V_{DD} =3.3V
	*Disable IHRC					
V_{IL}	Input low voltage for IO lines	0		0.2V_{DD}	V	
V_{IH}	Input high voltage for IO lines	0.7 V_{DD}		V_{DD}	V	
I_{OL}	IO lines sink current(Normal)				mA	V_{DD}=5.0V, V_{OL}=0.5V
	PB0/PB1		43			
	PA7		26			
	PA5		5			
	Others		13			
	IO lines sink current(Low)					
	PB0/PB1		8			
	PA7		9			
	PA5		5			
	Others		5			
I_{OH}	IO lines drive current(Normal)				mA	V_{DD}=5.0V, V_{OH}=4.5V
	PB1		30			
	PA7		18			
	PA5		0			
	Others		10			
	IO lines drive current(Low)					
	PA7		5			
	PA5		0			
	Others		3			
V_{IN}	Input voltage	-0.3		V_{DD}+0.3	V	
$I_{INJ\ (PIN)}$	Injected current on pin		1		uA	V_{DD} +0.3$\geqq V_{IN} \geqq$ -0.3
R_{PH}	Pull-high Resistance		110		KΩ	V_{DD}=5.0V
			200			V_{DD}=3.3V

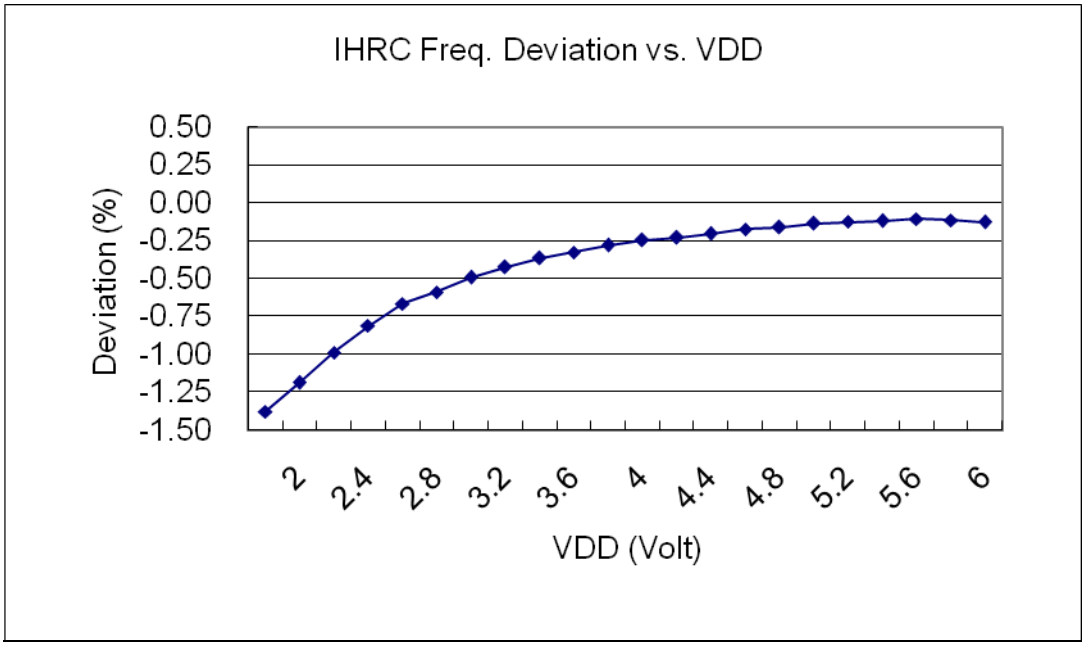
Symbol	Description	Min.	Typ	Max.	Unit	Conditions
V_{LVR}	Low Voltage Detect Voltage *		4.0 3.5 3.0 2.75 2.5 2.2 2.0 1.8		V	
f_{IHRC}	Frequency of IHRC after calibration *	15.2	16*	16.8	MHz	V_{DD} =2.3V~5.5V, -20°C <Ta<70°C*
f_{ILRC}	Frequency of ILRC *		58		KHz	
t_{INT}	Interrupt pulse width	30			ns	V_{DD} = 5.0V
t_{WDT}	Watchdog timeout period		8192 16384 65536 26214 4		ILRC clock period	misc[1:0]=00 (default) misc[1:0]=01 misc[1:0]=10 misc[1:0]=11
t_{SBP}	System boot-up period from power-on (Fast boot up) System boot-up period from power-on (Slow boot up)		780 47		us ms	@ V_{DD} =5V @ V_{DD} =5V
t_{RST}	External reset pulse width	120			us	@ V_{DD} =5V

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

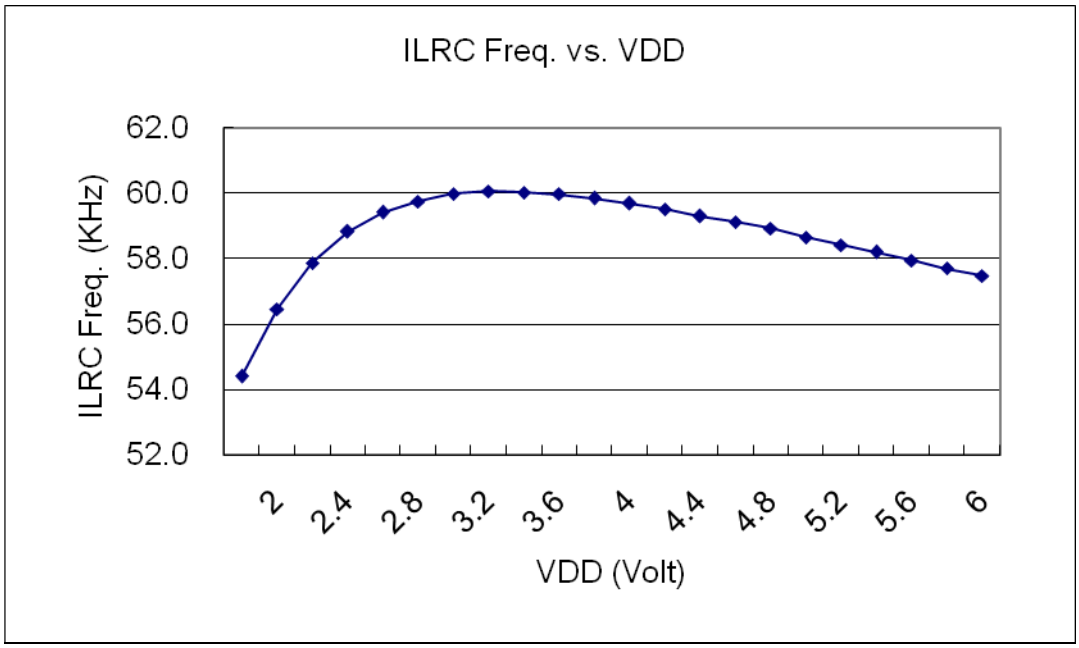
Absolute Maximum Ratings

- Operating Voltage …… 2.3V ~ 5.5V
- Input Voltage …… -0.3V ~ V_{DD} + 0.3V
- Operating Temperature …… -20°C ~ 70°C
- Storage Temperature …… -50°C ~ 125°C
- Junction Temperature …… 150°C

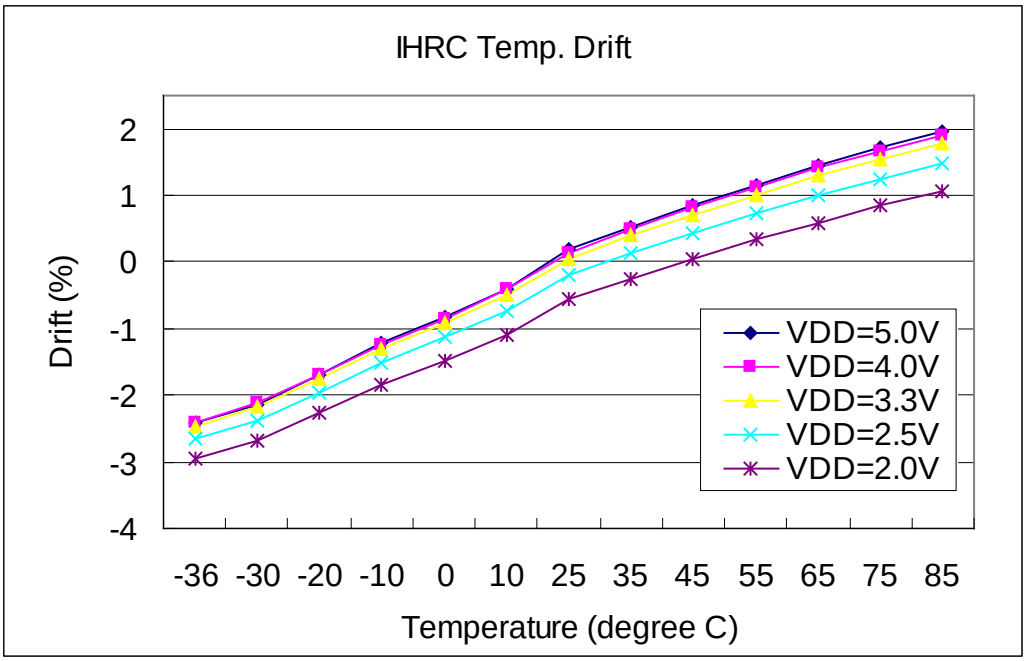
Typical IHRC Frequency vs. VDD (calibrated to 16MHz)



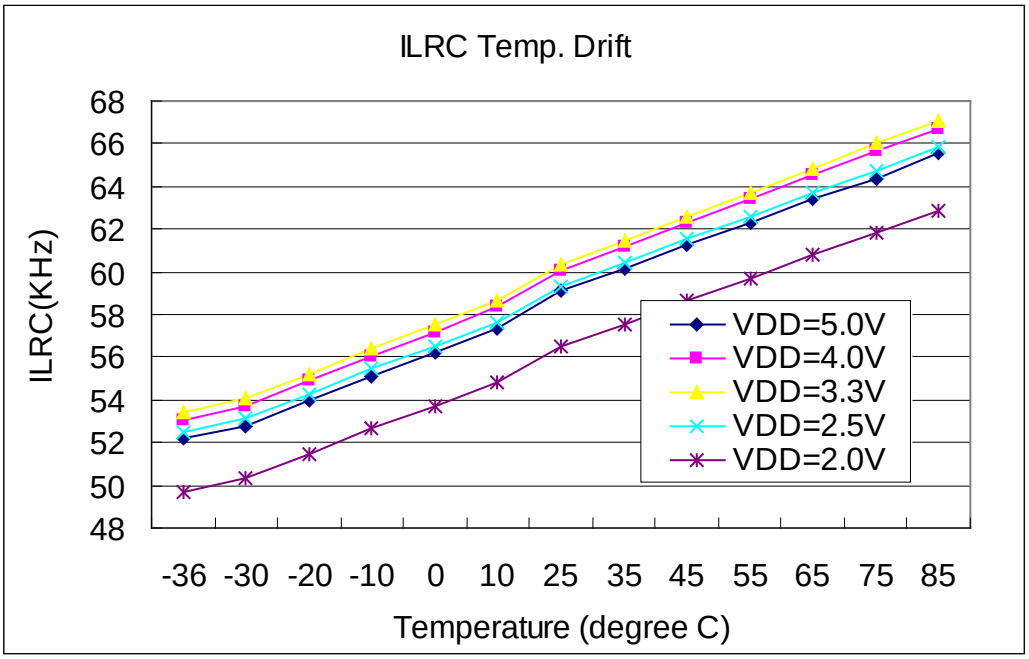
Typical ILRC Frequency vs. VDD



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



Typical ILRC Frequency vs. Temperature

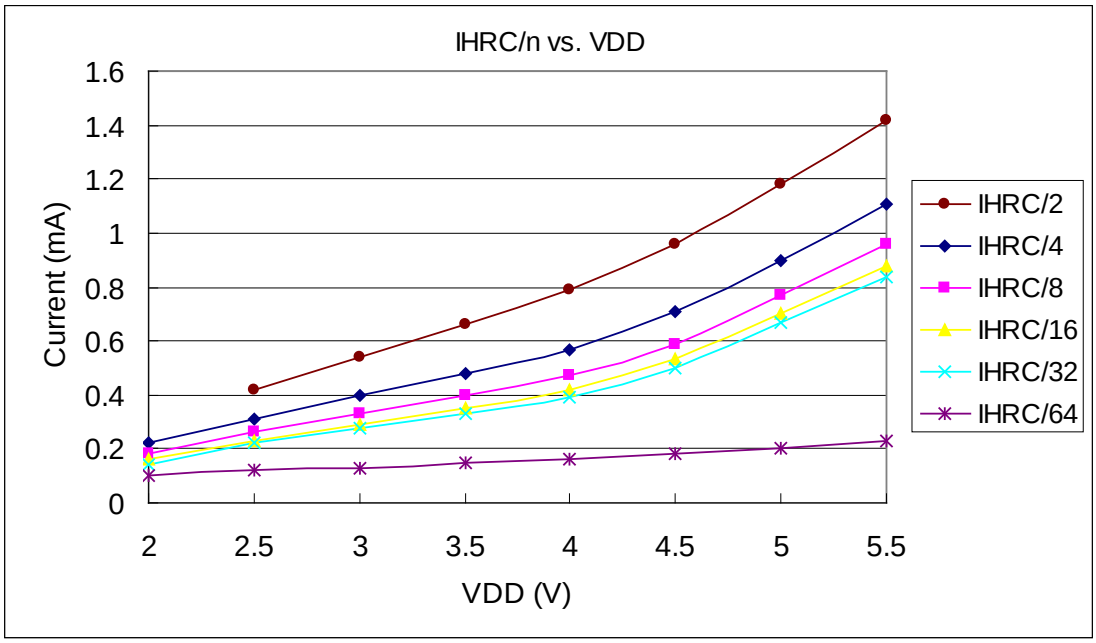


Typical Operating Current vs. VDD and CLK=IHRC/n

- Conditions:

  tog pa0(1s), no t16m, no interrupt, no floating IO pins, disable ILRC, **EOSCR.0=0**

  Touch:Disable

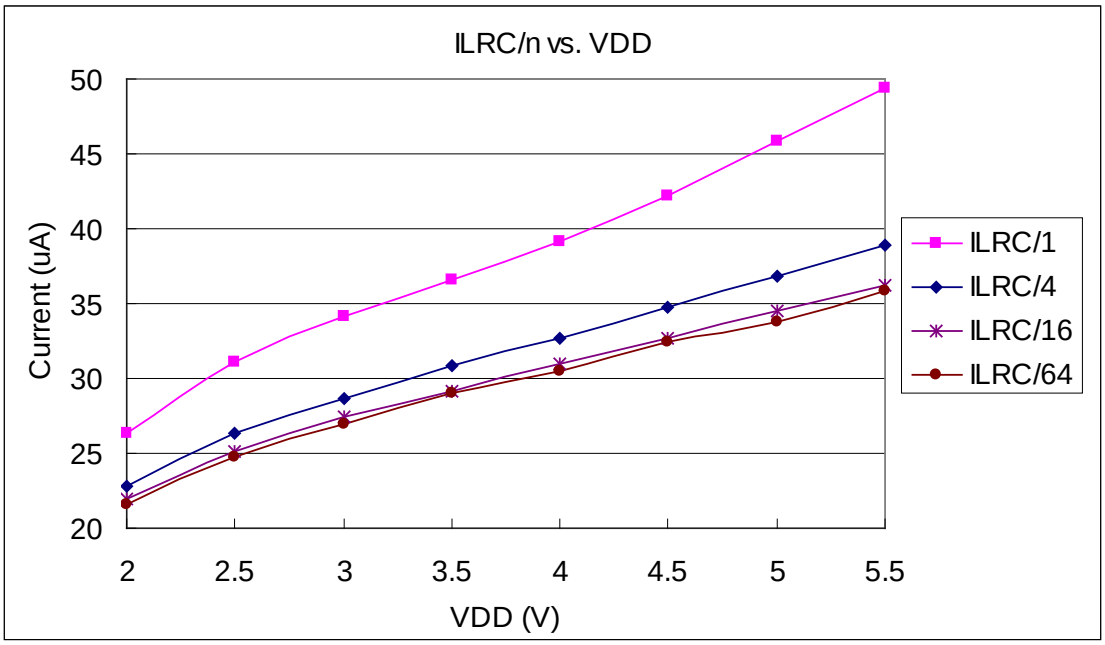


Typical Operating Current vs. VDD and CLK=ILRC/n

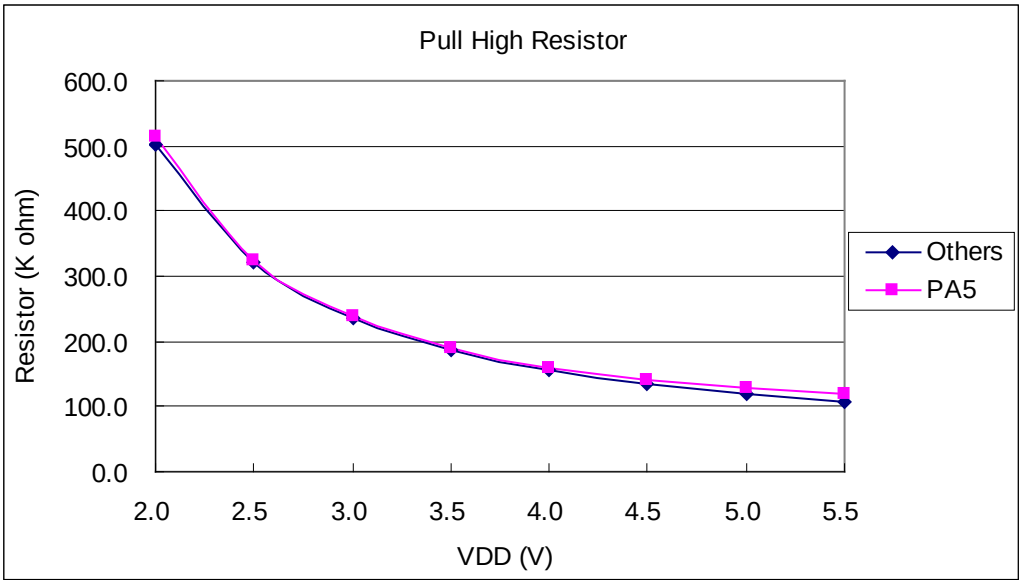
- Conditions:

  tog pa0(1s), no t16m, no interrupt, no floating IO pins, disable ILRC, **EOSCR.0=0**

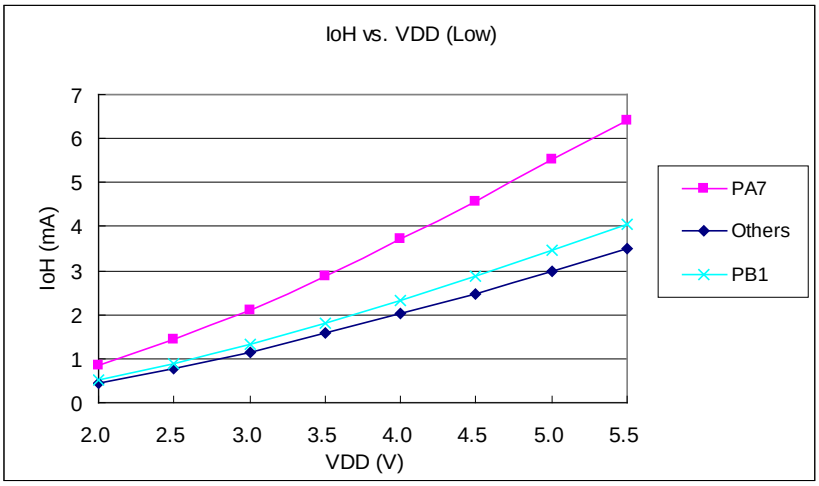
  Touch:Disable

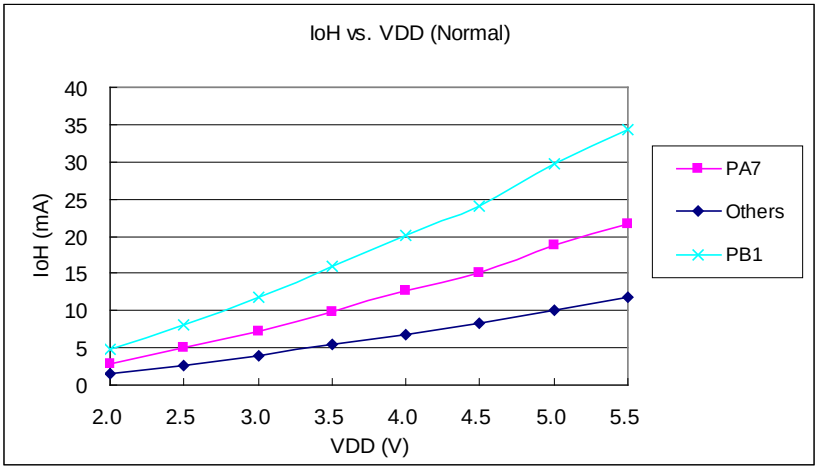


Typical IO pull high resistance

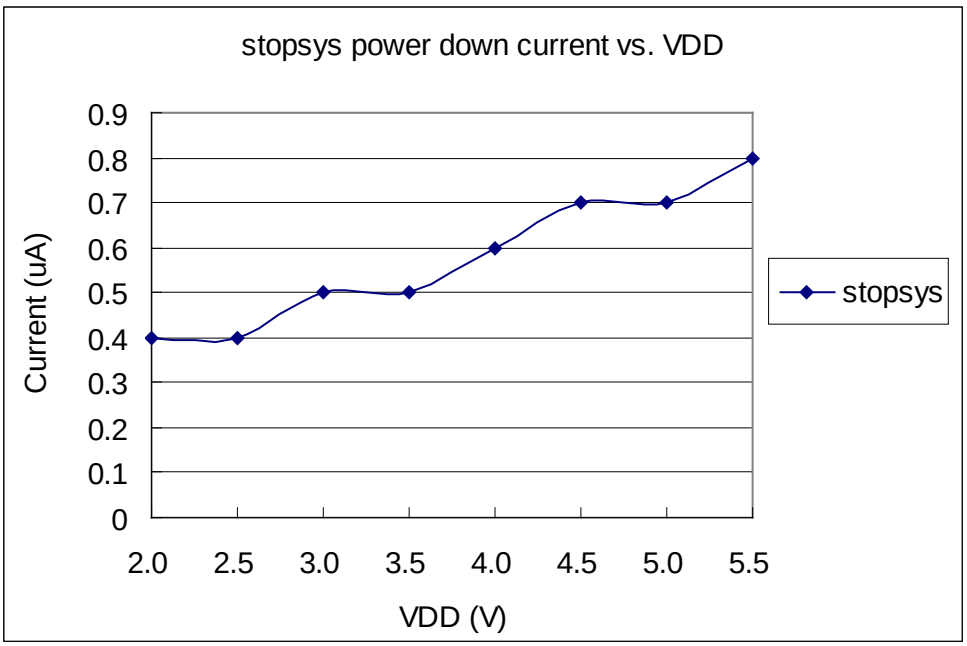


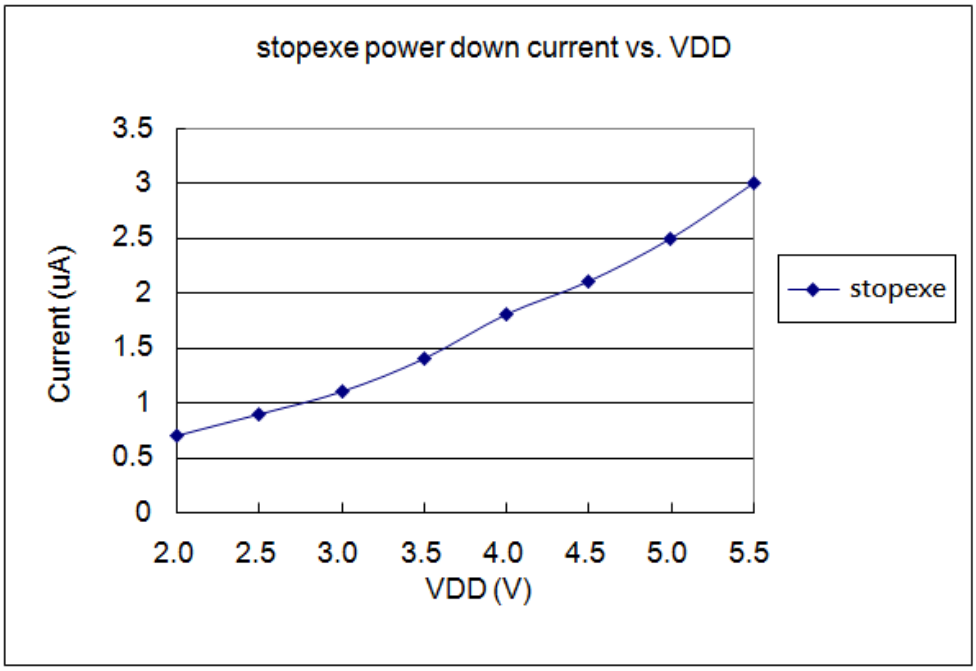
Typical IO driving current (I_{OH}) and sink current (I_{OL})





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





Functional Description

Program Memory – OTP

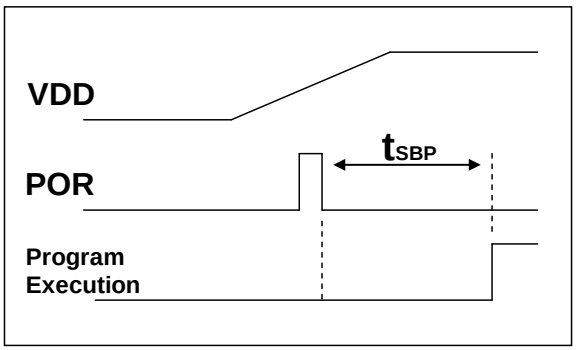
The OTP (One Time Programmable) program memory is used to store the program instructions to be executed. The OTP program memory may contains the data, tables and interrupt entry. After reset, the initial address for FPP0 is 0x000. The interrupt entry is 0x010 if used, the last 16 addresses are reserved for system using, like checksum, serial number, etc. The OTP program memory for MCU401 is a 1.5KW that is partitioned as Table 1. The OTP memory from address 0x5F0 to 0x5FF is for system using, address space from 0x001 to 0x00F and from 0x011 to 0x5EF is user program space.

Address	Function
0x000	FPP0 reset – goto instruction
0x001	User program
•	•
•	•
0x00F	User program
0x010	Interrupt entry address
0x011	User program
•	•
0x5EF	User program
0x5F0	System Using
•	•
0x5FF	System Using

Table 1: Program Memory Organization

Boot Up

POR (Power-On-Reset) is used to reset MCU401 when power up, the boot up time can be optional fast or slow, time for fast boot up is about 32 ILRC clock cycles and 2048 ILRC clock cycles for slow boot up. Customer must ensure the stability of supply voltage after power up no matter which option is chosen, the power up sequence is shown in the Fig. 2 and t_{SBP} is the boot up time.



Boot up from Power-On Reset

Fig. 2: Power Up Sequence

Data Memory – SRAM

The access of data memory can be byte or bit operation. Besides data storage, the SRAM data memory is also served as data pointer of indirect access method and the stack memory.

The stack memory is defined in the data memory. The stack pointer is defined in the stack pointer register; 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 96 bytes data memory of MCU401 can be accessed by indirect access mechanism.

Oscillator and clock

There are two oscillator circuits provided by MCU401: internal high RC oscillator(IHRC) and internal low RC oscillator(ILRC), and these two oscillators are enabled or disabled by registers clkmd.4 and clkmd.2 independently. User can choose one of these two 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*
IHRC	clkmd.4	Enabled
ILRC	clkmd.2	Enabled

Table 2: Oscillator Module

Internal High RC oscillator and Internal Low RC oscillator

After boot-up, the IHRC and ILRC oscillators are enabled. The frequency of IHRC can be calibrated to eliminate process variation by ***ihrcr*** register; normally it is calibrated to 16MHz. The frequency deviation can be within 2% normally after calibration and it still drifts slightly with supply voltage and operating temperature, the total drift rate is about ±5% for VDD=2.3V~5.5V and -20^{o}C~70^{o}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 **58KHz** under 5V, however, its frequency will vary by process, supply voltage and temperature, please refer to DC specification and do not use for accurate timing application.

IHRC calibration

The IHRC frequency may be different chip by chip due to manufacturing variation, MCU401 provide the IHRC frequency calibration to eliminate this variation, and this function can be selected when compiling user's program and the command will be inserted into user's program automatically. The calibration command is shown as below:

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

Where,

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

p2=14 ~ 18; In order to calibrate the chip to different frequency, 16MHz is the usually one.

p3=2.3 ~ 5.5; In order to calibrate the chip under different supply voltage.

IHRC Frequency Calibration and System Clock

During compiling the user program, the options for IHRC calibration and system clock are shown as Table 3:

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 3: 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 MCU401 for different option:

(1) **.**ADJUST_IC SYSCLK=IHRC/2, IHRC=16MHz, VDD=5V

After boot up, CLKMD = 0x34：

- IHRC frequency is calibrated to 16MHz@VDD=5V and IHRC module is enabled
- System CLK = IHRC/2 = 8MHz
- Watchdog timer is disabled, ILRC is enabled, PA5 is in input mode

(2) **.**ADJUST_IC SYSCLK=IHRC/4, IHRC=16MHz, VDD=3.3V

After boot, CLKMD = 0x14：

- IHRC frequency is calibrated to 16MHz@VDD=3.3V and IHRC module is enabled
- System CLK = IHRC/4 = 4MHz
- Watchdog timer is disabled, ILRC is enabled, PA5 is in input mode

(3) **.**ADJUST_IC SYSCLK=IHRC/8, IHRC=16MHz, VDD=2.5V

After boot, CLKMD = 0x3C：

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

(4) **.**ADJUST_IC SYSCLK=IHRC/16, IHRC=16MHz, VDD=2.3V

After boot, CLKMD = 0x1C：

- IHRC frequency is calibrated to 16MHz@VDD=2.3V and IHRC module is enabled
- System CLK = IHRC/16 = 1MHz
- Watchdog timer is disabled, ILRC is enabled, PA5 is in input mode

(5) .ADJUST_IC SYSCLK=IHRC/32, IHRC=16MHz, VDD=5V

After boot, CLKMD = 0x7C：

- IHRC frequency is calibrated to 16MHz@VDD=5V and IHRC module is enabled
- System CLK = IHRC/32 = 500KHz
- Watchdog timer is disabled, ILRC is enabled, PA5 is in input mode

(6) .ADJUST_IC SYSCLK=ILRC, IHRC=16MHz, VDD=5V

After boot, 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

System Clock and LVR levels

The clock source of system clock comes from IHRC or ILRC, the hardware diagram of system clock in the MCU401 is shown as Fig. 3.

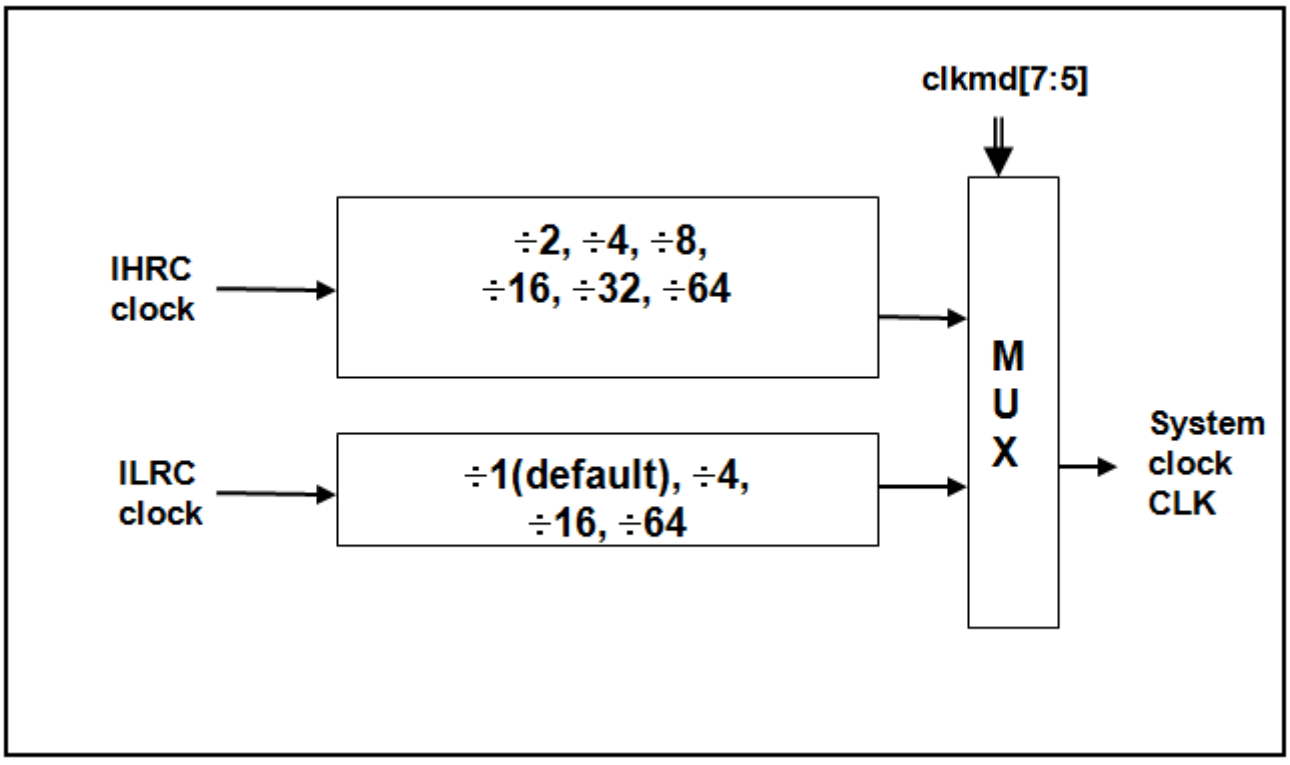


Fig. 3: Options of System Clock

User can choose different operating system clock depends on its requirement; the selected operating system clock should be combined with supply voltage and LVR level to make system stable.

16-bit Timer (Timer16)

MCU401 provide a 16-bit hardware timer (Timer16) and its clock source may come from system clock (CLK), 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. 4.

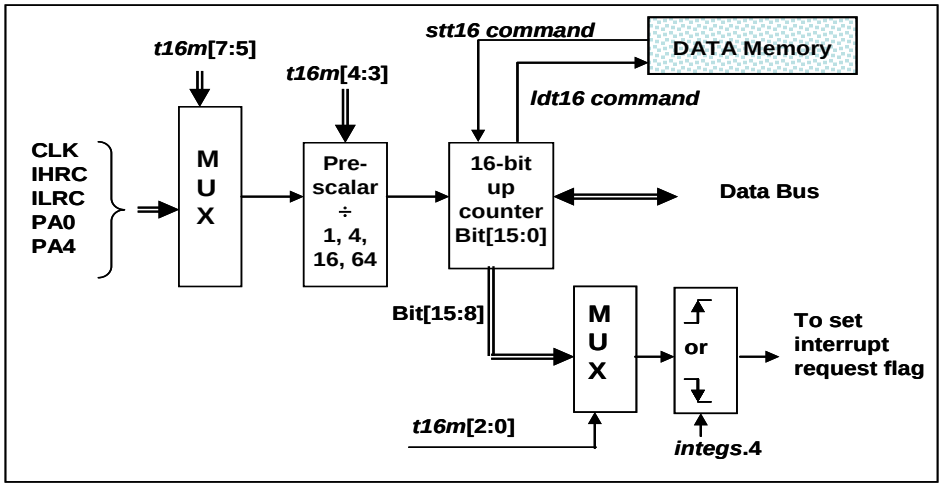


Fig. 4: Hardware diagram of Timer16

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

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

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

```
$     T16M      SYSCLK, /64, BIT15;
      // choose (SYSCLK/64) as clock source, every 2^16 clock to set INTRQ.2=1
      // if system clock SYSCLK = IHRC / 2 = 8 MHz
      // SYSCLK/64 = 8 MHz/64 = 8 uS, about every 524 mS to generate INTRQ.2=1

$     T16M      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
```

Watchdog Timer

The watchdog timer (WDT) is a counter with clock coming from ILRC and its frequency is about **58KHz**@5V. There are four different timeout periods of watchdog timer can be chosen by setting the ***misc*** register, it is:

- 262144 ILRC clock period when misc[1:0]=11
- 65536 ILRC clock period when misc[1:0]=10
- 16384 ILRC clock period when misc[1:0]=01
- 8192 ILRC clock period when misc[1:0]=00 (default)

The frequency of ILRC may drift a lot due to the variation of manufacture, supply voltage and temperature; user should reserve guard band for safe operation. WDT can be cleared by power-on-reset or by command ***wdreset*** at any time. When WDT is timeout, MCU401 will be reset to restart the program execution. The relative timing diagram of watchdog timer is shown as Fig. 5.

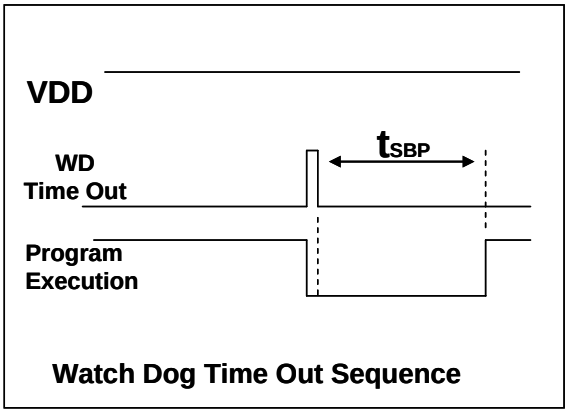


Fig. 5: Sequence of Watch Dog Time Out

Interrupt

There are six interrupt lines for MCU401:

- External interrupt PA0 / PA5
- External interrupt PB0Timer16 interrupt
- Timer2 interrupt
- Two touch key interrupts (TK_OV and TK_END)

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. 6. All the interrupt request flags are set by hardware and cleared by writing ***intrq*** register. When the request flags are set, it can be rising edge, falling edge or both, depending on the setting of register ***integs***. All the interrupt request lines are also controlled by ***engint*** instruction (enable global interrupt) to enable interrupt operation and ***disgint*** instruction (disable global interrupt) to disable it. The stack memory for interrupt is shared with data memory and its address is specified by stack register ***sp.*** Since the program counter is 16 bits width, the bit 0 of stack register ***sp*** should be kept 0. Moreover, user can use ***pushaf*** / ***popaf*** instructions to store or restore the values of ***ACC*** and ***flag*** register ***to*** / ***from*** stack memory.

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

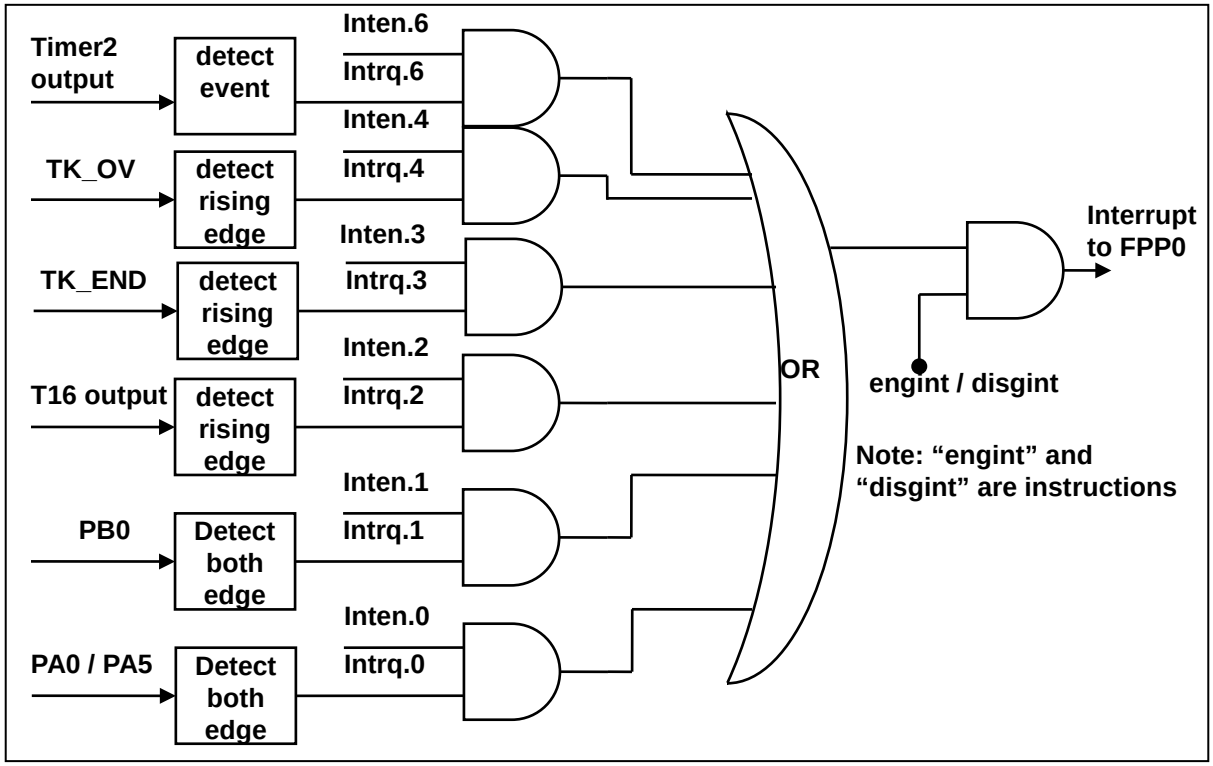


Fig. 6: Hardware diagram of Interrupt controller

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;     // User can not use this instruction
           INTRQ.0 =  0;     // User is recommended to use this instruction
        ...
    }
    ...
    POPAF                    // restore ALU and FLAG register
}
```

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. Table 4 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
STOPSYS	Stop	Stop
STOPEXE	No Change	No Change

Table 4: Differences in oscillator modules between STOPSYS and STOPEXE

Power-Save mode ("stopexe")

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

- IHRC and ILRC 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.

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 following shows the internal status of MCU401 in detail when “***stopsys***” command is issued:

- All the oscillator modules are turned off
- OTP memory is turned off
- The contents of SRAM and registers remain unchanged
- Wake-up sources: ANY IO toggle.
- If PA is input mode and set to analog input by ***padier*** 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, disable watchdog timer
CLKMD.4 = 0;        //  disable IHRC
...
while (1)
{
        STOPSYS;        //  enter power-down
        if (…) break;   //  if wakeup happen and check OK, then return to high speed,
                        //  else stay in power-down mode again.
}
CLKMD  =  0x34;     //  Change clock from ILRC to IHRC/2
```

Wake-up

After entering the Power-Down or Power-Save modes, the MCU401 can be resumed to normal operation by toggling IO pins, Timer16 interrupt is available for Power-Save mode ONLY. Table 5 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

Table 5: Differences in wake-up sources between Power-Save mode and Power-Down mode

When using the IO pins to wake-up the MCU401, registers ***padier*** should be properly set to enable the wake-up function for every corresponding pin. The time for normal wake-up is about 2048 ILRC clocks counting from wake-up event; fast wake-up can be selected to reduce the wake-up time by ***misc*** register, and the time for fast wake-up is 32 ILRC clocks from IO toggling.

Suspend mode	*Wake-up mode*	*Wake-up time (t_{WUP}) from IO toggle*
STOPEXE suspend	fast wake-up	32 * $T_{ILRC,}$ Where T_{ILRC} is the time period of ILRC
STOPSYS suspend	fast wake-up	32 * $T_{ILRC,}$ Where T_{ILRC} is the time period of ILRC
STOPEXE suspend	normal wake-up	2048 * T_{ILRC}, Where T_{ILRC} is the clock period of ILRC
STOPSYS suspend	normal wake-up	2048 * T_{ILRC}, Where T_{ILRC} is the clock period of ILRC

Table 6: Suspend mode / Wake-up mode / Wake-up time from IO toggle

IO Pins

Other than PA5, all the pins can be independently set into two states output or input by configuring the data registers (***pa***), control registers (***pac***) and pull-high registers (***paph)***. 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 7 shows the configuration table of bit 0 of port A. The hardware diagram of IO buffer is also shown as Fig. 7.

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 7: PA0 Configuration Table

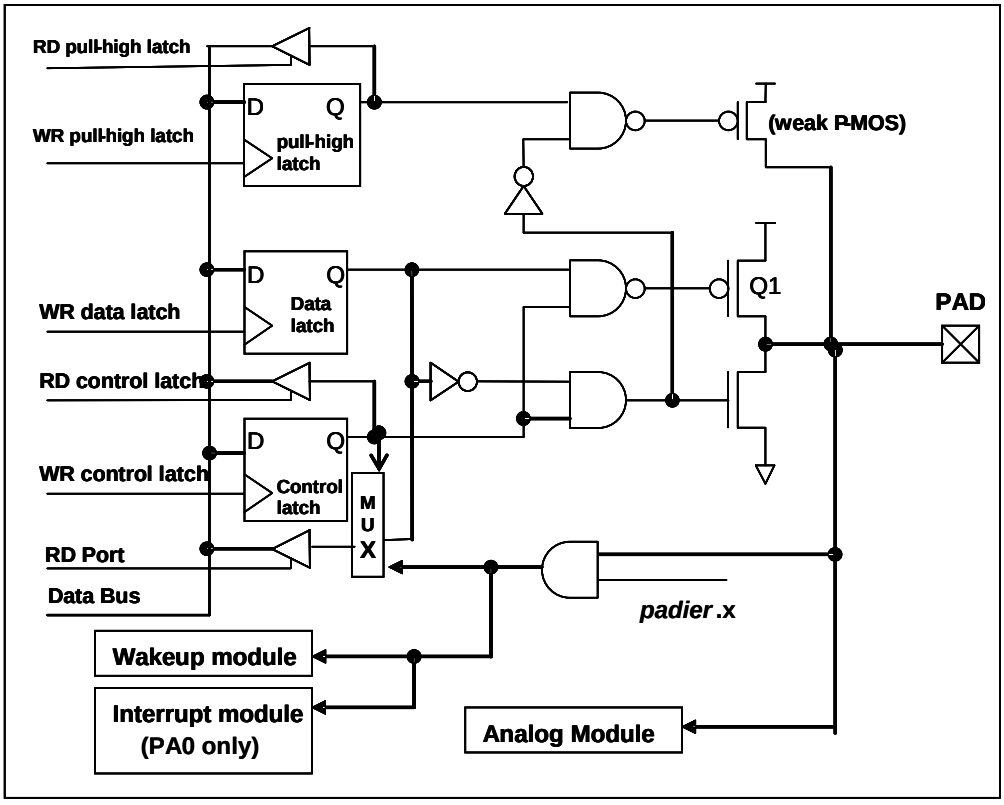


Fig. 7: Hardware diagram of IO buffer

Other than PA5, all the IO pins have the same structure; PA5 can be open-drain ONLY when setting to output mode (without Q1). When MCU401 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*** to high***.*** The same reason, ***padier***.0 should be set to high when PA0 is used as external interrupt pin.

Reset

There are many causes to reset the MCU401, once reset is asserted, all the registers in MCU401 will be set to default values, 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 LVR; however, the content will be kept when reset comes from PRST# pin or WDT timeout.

Timer (Timer2) with PWM generation

An 8-bit hardware timer (Timer2) with PWM generation is implemented in the MCU401, please refer to Fig. 8 shown its hardware diagram, the clock sources of Timer2 may come from system clock, internal high RC oscillator (IHRC), internal low RC oscillator (ILRC), PB0, PA0 and PA4, bit [7:4] of register tm2c are used to select the clock of Timer2. Please notice that if IHRC is selected for Timer2 clock source, the clock sent to Timer2 will keep running when using ICE in halt state. The output of Timer2 can be sent to pin PA3, PB1 or PB4, depending on bit [3-2] of tm2c register. A clock pre-scaling module is provided with divided-by-1, 4, 16, and 64 options, controlled by bit [6:5] of tm2s register; one scaling module with divided-by-1~31 is also provided and controlled by bit [4:0] of tm2s register. In conjunction of pre-scaling function and scaling function, the frequency of Timer2 clock (TM2_CLK) can be wide range and flexible. TM2_CLK can be selected as system clock in order to provide special frequency of system clock, please refer to ***clkmd*** register.

The Timer2 counter performs 8-bit up-counting operation only; the counter values can be set or read back by tm2ct register. The 8-bit counter will be clear to zero automatically when its values reach for upper bound register, the upper bound register is used to define the period of timer or duty of PWM. There are two operating modes for Timer2: period mode and PWM mode; period mode is used to generate periodical output waveform or interrupt event; PWM mode is used to generate PWM output waveform with optional 6-bit or 8-bit PWM resolution, Fig. 9 shows the timing diagram of Timer2 for both period mode and PWM mode.

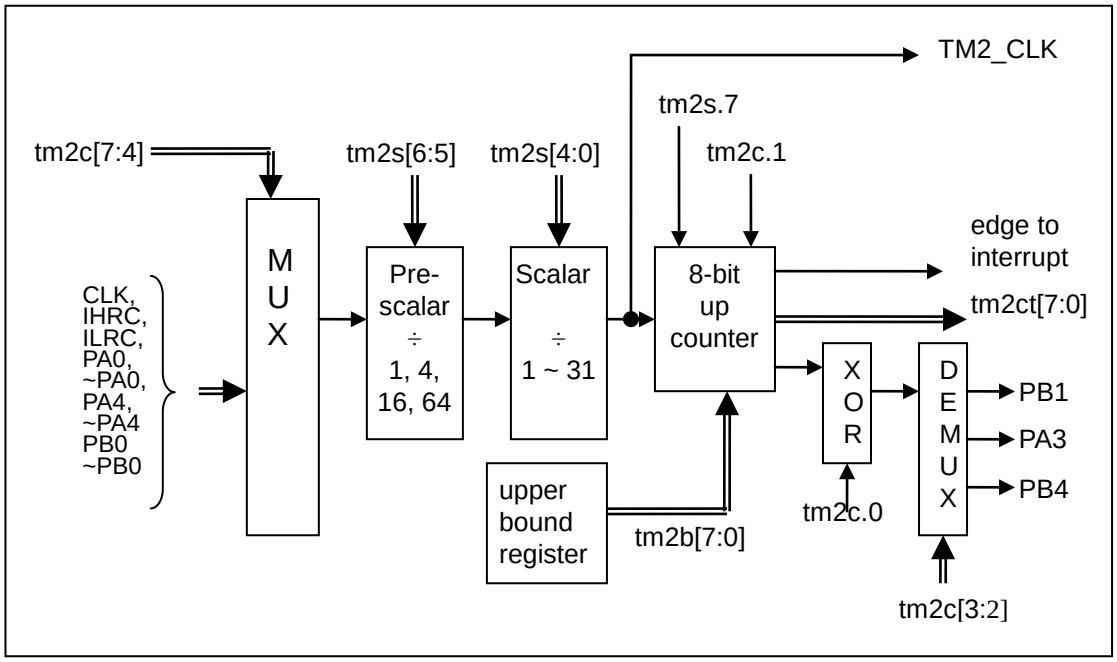
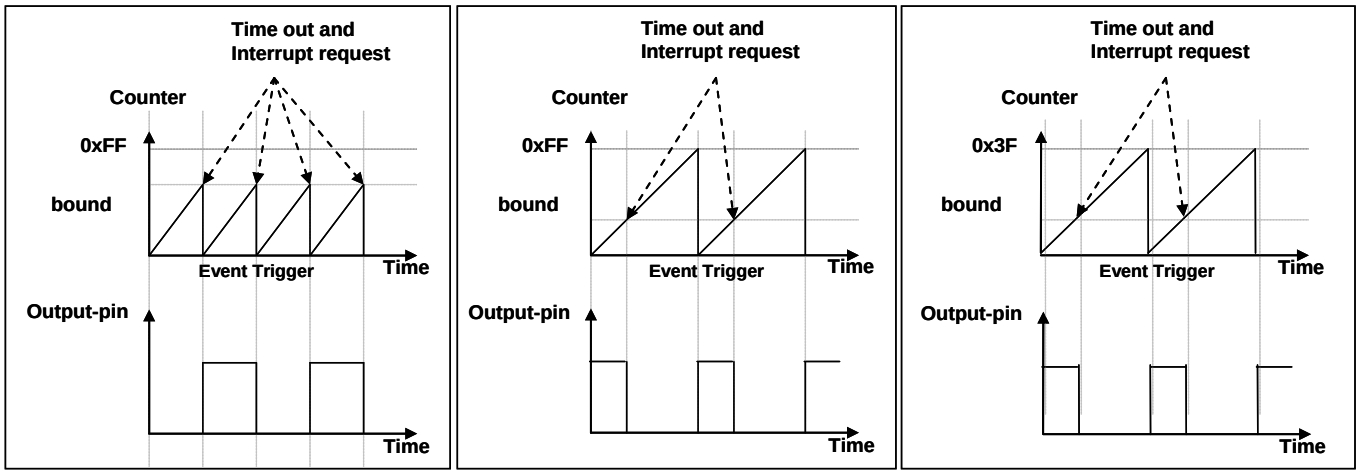


Fig. 8: Timer2 hardware diagram



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

Fig. 9: Timing diagram of Timer2 in period mode and PWM mode (tm2c.1=1)

Using the Timer2 to generate periodical waveform

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

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

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

Example 1:

tm2c = 0b0001_1000, Y=8MHz
tm2b = 0b0111_1111, K=127
tm2s = 0b0_00_00000, S1=1, S2=0
➔ frequency of output = 8MHz ÷ [2 × (127＋1) × 1 × (0＋1)] = 31.25kHz

Example 2:

tm2c = 0b0001_1000, Y=8MHz
tm2b = 0b0111_1111, K=127
tm2s[7:0] = 0b0_11_11111, S1=64 , S2 = 31
➔ frequency = 8MHz ÷ (2 × (127＋1) × 64 × (31＋1)) =15.25Hz

Example 3:

tm2c = 0b0001_1000, Y=8MHz
tm2b = 0b0000_1111, K=15
tm2s = 0b0_00_00000, S1=1, S2=0
➔ frequency = 8MHz ÷ (2 × (15＋1) × 1 × (0＋1)) = 250kHz

Example 4:

tm2c = 0b0001_1000, Y=8MHz
tm2b = 0b0000_0001, K=1
tm2s = 0b0_00_00000, S1=1, S2=0
➔ frequency = 8MHz ÷ (2 × (1＋1) × 1 × (0＋1)) =2MHz

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

```
void  FPPA0 (void)
{
    . ADJUST_IC  SYSCLK=IHRC/2, IHRC=16MHz, VDD=5V
    …
    tm2ct = 0x0;
    tm2b = 0x7f;
    tm2s = 0b0_00_00001;          //   8-bit PWM, pre-scalar = 1, scalar = 2
    tm2c = 0b0001_10_0_0;         //   system clock, output=PA3, period mode
    while(1)
    {
     nop;
    }
}
```

Using the Timer2 to generate 8-bit PWM waveform

If 8-bit PWM mode is selected, it should set ***tm2c***[1]=1 and ***tm2s***[7]=0, the frequency and duty cycle of output waveform can be summarized as below:

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

Duty of Output = (K＋1) ÷ 256

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

Example 1:

tm2c = 0b0001_1010, Y=8MHz
tm2b = 0b0111_1111, K=127
tm2s = 0b0_00_00000, S1=1, S2=0
➔ frequency of output = 8MHz ÷ (256 × 1 × (0+1)) = 31.25kHz
➔ duty of output = [(127+1) ÷ 256] × 100% = 50%

Example 2:

tm2c = 0b0001_1010, Y=8MHz
tm2b = 0b0111_1111, K=127
tm2s = 0b0_11_11111, S1=64, S2=31
➔ frequency of output = 8MHz ÷ (256 × 64 × (31+1)) = 15.25Hz
➔ duty of output = [(127+1) ÷ 256] × 100% = 50%

Example 3:

tm2c = 0b0001_1010, Y=8MHz
tm2b = 0b1111_1111, K=255
tm2s = 0b0_00_00000, S1=1, S2=0
➔ frequency of output = 8MHz ÷ (256 × 1 × (0+1)) = 31.25kHz
➔ duty of output = [(255+1) ÷ 256] × 100% = 100%

Example 4:

tm2c = 0b0001_1010, Y=8MHz
tm2b = 0b0000_1001, K = 9
tm2s = 0b0_00_00000, S1=1, S2=0
➔ frequency of output = 8MHz ÷ (256 × 1 × (0+1)) = 31.25kHz
➔ duty of output = [(9+1) ÷ 256] × 100% = 3.9%

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

```
void   FPPA0 (void)
{
   .ADJUST_IC  SYSCLK=IHRC/2, IHRC=16MHz, VDD=5V
   wdreset;
   tm2ct = 0x0;
   tm2b = 0x7f;
   tm2s = 0b0_00_00001;      //     8-bit PWM, pre-scalar = 1, scalar = 2
   tm2c = 0b0001_10_1_0;     //     system clock, output=PA3, PWM mode
   while(1)
   {
       nop;
   }
}
```

Using the Timer2 to generate 6-bit PWM waveform

If 6-bit PWM mode is selected, it should set ***tm2c***[1]=1 and ***tm2s***[7]=1, the frequency and duty cycle of output waveform can be summarized as below:

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

Duty of Output = [(K＋1) ÷ 64] × 100%

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

Example 1:

tm2c = 0b0001_1010, Y=8MHz

tm2b = 0b0001_1111, K=31

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

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

➔ duty = [(31+1) ÷ 64] × 100% = 50%

Example 2:

tm2c = 0b0001_1010, Y=8MHz

tm2b = 0b0001_1111, K=31

tm2s = 0b1_11_11111, S1=64, S2=31

➔ frequency of output = 8MHz ÷ (64 × 64 × (31+1)) = 61.03 Hz

➔ duty of output = [(31+1) ÷ 64] × 100% = 50%

Example 3:

tm2c = 0b0001_1010, Y=8MHz

tm2b = 0b0011_1111, K=63

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

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

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

Example 4:

tm2c = 0b0001_1010, Y=8MHz

tm2b = 0b0000_0000, K=0

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

➔ frequency = 8MHz ÷ (64 × 1 × (0+1)) = 125kHz

➔ duty = [(0+1) ÷ 64] × 100% =1.5%

Touch Function

Block Diagram

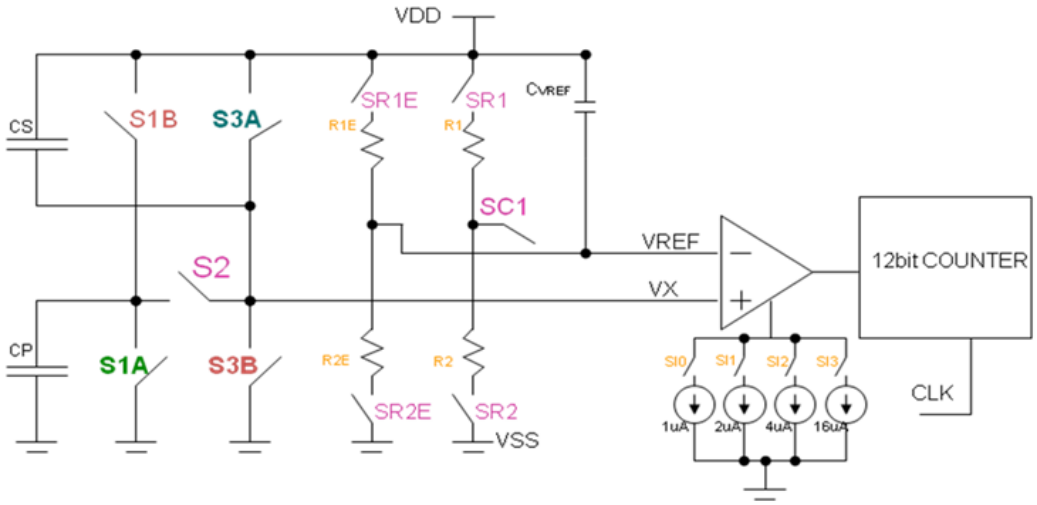


Fig. 10: Touch hardware diagram

Functional description of Touch pad

Touch pads operate in one of several ways, including capacitive sensing and resistive touchscreen. The circuit blocks shown above are the method of capacitive sensing, detecting the capacitive virtual ground effect of a finger, or the capacitance between sensors.

A serial of resisters R1, R2, R1E and R2E are used to generate a reference voltage VREF that can be 0.8VDD ~ 0.5VDD designated by register settings for comparator input.

An accurate、external capacitor CS is fully discharged by switch S3 to make node VX potential equal to VDD (note 1) before each detection process, then by switching S2/S1 periodically, VX value will go down (note 2) clock by clock due to charge sharing of capacitors and CP discharge .

Once VX reach reference value, comparator output will change it's state. The periods it needs to trigger the comparator is related to the ratio of CS and CP, while CP represents the total capacitance that is the combination of PCB, wire and touch pad whose capacitance can be varied by human finger's touch. Once the CP value is altered, the periods required to change state shorten. By counting the discrepancy of clock periods, the circuitry can decide if the touch pad is enabled.

The current consumption of comparator can be adjusted accordingly by registers in order to save power when operating at lower clock speed.

Note 1 : It's for type A . VX will be discharged to VSS for type B.

Note 2 : It's for type A also. VX will rise up for type B.

IO Registers

ACC Status Flag Register (*flag*), IO address = 0'h00

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

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

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

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 01x: reserved 100: reserved 101: reserved 110: ILRC/4 111: ILRC (default)	000: IHRC/16 001: IHRC/8 010: ILRC/16 011: IHRC/32 100: IHRC/64 110: ILRC/64 1xx: reserved.
4	1	R/W	IHRC oscillator Enable. 0 / 1: disable / enable	
3	0	RW	Clock Type Select. This bit is used to select the clock type in bit [7:5]. 0 / 1: Type 0 / Type 1	
2	1	R/W	ILRC Enable. 0 / 1: disable / enable If ILRC is disabled, watchdog timer is also disabled.	
1	1	R/W	Watch Dog Enable. 0 / 1: disable / enable	
0	0	R/W	Pin PA5/PRST# function. 0 / 1: PA5 / PRST#.	

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

Bit	Reset	R/W	Description
7	-	-	Reserved.
6	-	R/W	Enable interrupt from Timer2. 0 / 1: disable / enable.
5	-	-	Reserved.
4	-	R/W	Enable interrupt from Touch Key TK_OV. 0 / 1: disable / enable.
3	-	R/W	Enable interrupt from Touch Key TK_END. 0 / 1: disable / enable.
2	-	R/W	Enable interrupt from Timer16 overflow. 0 / 1: disable / enable.
1	-	R/W	Enable interrupt from PB0. 0 / 1: disable / enable.
0	-	R/W	Enable interrupt from PA0 / PA5. 0 / 1: disable / enable.

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	-	-	Reserved.
4	-	R/W	Interrupt Request from Touch Key TK_OV, this bit is set by hardware and cleared by software. 0 / 1: No request / Request
3	-	R/W	Interrupt Request from Touch Key TK_END, this bit is set by hardware and cleared by software. 0 / 1: No request / Request
2	-	R/W	Interrupt Request from Timer16, this bit is set by hardware and cleared by software. 0 / 1: No request / Request
1	-	R/W	Interrupt Request from pin PB0, this bit is set by hardware and cleared by software. 0 / 1: No request / Request
0	-	R/W	Interrupt Request from pin PA0 / PA5, this bit is set by hardware and cleared by software. 0 / 1: No request / Request

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

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

MISC Register (*misc*), IO address = 0x08

Bit	Reset	R/W	Description
7 - 6	-	-	Reserved
5	0	WO	Enable fast Wake up. Fast wake-up is NOT supported when EOSC is enabled. 0: Normal wake up. The wake-up time is 2048 ILRC clocks (Not for fast boot-up) 1: Fast wake up. The wake-up time is 32 ILRC clocks.
4 - 3	00	-	Reserved
2	0	WO	Disable LVR function. 0 / 1 : Enable / Disable
1 – 0	00	WO	Watchdog time out period 00: 8192 ILRC clock period 01: 16384 ILRC clock period 10: 65536 ILRC clock period 11: 262144 ILRC clock period

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

Bit	Reset	R/W	Description
7 – 1	-	-	Reserved. Please keep 0.
0	0	WO	Power-down the LVR hardware modules. 0 / 1: normal / power-down.

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

Bit	Reset	R/W	Description
7 – 5	-	-	Reserved. Please keep 0.
4	0	WO	Timer16 edge selection. 0 : rising edge to trigger interrupt 1 : falling edge to trigger interrupt
3 – 2	00	WO	PB0 edge selection. 00 : both rising edge and falling edge to trigger interrupt 01 : rising edge to trigger interrupt 10 : falling edge to trigger interrupt 11 : reserved.
1 – 0	00	WO	PA0 / PA5 edge selection. 00 : both rising edge and falling edge to trigger interrupt 01 : rising edge to trigger interrupt 10 : falling edge to trigger interrupt 11 reserved.

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

Bit	Reset	R/W	Description
7 – 3	11111	WO	Enable PA7~PA3 wake up event. 1 / 0 : enable / disable. These bits can be set to low to disable wake up from PA7~PA3 toggling. Note: For ICE emulation, the function is disabled when this bit is “1” and “0” is enabled.
2	-	-	Reserved.
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, the function 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.

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 can be set to low to disable wake-up from PB7 toggling. Note: For ICE emulation, the function is disabled when this bit is “1” and “0” is enabled.
6	1	WO	Enable PB6 wake-up event. 1 / 0 : enable / disable. This bit can be set to low to disable wake-up from PB6 toggling. Note: For ICE emulation, wakeup is disabled when this bit is “1” and “0” is enabled.
5	1	WO	Enable PB5 wake-up event. 1 / 0 : enable / disable. This bit can be set to low to disable wake-up from PB5 toggling. Note: For ICE emulation, wakeup is disabled when this bit is “1” and “0” is enabled.
4	1	WO	Enable PB4 wake-up event. 1 / 0 : enable / disable. This bit can be set to low to disable wake-up from PB4 toggling. Note: For ICE emulation, wakeup 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 can be set to low to disable wake-up from PB3 toggling.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 can be set to low to disable wake-up from PB2 toggling. 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 can be set to low to disable wake-up from PB1 toggling. Note: For ICE emulation, the function is disabled when this bit is “1” and “0” is enabled.
0	1	WO	Enable PB0 digital input, wake-up event and interrupt request. 1 / 0 : enable / disable. This bit can be set to low to disable wake up from PB0 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:

“$ PBDIER 0xhh” ;

For example:

$ PBDIER 0x F0;

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

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

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

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

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

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

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

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

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

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

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

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

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

Timer2 Control Register (tm2c), IO address = 0x1c

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 – 2	00	R/W	Timer2 output selection. 00 : disable 01 : PB1 10 : PA3 11 : PB4
1	0	R/W	Timer2 mode selection. 0 / 1 : period mode / PWM mode
0	0	R/W	Enable to inverse the polarity of Timer2 output. 0 / 1: disable / enable

Timer2 Counter Register (tm2ct), IO address = 0x1d

Bit	Reset	R/W	Description
7 – 0	0'h00	R/O	Bit [7:0] of Timer2 counter register.

Note: Timer2 is designed for PWM mode and Period mode, so do not read tm2ct register.

Timer2 Scalar Register (tm2s), IO address = 0x17

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

Timer2 Bound Register (tm2b), IO address = 0x09

Bit	Reset	R/W	Description
7 – 0	0'h00	WO	Timer2 bound register.

MISC Register 3 (misc3), IO address = 0x1E

Bit	Reset	R/W	Description
7 – 2	-	-	Reserved.
1	0	WO	Select Pin CS / PA7 function. 0 / 1: CS / PA7.
0	0	WO	Select interrupt PA0 / PA5 from pin PA5 / PA0. 0 / 1: PA5 / PA0.

Touch Selection Register (ts), IO address = 0x20

Bit	Reset	R/W	Description
7 – 4	-	R/W	Touch Pad clock selection (TK_CLK) 0000: reserved 0001: reserved 0010: IHRC/4 0011: IHRC/8 0100: IHRC/16 0101: IHRC/32 0110: IHRC/64 0111: IHRC/128 1111: ILRC Others: reserved
3 – 2	-	R/W	Touch Pad VREF selection 00: 0.5 * VCC 01: 0.6 * VCC 10: 0.7 * VCC 11: 0.8 * VCC
1 – 0	-	R/W	Select the discharge time before starting the touch function (TK_DISCHG) 00: 0 * CLK 01: 32 * CLK 10: 64 * CLK 11: 128 * CLK

Touch Charge Control Register (tcc), IO address = 0x21

Bit	Reset	R/W	Description		
7	-	-	Reserved		
6 - 4	-	R/W	Touch control and status		
			Data	Command (W)	Status (R)
			000	TK_STOP (Touch module power down)	Ready / End
			001	TK_RUN	Running
			011	Discharge (Discharge CS capacitor)	Discharging
			Others	Reserved	Reserved
3 – 0	-	-	reserved		

Touch Key Enable 2 Register (tke2), IO address = 0x22

Bit	Reset	R/W	Description
7 – 4	-	-	Reserved
3	0	R/W	Enable TK11. 0/1: disable/enable
2	0	R/W	Enable TK10. 0/1: disable/enable
1	0	R/W	Enable TK9. 0/1: disable/enable
0	0	R/W	Enable TK8. 0/1: disable/enable

Touch Key Enable 1 Register (tke1), IO address = 0x24

Bit	Reset	R/W	Description
7	0	R/W	Enable TK7. 0/1: disable/enable
6	0	R/W	Enable TK6. 0/1: disable/enable
5	0	R/W	Enable TK5. 0/1: disable/enable
4	0	R/W	Enable TK4. 0/1: disable/enable
3	0	R/W	Enable TK3. 0/1: disable/enable
2	0	R/W	Enable TK2. 0/1: disable/enable
1	0	R/W	Enable TK1. 0/1: disable/enable
0	-	-	reserved

Touch Parameter Setting Register (tps), IO address = 0x26

Bit	Reset	R/W	Description
7 - 6	00	R/W	Debouncing period selection 00: 2 clocks 01: 4 clocks 10: 8 clocks 11: 16 clocks
5 - 4	00	R/W	Resistor change 00: X1 01: X2 10: X3 11: X4
3 - 2	00	R/W	Period of the frequency switching 00: 8us 10: 16us x1: Random
1 – 0	00	R/W	Select the frequency switching range 00: Disable 01: -3a, -2a, -a, 0, a, 2a, 3a 10: -2a, -a, 0, a, 2a 11: -a, 0, a Where a is the minimum resolution of IHRC

IO Special Function Register (iosf), IO address = 0x27

Bit	Reset	R/W	Description
7	0	R/W	Enable IO special function. 0 / 1 : disable / enable When this bit is enabled, PA5\PB0\PB1 is defined as special function. PA5 is forced to be input mode, and PB0, PB1 is forced to be output mode and the output data come from ODATA[3:0]. PA5 (input) / PB0 (output) / PB1 (output) 1 / ODATA bit0 / ODATA bit 1 0 / ODATA bit2 / ODATA bit 3
6 – 4	-	-	Reserved
3	-	R/W	ODATA[3]
2	-	R/W	ODATA[2]
1	-	R/W	ODATA[1]
0	-	R/W	ODATA[0]

Touch Parameter Setting Register 2 (tps2), IO address = 0x28

Bit	Reset	R/W	Description
7 – 6	00	R/W	Touch module type 00: Type A 01: Type B 1X: Reserved
5 - 3	000	R/W	Touch current source 000: 1uA 001: 0.5uA 010: 0.25uA 011: 0.125uA 100: Common with ILRC module Others: Reserved
2	0	R/W	Resistor selection for type A 0 / 1: 16M / 8M Ohm
1 – 0	00	R/W	Vref non-floating cycles selection 00: Vref always floating after discharge 01: Vref always on after discharge 10: First 64 cycles after discharge 11: First 128 cycles after discharge

Touch Key Charge Counter High Register (tkch), IO address = 0x2B

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

Touch Key Charge Counter Low Register (tkcl), IO address = 0x2C

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

Instructions

Symbol	Description
ACC	Accumulator (Abbreviation of accumulator)
a	Accumulator (Symbol of accumulator in program)
sp	Stack pointer
flag	ACC status flag register
I	Immediate data
&	Logical AND
|	Logical OR
←	Movement
^	Exclusive logic OR
+	Add
–	Subtraction
~	NOT (logical complement, 1's complement)
∓	NEG (2's complement)
OV	Overflow (The operational result is out of range in signed 2's complement number system)
Z	Zero (If the result of ALU operation is zero, this bit is set to 1)
C	Carry (The operational result is to have carry out for addition or to borrow carry for subtraction in unsigned number system)
AC	Auxiliary Carry (If there is a carry out from low nibble after the result of ALU operation, this bit is set to 1)
pc0	Program counter for FPP0
M.n,	Only addressed in 0~0x3F (0~63) is allowed
IO.n	Only addressed in 0~0x3F (0~63) is allowed

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* pa, a; Result: pa ← a Affected flags: 『N』Z 『N』C 『N』AC 『N』OV
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, 0'h00 ; // assign 0'h 00 to an address (MSB), should be 0 mov hb@RAMIndex, a ; // save pointer to RAM (MSB) … idxm a, RAMIndex ; // move 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, 0'h00 ; // assign 0'h 00 to an address (MSB), should be 0 *mov* hb@RAMIndex, a ; // save pointer to RAM (MSB) … *mov* a, 0xA5 ; *idxm* RAMIndex, a ; // move 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

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 *comp* lement 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

Shift Operation Instructions

sr a	Shift right of ACC, shift 0 to bit 7 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 bit 7 to flag 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, shift 0 to bit 7 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 bit 7 to flag 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 shift 0 to bit 0 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 bit 0 to flag 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, shift 0 to bit 0 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 bit 0 to flag 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

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 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, 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

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* pa.5 ; Result: set bit 5 of port A to high Affected flags: 『N』Z 『N』C 『N』AC 『N』OV
set0 M.n	Set bit n of memory to low Example: *set0* MEM.5 ; Result: set bit 5 of MEM to low Affected flags: 『N』Z 『N』C 『N』AC 『N』OV
set1 M.n	Set bit n of memory to high Example: *set1* MEM.5 ; Result: set bit 5 of MEM to high Affected flags: 『N』Z 『N』C 『N』AC 『N』OV

swapc IO.n

Swap the nth bit of IO port with carry bit

Example: swapc IO.0;

Result: C ← IO.0 , IO.0 ← C

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

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

Affected flags: 『N』Z 『Y』C 『N』AC 『N』OV

Application Example1 (serial output) :

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

Application Example2 (serial input) :

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

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
cneqsn a, M	Compare ACC with memory and skip next instruction if both are not equal. Flag will be changed like as (a ← a - M) Example: *cneqsn* a, MEM; Result: If a≠MEM, skip next instruction Affected flags: 『Y』Z 『Y』C 『Y』AC 『Y』OV
cneqsn a, I	Compare ACC with immediate data and skip next instruction if both are no equal. Flag will be changed like as (a ← a - I) Example: *cneqsn* a,0x55 ; *inc* MEM ; *goto* error ; Result: If a≠0x55, then “goto error”; Otherwise, “inc MEM”. Affected flags: 『Y』Z 『Y』C 『Y』AC 『Y』OV
t0sn IO.n	Check IO bit and skip next instruction if it’s low Example: *t0sn* pa.5; Result: If bit 5 of port A is low, skip next instruction Affected flags: 『N』Z 『N』C 『N』AC 『N』OV
t1sn IO.n	Check IO bit and skip next instruction if it’s high Example: *t1sn* pa.5 ; Result: If bit 5 of port A is high, skip next instruction Affected flags: 『N』Z 『N』C 『N』AC 『N』OV
t0sn M.n	Check memory bit and skip next instruction if it’s low Example: t0sn MEM.5 ; Result: If bit 5 of MEM is low, then skip next instruction Affected flags: 『N』Z 『N』C 『N』AC 『N』OV
t1sn M.n	Check memory bit and skip next instruction if it’s high EX: t1sn MEM.5 ; Result: If bit 5 of MEM is high, then skip next instruction Affected flags: 『N』Z 『N』C 『N』AC 『N』OV
izsn a	Increment ACC and skip next instruction if ACC is zero Example: izsn a; Result: a ← a + 1,skip next instruction if a = 0 Affected flags: 『Y』Z 『Y』C 『Y』AC 『Y』OV

dzsn a	Decrement ACC and skip next instruction if ACC is zero Example: *dzsn* a; Result: A ← A - 1,skip next instruction if a = 0 Affected flags: 『Y』Z 『Y』C 『Y』AC 『Y』OV
izsn M	Increment memory and skip next instruction if memory is zero Example: *izsn* MEM; Result: MEM ← MEM + 1, skip next instruction if MEM= 0 Affected flags: 『Y』Z 『Y』C 『Y』AC 『Y』OV
dzsn M	Decrement memory and skip next instruction if memory is zero Example: *dzsn* MEM; Result: MEM ← MEM - 1, skip next instruction if MEM = 0 Affected flags: 『Y』Z 『Y』C 『Y』AC 『Y』OV

System control Instructions

call label	Function call, address can be full range address space Example: *call* function1; Result: [sp] ← pc + 1 pc ← function1 sp ← sp + 2 Affected flags: 『N』Z 『N』C 『N』AC 『N』OV
goto label	Go to specific address which can be full range address space Example: *goto* error; Result: Go to error and execute program. Affected flags: 『N』Z 『N』C 『N』AC 『N』OV
ret I	Place immediate data to ACC, then return Example: *ret* 0x55; Result: A← 55h ret ; Affected flags: 『N』Z 『N』C 『N』AC 『N』OV
ret	Return to program which had function call Example: ret; Result: sp ← sp - 2 pc ← [sp] Affected flags: 『N』Z 『N』C 『N』AC 『N』OV
reti	Return to program that is interrupt service routine. After this command is executed, global interrupt is enabled automatically. Example: reti; Affected flags: 『N』Z 『N』C 『N』AC 『N』OV
nop	No operation Example: nop; Result: nothing changed Affected flags: 『N』Z 『N』C 『N』AC 『N』OV

pcadd a	Next program counter is current program counter plus ACC. Example: pcadd a; Result: pc ← pc + a Affected flags: 『N』Z 『N』C 『N』AC 『N』OV Application Example: --- … mov a, 0x02 ; pcadd a ; // PC <- PC+2 goto err1 ; goto correct ; // jump here goto err2 ; goto err3 ; … correct: // jump here … ---
engint	Enable global interrupt enable Example: *engint*; Result: Interrupt request can be sent to FPP0 Affected flags: 『N』Z 『N』C 『N』AC 『N』OV
disgint	Disable global interrupt enable Example: *disgint* ; Result: Interrupt request is blocked from FPP0 Affected flags: 『N』Z 『N』C 『N』AC 『N』OV
stopsys	System halt. Example: stopsys; Result: Stop the system clocks and halt the system Affected flags: 『N』Z 『N』C 『N』AC 『N』OV
stopexe	CPU halt. The oscillator module is still active to output clock, however, system clock is disabled to save power. Example: stopexe; Result: Stop the system clocks and keep oscillator modules active. Affected flags: 『N』Z 『N』C 『N』AC 『N』OV
reset	Reset the whole chip, its operation will be same as hardware reset. Example: reset; Result: Reset the whole chip. Affected flags: 『N』Z 『N』C 『N』AC 『N』OV
wdreset	Reset Watchdog timer. Example: wdreset ; Result: Reset Watchdog timer. Affected flags: 『N』Z 『N』C 『N』AC 『N』OV

Summary of Instructions Execution Cycle

2T	*goto, call, pcadd, ret, reti , idxm*
1T/2T	*ceqsn, cneqsn, t0sn, t1sn, dzsn, izsn*
1T	Others

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*	-	-	-	-	*ldt16 word*	-	-	-	-
stt16 word	-	-	-	-	*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	*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*	-	-	-	-	*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 IO, a	-	-	-	-	*xor a, M*	Y	-	-	-	*xor M, a*	Y	-	-	-
not a	Y	-	-	-	*not M*	Y	-	-	-	*neg a*	Y	-	-	-
neg M	Y	-	-	-	*set0 IO.n*	-	-	-	-	*set1 IO.n*	-	-	-	-
set0 M.n	-	-	-	-	*set1 M.n*	-	-	-	-	*ceqsn a, I*	Y	Y	Y	Y
ceqsn a, M	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
call label	-	-	-	-	*goto label*	-	-	-	-	*ret I*	-	-	-	-
ret	-	-	-	-	*reti*	-	-	-	-	*nop*	-	-	-	-
pcadd a	-	-	-	-	*engint*	-	-	-	-	*disgint*	-	-	-	-
stopsys	-	-	-	-	*stopexe*	-	-	-	-	*reset*	-	-	-	-
wdreset	-	-	-	-	*nadd M, a*	Y	Y	Y	Y	*ceqsn a, I*	Y	Y	Y	Y

Code Option Table

Option	Selection	Description
Security	Enable	Security 7/8 words Enable
	Disable	Security Disable
LVR	4.0V	LVR typical range 4.0V
	3.5V	LVR typical range 3.5V
	3.0V	LVR typical range 3.0V
	2.75V	LVR typical range 2.75V
	2.5V	LVR typical range 2.5V
	2.2V	LVR typical range 2.2V
	2.0V	LVR typical range 2.0V
	1.8V	LVR typical range 1.8V
Boot-up Time	Slow	About 3000 ILRC clock cycles
	Fast	About 45 ILRC clock cycles

Special Notes

This chapter is to remind user who use MCU401 IC in order to avoid frequent errors upon operation.

Using IC

IO pin usage and setting

(1) IO pin as digital input and enable wakeup function

- ◆ Configure IO pin as input
- ◆ Set PADIER and PBDIER registers to set the corresponding bit to 1.
- ◆ The functions of PADIER and PBDIER registers are contrary to ICE functions. Please use following program in order to keep ICE emulation consisting with MCU401 IC procedure.

```
$ PADIER 0xF0;
$ PBDIER 0x0F;
```

(2) PA5 is set to be output pin

- ◆ PA5 can be set to be Open-Drain output pin only, output high requires adding pull-up resistor.

(3) PA5 is set to be PRSTB input pin

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

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

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

Interrupt

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

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

Step2: Clear INTRQ register.

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

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

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

*Use DISGINT in the main program to disable all interrupts.

*When interrupt service routine starts, use PUSHAF instruction to save ALU and FLAG register. POPAF instruction is to restore ALU and FLAG register before RETI as below:

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

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

System clock switching

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

- Switch system clock from ILRC to IHRC/2

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

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

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

Power down mode, wakeup and watchdog

Watchdog will be inactive once ILRC is disabled.

TIMER time out

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

IHRC

(1) The IHRC frequency calibration is performed when IC is programmed by the writer.

(2) Because the characteristic of the Epoxy Molding Compound (EMC) would some degrees affects the IHRC frequency (either for package or COB), if the calibration is done before molding process, the actual IHRC frequency after molding may be deviated or becomes out of spec. Normally , the frequency is getting slower a bit.

(3) It usually happens in COB package or Quick Turnover Programming (QTP). And we would not take any responsibility for this situation.

(4) Users can make some compensatory adjustments according to their own experiences. For example, users can set IHRC frequency to be 0.5% ~ 1% higher and aim to get better re-targeting after molding.