SC8P115xA user manual

IO type OTP MCU

Rev. 1.3.1

Please be reminded about following CMS's policies on intellectual property

* Cmsemicron Limited(denoted as 'our company' for later use) has already applied for relative patents and entitled legal rights. Any patents related to CMS's MCU or other producrts is not authorized to use. Any individual, organization or company which infringes s our company's interlectual property rights will be forbidden and stopped by our company through any legal actions, and our company will claim the lost and required for compensation of any damage to the company.

* The name of Cmsemicron Limited and logo are both trademarks of our company.

* Our company preserve the rights to further elaborate on the improvements about products' function, reliability and design in this manual. However, our company is not responsible for any usage about this munal. The applications and their purposes in this manual are just for clarification, our company does not guarantee that these applications are feasible without further improvements and changes, and our company does not recommend any usage of the products in areas where people's safety is endangered during accident. Our company's products are not authorzed to be used for life-saving or life support devices and systems.our company has the right to change or improve the product without any notification, for latest news, please visit our website: www.mcu.com.cn

Content

Precautions for use

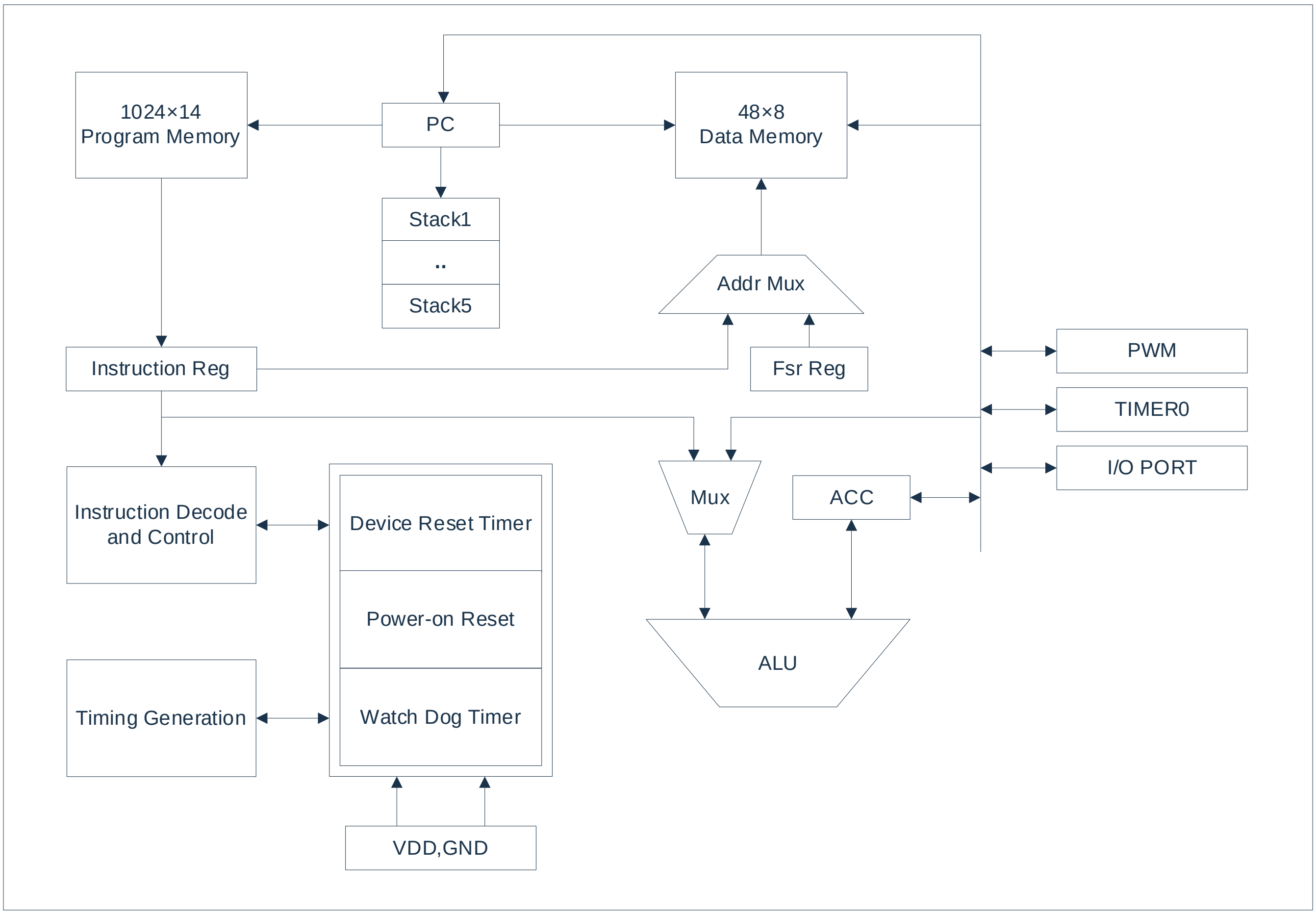
Serial number	Precautions	
1	PB3 pull-up is enabled, PB3 is floating, and the voltage of PB3 is about 0.7VDD.	
2	PB3 input level:	TA=25℃, VDD=5V, pull-up enable: VIH=1.7V, VIL=0.9V. TA=25℃, VDD=5V, pull-up disabled: VIH=2.2V, VIL=1.4V.
3	PB3 pull-up resistor:	TA=25℃, VIH=0.5VDD: RPH=36K±7K.
4	IRCF[2:0]=000:	At this time, the internal RC oscillator of the chip is stopped, and the PWM function is disabled.

1. Product Overview

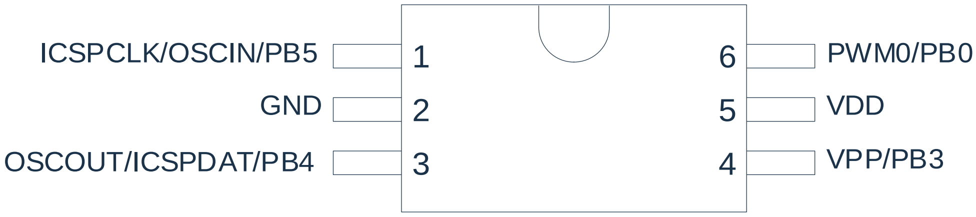
1.1 Functional characteristics and selection table

Model	SC8P1151A	SC8P1152A
Pin position	6	8
Package form	SOT23-6	SOP8
I/O	3+1	5+1
ROM (OTP)	1K×14Bit	1K×14Bit
RAM	48 bytes	48 bytes
External oscillation	32768 for timer	32768 for timer
Internal oscillator frequency (MHz)	8	8
Advanced PWM	0	0
Ordinary PWM	1	3
TIMER(8Bit)	1	1
External Interrupt	2	2
Internal interrupt	1	1
IO internal pull-up resistor	all IO	all IO
IO internal pull-down resistor	All IO (except PB3)	All IO (except PB3)
Wake	All interrupts, watchdog	All interrupts, watchdog
LVR	have	have
WDT	have	have
Stack	Level 5	Level 5
Instruction	60	60
Operating Voltage	2.0V-5.5V	2.0V-5.5V
working temperature	-25℃-85℃	-25℃-85℃

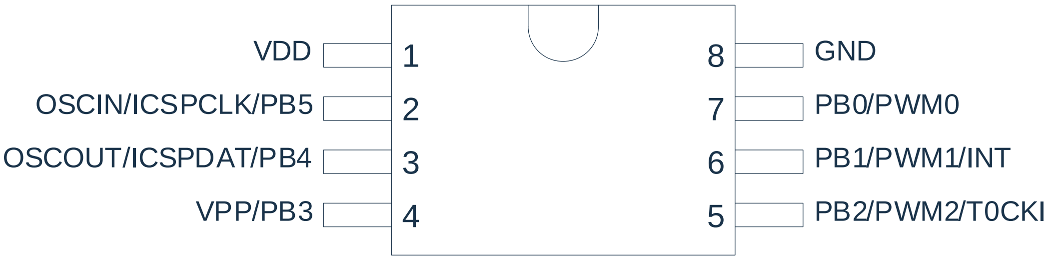
1.2 System structure block diagram



1.3 Pin assignment



SC8P1151A



SC8P1152A

Pin Description:

Pin name	IO type	Pin description	Shared pin
PB0-PB2 PB4,PB5	I/O	Programmable as: floating input port, input port with pull-up and pull-down resistor, push-pull output port, open-drain output port; can also be used as XT oscillator port, external interrupt input port, level change interrupt, timer input port, PWM output mouth.	OSCIN, OSCOUT, INT, T2CKI, PWM0, PWM1, PWM2
PB3	I/O	Programmable as: floating input port, input port with pull-up resistor, open-drain output port.	
VDD, GND	P	Power supply voltage input pin, ground pin	
PWM0-PWM2	O	PWM output port	
OSCIN，OSCOUT	I/O	XT oscillator port	
INT	I	External interrupt input port	
T0CKI	I	TMR0 timer/counter input	

1.4 System Configuration Register

The system configuration register (CONFIG) is an OTP option for the initial conditions of the MCU. It can only be programmed by company's programmer, and users cannot access and operate it. It contains the following:

1. WAKEUP_TIME（Start time selection after wake up）
   - 18ms The startup time after wake up is 18ms
   - 9ms The startup time after wake up is 9ms
   - 4.5ms The startup time after wake up is 4.5ms
   - 2.25ms The startup time after wake up is 2.25ms
2. WDT（watchdog selection）
   - ENABLE Turn on the watchdog timer
   - DISABLE Turn off watchdog timer
3. PROTECT（encryption）
   - DISABLE OTP codes are not encrypted
   - ENABLE OTP code encryption
4. LVR_SEL（Low voltage detection option）
   - 1.8V Low voltage detection voltage selection 1.8V
   - 2.8V Low voltage detection voltage selection 2.8V
   - DISABLE Disable low voltage detection
5. SLEEP_LVR（LVR state during sleep）
   - ENABLE Turn on LVR after hibernation
   - DISABLE Turn off LVR after hibernation

1.5 In-Circuit Serial Programming

The SC8P115xA microcontroller can be serially programmed in the final application circuit. Programming can be done simply with the following 5 wires:

- Power line (VDD)
- Ground wire (GND)
- Data cable (DAT)
- Data cable2 (DAT2)
- Clock line (CLK)
- High Voltage Line (VPP)

This allows users to build boards with unprogrammed devices and only program the microcontroller before the product is shipped. So that the latest version of firmware or custom firmware can be programmed into the microcontroller.

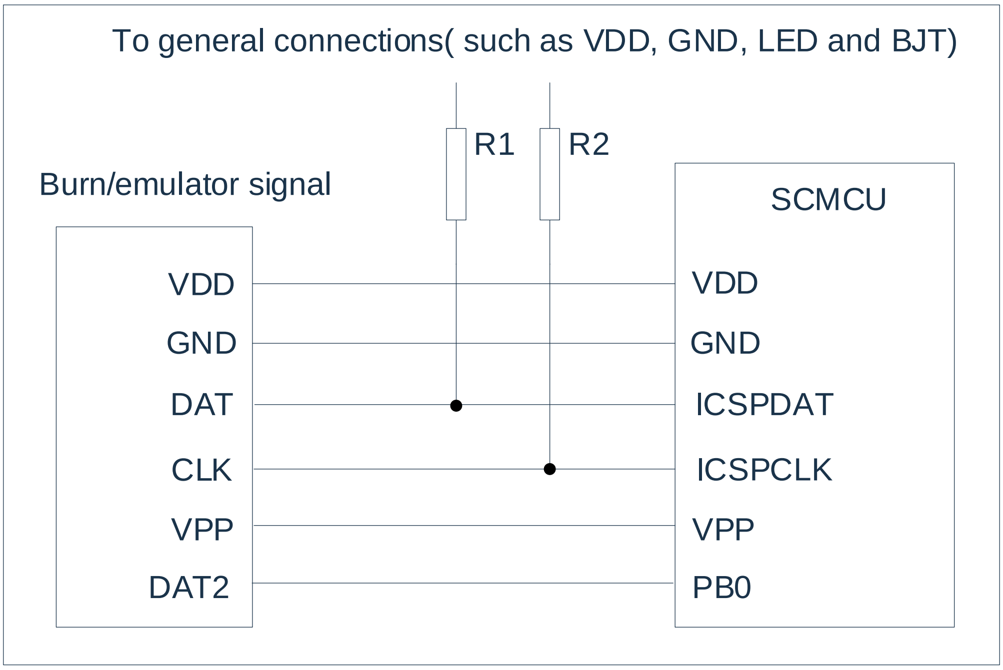


Figure 1-1: Typical In-Circuit Serial Programming Connection

In the above figure, R1 and R2 are electrical isolation devices, which are often replaced by resistors. The resistance values are as follows: R1≥4.7K, R2≥4.7K.

If the programming communication line is long, a small capacitor can be connected to the ground on the DAT and CLK pins to increase the stability of the communication, and the capacitance value cannot be greater than 100pF (101).

2. Central processing unit（CPU)

2.1 Memory

2.1.1 Program memory

OTP：1K

Address	Program memory	Description
0000H 0001H 0002H 0003H	reset vector	Program start, jump to user program
0004H	interrupt vector	Interrupt entry, user interrupt program
… … … 03FDH 03FEH	general storage area	User program area
03FFH	Jump to reset vector 0000H	Program ends

2.1.1.1 Reset vector (0000H)

SC8P115xA series microcontrollers have a word-length system reset vector (0000H). Has the following three reset methods：

- Power-on reset
- Watchdog reset
- Low Voltage Reset (LVR)

After any of the above resets occurs, the program will resume execution from 0000H, and the system registers will also be restored to their default values. The system reset mode can be judged according to the contents of PD and TO flag bits in the STATUS register. The following program demonstrates how to define the reset vector in ROM.

Example: Defining a Reset Vector

```
        ORG     0000H       ; System reset vector
        JP      START
        ORG     0010H       ; User program start
START:
        ...                 ; User program
        ...
        END                 ; Program ends
```

2.1.1.2 Interrupt vector

The interrupt vector address is 0004H. Once there is an interrupt response, the current value of the program counter PC will be stored in the stack buffer and jump to 0004H to start executing the interrupt service routine. All interrupts will enter the interrupt vector 0004H, and which interrupt to execute will be determined by the user according to the bits of the interrupt request flag register. The following sample program shows how to write an interrupt service routine.

Example: define the interrupt vector, the interrupt program is placed after the user program

```
            ORG     0000H       ; system reset vector
            JP      START
            ORG     0004H       ; start of interrupt program
INT_START:
            CALL    PUSH        ; Save ACC and STATUS
            ...                 ; user interrupt routine
            ...
INT_BACK:
            CALL    POP         ; Return ACC and STATUS
            RETI                ; Return from interrupt
START:
            ...                 ; User program
            ...
            END                 ; program ends
```

> Note: Since the SC8P115xA series chips do not provide dedicated pop and push instructions, the user needs to protect the interrupt site by himself.

Example: Interrupt protection scene

```
PUSH:
            LD      ACC         ; Save ACC to custom register ACC_BAK
            SWAPA   STATUS      ; Status register STATUS high and low byte swap
            LD      STATUS      ; Save to custom register STATUS_BAK
            RET                 ; Return
```

Example: Interrupted exit recovery scene

```
POP:
            SWAPA   STATUS_BAK  ; Swap the high and low byte of the data saved to STATUS_BAK to ACC
            LD      STATUS,A    ; Send the value of ACC to the status register STATUS
            SWAPR   ACC_BAK     ; Swap the high and low byte of the data saved to ACC_BAK
            SWAPA   ACC_BAK     ; Swap the high and low byte of the data saved to ACC_BAK to ACC
            RET                 ; return
```

2.1.1.3 Jump table

The jump table can achieve the multi-address jump function. Since the new PCL can be obtained by adding the values of PCL and ACC, multiple address jumps can be realized by adding different ACC values to PCL. If the ACC value is N, PCL+ACC means adding N to the current address. After the current instruction is executed, the PCL value will automatically increase by 1. Please refer to the following example. If an overflow occurs after PCL+ACC, the PC will not carry over automatically, so be careful when writing programs. In this way, users can easily realize multi-address jump by modifying the value of ACC.

PCLATH is the PC high-order buffer register. When operating on PCL, you must first assign a value to PCLATH.

Example: Correct multi-address jump program example

```
ROM address
              LDIA      00H
              LD        PCLATH,A     ; Must assign to PCLATH
              …
0010H:        ADDR      PCL          ;ACC+PCL
0011H:        JP        LOOP1        ;ACC=0, jump to LOOP1
0012H:        JP        LOOP2        ;ACC=1, jump to LOOP2
0013H:        JP        LOOP3        ;ACC=2, jump to LOOP3
0014H:        JP        LOOP4        ;ACC=3, jump to LOOP4
0015H:        JP        LOOP5        ;ACC=4, jump to LOOP5
0016H:        JP        LOOP6        ;ACC=5, jump to LOOP6
```

Example: wrong multi-address jump program example

```
ROM address
              LDIA      00H
              LD        PCLATH,A     ; must assign to PCLATH
              …
00FCH:        ADDR      PCL          ;ACC+PCL
00FDH:        JP        LOOP1        ;ACC=0, jump to LOOP1
00FEH:        JP        LOOP2        ;ACC=1, jump to LOOP2
00FFH:        JP        LOOP3        ;ACC=2, jump to LOOP3
0100H:        JP        LOOP4        ;ACC=3, jump to address 0000H
0101H:        JP        LOOP5        ;ACC=4, jump to address 0001H
0102H:        JP        LOOP6        ;ACC=5, jump to address 0002H
```

Note: Since PCL overflow will not automatically carry to the high order, when using PCL for multi-address jump, it should be noted that this segment of the program must not be placed in the paging of the ROM space.

2.1.2 Data memory

RAM: 74 byte

Address	data storage
000H	System register area
001H	
…	
…	
014H	
015H	System register area (Reserved)
…	
019H	
020H	General register area
021H	
022H	
…	
…	
04EH	
04FH	

The data memory consists of 74 × 8 bits and is divided into two functional areas: special function registers (26 × 8) and general-purpose data memory (48 × 8). Data memory cells are mostly read/write, but some are read only. The special function register package address is from 00H to 19H, and the general data register address is from 20H to 4FH.

2.1.2.1 General purpose data storage

The 020H~04FH addresses of the RAM belong to the general register area that the user can define freely, and the registers in this area are random values when powered on. After the system is powered on, if an accidental reset (non-power-on reset) occurs, this area register will keep the original value unchanged.

2.1.2.2 System dedicated data memory

Address	Name	Description
00H	INDF	Indirect addressing register
01H	TMR0	TMR0 data register
02H	PCL	Program pointer PC lower 8 bits
03H	STATUS	System Status Flag Register
04H	FSR	indirection pointer
05H	PORTB	PB port data register
06H	TRISB	PB port direction register
07H	OPTION_REG	Prescaler register
08H	OSCCON	Oscillator Control Register
09H	INTCON	Interrupt Control Register
0AH	PCLATH	Write buffer for the upper 2 bits of the program pointer PC
0BH	PDCONB	PB port pull-down resistor register
0CH	ODCONB	PB port open-drain output register
0DH	WPUB	PB port pull-up resistor register
0EH	IOCB	PB port level change interrupt register
0FH	TMR0PRD	TMR0 Period Register
10H	PWMCTR0	PWM Control Register 0
11H	PWMCTR1	PWM Control Register 1
12H	PWMCTR2	PWM Control Register 2
13H	PWMR	PWM Duty Cycle Indirect Addressing Register
14H	PWMPRD	PWM Period Register
15H	-	-
16H	-	-
17H	-	-
18H	-	-
19H	-	-

2.2 Addressing mode

2.2.1 Direct addressing

The RAM is operated through the working register (ACC).

Example: The value of ACC is sent to the 30H register

```
LD          30H,A
```

Example: The value of the 30H register is sent to the ACC

```
LD          A,30H
```

2.2.2 Immediate addressing mode

Transfer immediate data to working register (ACC)

Example: immediate data 12H is sent to ACC

```
LDIA        12H
```

2.2.3 Indirect addressing

Data memory can be addressed directly or indirectly. Indirectly addressable through the INDF register, which is not a physical register. When accessing the INDF, it will use the value in the FSR register as the address and point to the register of the address, so after the FSR register is set, the INDF register can be accessed as a destination register. An indirect read of INDF (FSR=0) will generate 00H. An indirect write to the INDF register will result in a no-op. The following example illustrates the use of indirect addressing in a program.

Example: Application of FSR and INDF

```
        LDIA        30H
        LD          FSR,A       ; Indirect addressing pointer points to 30H
        CLR         INDF        ; Clearing INDF is actually clearing the 30H address RAM
                                pointed to by FSR
```

Example: Indirect addressing clear RAM (20H-4FH) example:

```
        LDIA        1FH
        LD          FSR,A       ; Indirect addressing pointer points to 1FH
LOOP:
        INCR        FSR         ; Add 1 to the address, the initial address is 20H
        CLR         INDF        ; Clear the address pointed to by the FSR
        LDIA        4FH
        SUBA        FSR
        SNZB        STATUS,C    ; Cleared until the FSR address is 4FH
        JP          LOOP
```

2.3 Stack

The stack buffer of SC8P115xA has a total of 5 layers. The stack buffer is neither a part of the data memory nor a part of the program memory, and can neither be read nor written. Its operation is realized by the stack pointer (SP), and the stack pointer (SP) cannot be read or written. When the system is reset, the stack pointer will point to the top of the stack. When a subroutine call and interrupt occur, the program counter (PC) value is pushed into the stack buffer, and the value is returned to the program counter (PC) when returning from an interrupt or subroutine. The following figure illustrates how it works.

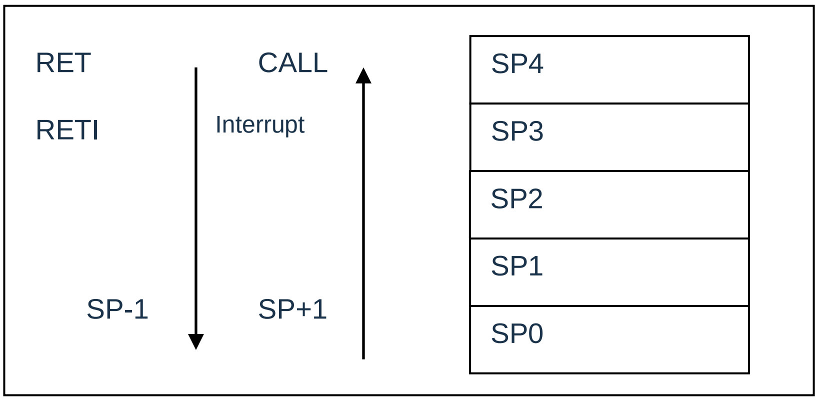


Figure 2-1: How the stack buffer works

The use of stack buffers will follow a FILO principle.

Note: The stack buffer has only 5 layers. If the stack is full and a non-maskable interrupt occurs, only the interrupt flag will be recorded, and the interrupt response will be suppressed. The interrupt will not be responded until the stack pointer is decremented; this function can prevent the interrupt from overflowing the stack. Similarly, if the stack is full and a subroutine call occurs, the stack will overflow. The content that entered the stack first will be lost. Only the last 5 return addresses are reserved. This point should be paid attention to when writing the program to avoid the program running away.

2.4 Working Register（ACC）

2.4.1 General

ALU is an 8-bit wide arithmetic logic unit, through which all mathematical and logical operations of the MCU are completed. It can add, subtract, shift and logical operations on the data; ALU also controls the status bit (in the STATUS status register), which is used to indicate the status of the operation result.

The ACC register is an 8-bit register. The operation result of the ALU can be stored here. It is not part of the data memory but is located in the CPU for the ALU to use in the operation. Therefore, it cannot be addressed and can only be addressed by the provided instruction. to use.

2.4.2 ACC application

Example: use ACC for data transfer

```
LD      A,R01     ; Assign the value of register R01 to ACC
LD      R02,A     ; Assign the value of ACC to register R02
```

Example: Use ACC as immediate addressing target operand

```
LDIA    30H       ; Assign 30H to ACC
ANDIA   30H       ; Perform an AND operation between the current ACC
                  value and the immediate value 30H,
XORIA   30H       ; The result is put into ACC
```

Example: Use ACC as the second operand of a two-operand instruction

```
SUBA    R01       ;R01-ACC, the result is put into ACC
SUBR    R01       ; R01-ACC, the result is put into R01
```

2.5 Program Status Register（STATUS)

The register STATUS contains ALU operation status information and system reset status information. Among them, bits TO and PD display system reset status information, including power-on reset, external reset and watchdog reset, etc.; bits C, DC and Z display the operation information of ALU.

03H	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
STATUS	---	---	---	TO	PD	Z	DC	C
Read/Write	---	---	---	R/W	R/W	R/W	R/W	R/W
Reset value	---	---	---	1	1	X	X	X

Bit7-Bit5	Not use.	
Bit4	TO:	Timeout bit;
	1=	Power on or execute the CLRWDT instruction or STOP instruction;
	0=	A WDT timeout occurred.
Bit3	PD:	drop potential;
	1=	Power on or execute the CLRWDT instruction;
	0=	A STOP instruction was executed.
Bit2	Z:	The result is zero bits;
	1=	the result of an arithmetic or logical operation is zero;
	0=	The result of an arithmetic or logical operation is not zero.
Bit1	DC:	half-carry/borrow bit;
	1=	The 4th low order of the result is carried to the high order;
	0=	The 4th low-order bit of the result is not carried over to the high-order bit.
Bit0	C:	carry/borrow bit;
	1=	The most significant bit of the result is carried over;
	0=	The most significant bit of the result is not carried over.

Except for the TO and PD bits in the STATUS register, all other bits can be set or cleared by instructions. For example, the result of the instruction: "CLRSTATUS" is STATUS="xxx00100", not all zeros as imagined. That is to say, after the instruction is executed, the values of TO and PD remain unchanged, and the Z flag bit is set to 1 due to clearing, so if you need to change the value of STATUS, it is recommended to use "SETB", "CLRB", "SWAPA", " SWAPR" these instructions, because these instructions do not affect the status flag bits.

The TO and PD flag bits can reflect the reason for the chip reset. The events affecting TO and PD and the states of TO and PD after various resets are listed below.

Event	TO	PD
power up	1	1
WDT overflow	0	X
STOP command	1	0
CLRWDT instruction	1	1
hibernate	1	0

Table of Events Affecting TO/PD

TO	PD	Reset reason
0	0	WDT overflow wakes up sleeping MCU
0	1	WDT overflow non-sleep state
1	1	power up

Status of TO/PD after reset

2.6 Prescaler（OPTION_REG）

The prescaler (OPTION_REG) register is an 8-bit, readable and writable register that contains various control bits for configuring the TMR0/WDT prescaler and TMR0.

07H	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
OPTION_REG	XT_EN	T0SYNC	T0CS	T0SE	PSA	PS2	PS1	PS0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset value	0	1	1	1	1	0	1	1

Bit7 XT_EN: Crystal enable bit.
- 0: Do not enable crystal oscillator.
- 1: Enable crystall oscillator

Bit6 T0SYNC: External clock and internal clock asynchronous enable bit.
- 0: External clock and internal clock synchronization.
- 1: Asynchronous external clock and internal clock.

Bit5 T0CS: TMR0 clock source selection bit.
- 0: Internal clock (FOSC/4).
- 1: External clock (T0CKI port input waveform or external crystal oscillator, depending on Bit7 XT_EN).

Bit4 T0SE: T0CKI signal triggers edge select bit.
- 0: rising edge trigger.
- 1: falling edge trigger.

Bit3 PSA: Prescaler assignment bits.
- 0: Allocated to TMR0.
- 1: Allocated to WDT.

Bit2~Bit0 PS2~PS0: Pre-allocated parameter configuration bits.

PS2	PS1	PS0	TMR0 frequency division ratio	WDT frequency division ratio
0	0	0	1:2	1:1
0	0	1	1:4	1:2
0	1	0	1:8	1:4
0	1	1	1:16	1:8
1	0	0	1:32	1:16
1	0	1	1:64	1:32
1	1	0	1:128	1:64
1	1	1	1:256	1:128

The prescaler register is actually a 8-bit counter. When it is used to monitor the register WDT, it is used as a postscaler; when used in a timer/counter, as a prescaler, usually collectively referred to as a prescaler. There is only one physical frequency divider in the chip, which can only be used for WDT or TMR0, and the two cannot be used at the same time. That is, if used for TMR0, the WDT cannot use the prescaler, and vice versa.

When used with WDT, the CLRWDT instruction will clear both the prescaler and the WDT timer.

When used for TMR0, all instructions related to writing to TMR0 (eg: CLR TMR0, SETB TMR0, 1, etc.) will clear the prescaler.

Whether the prescaler is used by TMR0 or WDT is completely software controlled. He can change dynamically. In order to avoid unnecessary chip resets, when switching from TMR0 to WDT, the following commands should be executed.

```
CLR      TMR0             ; TMR0 clear
CLRWDT                    ; WDT clear
LDIA     B'00xx1111'      ; Necessary steps that must be performed
LD       OPTION_REG, A    ; Necessary steps that must be performed
LDIA     B'00xx1xxx'      ; set new prescaler
LD       OPTION_REG, A
```

To switch the prescaler from being assigned to the WDT to being assigned to the TMR0 module, the following instruction should be executed.

```
CLRWDT                    ; WDT clear
LDIA     B'00xx0xxx'      ; set new prescaler
LD       OPTION_REG, A
```

Note: To obtain a 1:1 prescaler configuration for TMR0, the prescaler can be assigned to the WDT by setting the PSA bit of the option register.

2.7 Program Counter（PC）

The program counter (PC) controls the execution sequence of instructions in the program memory. It can address the entire range of ROM. After the instruction code is obtained, the program counter (PC) will automatically increase by one to point to the address of the next instruction code. But if perform jumps, conditional jumps, assignments to PCL, subroutine calls, initialization resets, interrupts, interrupt returns, subroutine returns, etc., the PC will load the address associated with the instruction instead of the address of the next instruction.

When a conditional jump instruction is encountered and the jump condition is met, the next instruction read during the execution of the current instruction will be discarded, and an empty instruction operation cycle will be inserted before the correct instruction can be obtained. Otherwise, the next instruction will be executed sequentially.

> Note: When the programmer uses PCL to make short jumps, the PC high-order buffer register PCLATH must be assigned a value first.

The PC values for several special cases are given below

When reset	PC=0000;
when interrupted	PC=0004 (the original PC+1 will be automatically pushed into the stack);
RET i, RET, RETI	PC=value from stack;
When operating PCL	PC[9:8]=PCLATH, PC[7:0]=user-specified value;
When JP	PC = the value specified by the program;
Other instructions	PC=PC+1;

2.8 Watchdog counter（WDT)

The Watch Dog Timer is an on-chip self-oscillating RC oscillator timer that does not require any peripheral components. Even if the main clock of the chip stops working, the WDT can keep timing. A WDT time-out will generate a reset. The CONFIG option is integrated in the SC8P115xA series chip, and the WDT can be disabled by setting. For details, please refer to the description of CONFIG option in chapter 1.5. When using it, it should be noted that when WDT is disabled, the CONFIG option and WDTEN must be turned off at the same time.

2.8.1 WDT cycle

WDT has a basic overflow period of 18ms (without prescaler). If you need a longer WDT period, you can assign the prescaler to WDT. The maximum frequency division ratio is 1:128. At this time, the period of WDT About 2.3s. The overflow period of WDT will be affected by parameters such as ambient temperature, power supply voltage, etc.

The "CLRWDT" and "STOP" instructions will clear the WDT timer and the count value in the prescaler (when the prescaler is assigned to the WDT). WDT is generally used to prevent the system from running out of control, or it can be said to prevent the microcontroller program from running out of control. Under normal circumstances, the WDT should be cleared by the "CLRWDT" instruction before it overflows to prevent a reset. If the program runs out of control due to some disturbance, the "CLRWDT" instruction cannot be executed before the WDT overflows, and the WDT overflows and resets. Reboot the system without losing control. If the reset is caused by WDT overflow, the "TO" bit of the status register (STATUS) will be cleared, and the user can judge whether the reset is caused by WDT overflow according to this bit.

Note:

1. If the WDT function is used, the "CLRWDT" instruction must be placed in some places in the program to ensure that the WDT can be cleared before overflow. Otherwise, the chip will be reset continuously, causing the system to not work properly.
2. The WDT cannot be cleared in the interrupt program, otherwise the "running" of the main program cannot be detected.
3. In the program, there should be an operation to clear the WDT in the main program. Try not to clear the WDT in multiple branches. This architecture can maximize the protection function of the watchdog counter.
4. The overflow time of different chips of the watchdog counter is different, so when setting the clear WDT time, there should be a greater redundancy with the WDT overflow time to avoid unnecessary WDT reset.

3. System clock

3.1 Overview

The clock signal is generated by internal oscillation, and four non-overlapping quadrature clock signals are generated in the chip, which are called Q1, Q2, Q3, and Q4 respectively. Inside the IC, each Q1 increments the program counter (PC) by one, and Q4 fetches the instruction from the program memory unit and latches it in the instruction register. The fetched instruction is decoded and executed between the next Q1 to Q4, which means that one instruction will only be executed in 4 clock cycles. The following figure shows the timing diagram of clock and instruction cycle execution.

An instruction cycle contains 4 Q cycles. The execution and fetching of instructions adopts a pipeline structure. Instruction fetching occupies one instruction cycle, while decoding and execution occupy another instruction cycle. However, due to the pipeline structure, from a macro point of view, the effective execution time is one instruction cycle. If an instruction causes the program counter address to change (such as JP), then the prefetched instruction opcode is invalid, and two instruction cycles are required to complete the instruction, which is why the PC operation instruction takes two clock cycles.

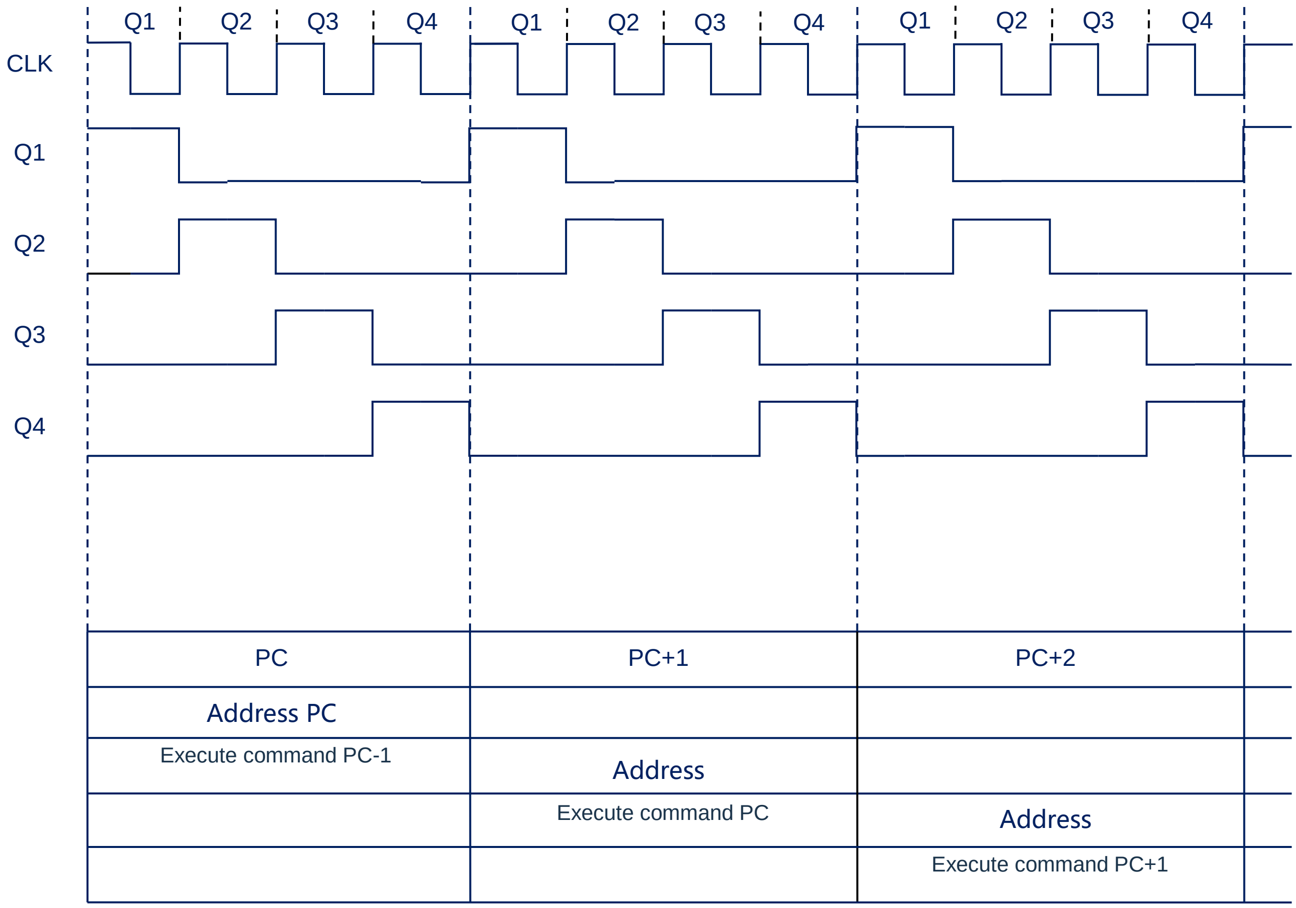


Figure 3-1: Clock and Instruction Cycle Timing Diagram

The relationship between oscillation frequency and command speed is listed below

Frequency	Double instruction cycle	Single instruction cycle
1MHz	8us	4us
2MHz	4us	2us
4MHz	2us	1us
8MHz	1us	500ns

3.2 System oscillator

SC8P115xA has only one oscillation mode: internal RC oscillation.

3.2.1 Internal RC oscillation

The default oscillation mode of the chip is internal RC oscillation, and its oscillation frequency is 8M.

3.3 Oscillator Control Register

The Oscillator Control (OSCCON) register controls the system clock, frequency selection, WDT enable and TMR0 enable.

08H	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
OSCCON	SWDTEN	IRCF2	IRCF1	IRCF0	---	---	---	TMR0EN
Read/Write	R/W	R/W	R/W	R/W	---	---	---	R/W
Reset value	1	1	1	0	---	---	---	0

Bit7 SWDTEN: WDT enable bit.
0: Disable WDT.
1: Enable WDT.

Bit6~Bit4 IRCF2~0: Frequency selection configuration bits.

IRCF2	IRCF1	IRCF0	Core clock divider
0	0	0	F_{WDT}(8K)
0	0	1	F_{OSC}/64
0	1	0	F_{OSC}/32
0	1	1	F_{OSC}/16
1	0	0	F_{OSC}/8
1	0	1	F_{OSC}/4
1	1	0	F_{OSC}/2
1	1	1	F_{OSC}

Bit6~Bit4 Not use

Bit0 TMR0EN: TMR0 enable bit.
0: Disable TMR0.
1: Enable TMR0.

3.4 Start-up time

The start-up time (OSC TIME) refers to the time from the chip reset to the stable oscillation of the chip, which is fixed at 18ms. This start-up time will exist regardless of whether the chip is reset due to power-on or other reasons.

4. Reset

SC8P115xA can use the following 3 reset methods:

- Power-on reset.
- Low voltage reset (LVR enabled).
- Watchdog overflow reset under normal operation.

When any of the above resets occurs, all system registers will be restored to their default states, the program will stop running, and the program counter PC will be cleared to zero. After the reset, the program will start to run from the reset vector 0000H. The PD and TO flag bits of STATUS can give information about the system reset status (see the description of STATUS for details). Users can control the program running path according to the status of PD and TO.

Any reset situation requires a certain response time. The system provides a complete reset process to ensure the smooth progress of the reset action.

4.1 Power on reset

Power-on reset is closely related to LVR operation. The power-on process of the system is in the form of a gradually rising curve, and it takes a certain amount of time to reach the normal level. The normal sequence of power-on reset is given below:

- Power on: the system detects the rise of the power supply voltage and waits for it to stabilize;
- System initialization: all system registers are set to initial values.
- The oscillator starts working: the oscillator starts to provide the system clock.
- Execute the program: After power-on, the program starts to run.

4.2 Power down reset

A power-down reset is used for system voltage dips caused by external factors (eg, disturbances or changes in external loads). When using external reset, power-down reset may cause abnormal system working state or program execution error, voltage drop may enter system dead zone, system dead zone means that the power supply cannot meet the minimum operating voltage requirements of the system.

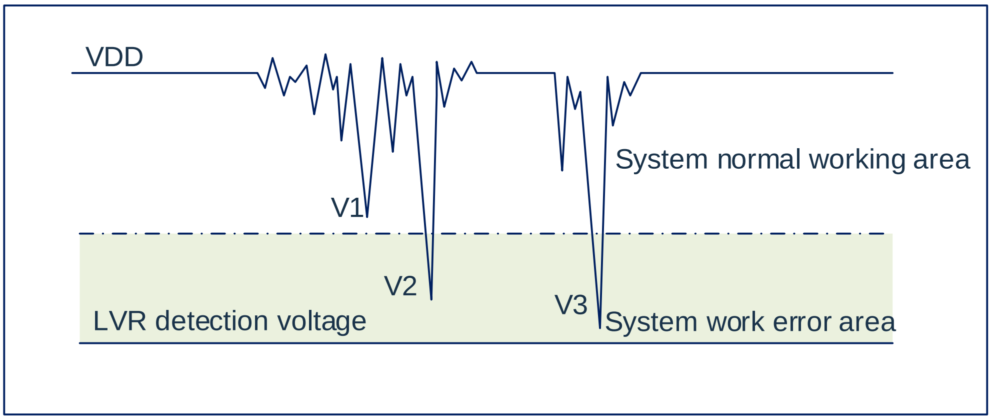


Figure 4-1: Schematic diagram of power-down reset

The figure above is a typical power-down reset schematic. In the figure, VDD is seriously disturbed, and the voltage drop is very low. In the area above the dotted line, the system works normally, and in the area below the dotted line, the system enters an unknown working state, which is called the dead zone. When VDD drops to V1, the system is still in a normal state; when VDD drops to V2 and V3, the system enters the dead zone, which is prone to errors.

The system may enter the dead zone under the following conditions:

- DC in use：
  - Battery power is generally used in DC applications. When the battery voltage is too low or the microcontroller drives the load, the system voltage may drop and enter the dead zone. At this time, the power supply will not drop further to the LVD detection voltage, so the system remains in the dead zone.
- AC in use：
  - When the system is powered by AC, the DC voltage value is affected by noise in the AC power supply. When the external load is too high, such as driving a motor, the interference generated by the load action also affects the DC power supply. If VDD drops below the minimum operating voltage due to interference, the system may enter an unstable working state.
  - In AC application, the system power-on and power-off times are long. Among them, the power-on sequence protection enables the system to be powered on normally, but the power-off process is similar to the situation in DC application. After the AC power supply is turned off, the VDD voltage tends to enter the dead zone during the process of slow decline.

As shown in the figure above, the normal operating voltage region of the system is generally higher than the system reset voltage, and the reset voltage is determined by the low voltage detection (LVR) level. When the system execution speed increases, the minimum system operating voltage also increases accordingly, but since the system reset voltage is fixed, there will be a voltage area between the system minimum operating voltage and the system reset voltage, and the system cannot work normally, nor will reset, this area is the dead zone.

4.2.1 Improvement method of power-down reset

The following suggestions are given:

- Enable the low voltage detection function of MCU;
- Turn on the watchdog timer;
- Reduce the operating frequency of the system;
- Increase the voltage drop slope.

Enable the low voltage detection function of the MCU

SC8P115xA series chips have integrated low voltage detection (LVR) function, which can be controlled by programming CONFIG. For details, see Chapter 1.5 about programming CONFIG selection instructions. When the LVR function is enabled, when the system voltage drops below the LVR voltage, the LVR is triggered, and the system is reset. Since the LVR voltage is always higher than the minimum operating voltage of the chip, there is no system operating dead zone.

Watchdog timer

The watchdog timer is used to ensure the normal operation of the program. When the system enters the working dead zone or the program runs in error, the watchdog timer will overflow, and the system will be reset.

Reduce the operating frequency of the system

The faster the operating frequency of the system, the higher the minimum operating voltage of the system. Therefore, the scope of the working dead zone is increased, and the minimum working voltage can be reduced by reducing the working speed of the system, thereby effectively reducing the probability of the system working at the dead zone voltage.

Increase the voltage drop slope

This method can be used in an environment where the system works in an AC power supply. Generally, in an AC power supply system, the system voltage drops very slowly during the power-down process, which will cause the chip to work at the dead zone voltage for a long time. The working state of the chip may be wrong. It is recommended to add a discharge resistor between the chip power supply and the ground wire, so that the MCU can quickly pass through the dead zone and enter the reset zone to avoid the possibility of chip power-on errors.

4.2.2 Watchdog reset

The watchdog reset is a protection setting of the system. Under normal conditions, the watchdog timer is cleared by the program. If an error occurs, the system is in an unknown state, the watchdog timer overflows, and the system is reset at this time. After the watchdog is reset, the system restarts into a normal state.

The timing of watchdog reset is as follows:

- Watchdog timer status: the system detects whether the watchdog timer overflows, and if it overflows, the system resets;
- Initialization: All system registers are set to default state;
- The oscillator starts working: the oscillator starts to provide the system clock;
- Program: reset ends, program starts running

For the application of the watchdog timer, please refer to chapter 2.8 WDT application.

5. System working mode

SC8P115xA series MCU has two working modes, one is normal working mode, and the other is sleep working mode. In the normal working mode, each functional module is in the working state. In the sleep state, the system clock stops, and the chip keeps the original state unchanged. At this time, if the WDT function is not disabled by the programming CONFIG option, the WDT timer will always be Work.

5.1 Sleep mode

The power saving mode is started by the STOP instruction. In the power saving mode, the system oscillation is stopped to reduce power consumption, and all peripherals stop working. Power saving mode can be woken up by reset, WDT overflow or interrupt. When the power saving mode is woken up, the clock circuit still needs oscillation stabilization time. When the power-saving mode is awakened by reset, the system starts to execute the program from the address 0000H; when the power-saving mode is awakened by an interrupt or WDT overflow, the PC starts to execute the program from the next address of the STOP instruction.

5.1.1 Sleep mode application example

Before the system enters the sleep mode, if the user needs to obtain a small sleep current, please confirm the status of all I/Os. Each input port has a fixed state to prevent the port line level in an indeterminate state and increase the sleep current when the I/O is in the input state; according to the functional requirements of the actual solution, the WDT function can be disabled to reduce the sleep current.

Example: A handler that goes to sleep

```
SLEEP_MODE:
        LDIA        B'00000000'
        LD          TRISB, A        ; PB port is set as output port
        ...                         ; Each output port is set to no load state
        LDIA        0A5H
        LD          SP_FLAG, A      ;Set the sleep state memory register (user-defined)
        CLRWDT                      ;clear WDT

        STOP                        ; Execute the STOP instruction
```

5.1.2 Wake-up from sleep mode

When the system is in sleep state, there are the following four conditions that can make the CPU exit the sleep state：

- Watchdog overflow;
- External interrupt;
- TMR0 timing interrupt (the crystal oscillator enable must be turned on);
- After the system is powered off, power it on again.

In the dormant MCU, when an interrupt or watchdog overflow occurs, the chip starts to run the program from the next address of the STOP instruction. When other situations occur, the chip will start to run the program from the reset address (0000H), and the user can decide the reset according to the TO and PD flags of STATUS and SP_FLAG (the user should define it by himself).

Example: Sleep wakes up user handler

```
            ORG     0000H
            JP      START        ; go to reset handler
            ORG     0004H
            JP      INT_START    ; go to interrupt handler
            ORG     0010H
START:                           ; reset handler
            SZB     STATUS, PD
            JP      START_2      ; Not a handler for reset from sleep
            SZB     STATUS, TO
            JP      START_3      ; Non-WDT wake-up MCU handler
            ...
            ...
            ...
SLEEP_MODE:                      ; sleep subroutine
            ...                  ; Set the state before sleep
            STOP                 ; The chip enters the SLEEP state handler
            NOP                  ; Press the key to wake up, add an empty command and wait for the clock to stabilize
            JP      XXXX         ; Press the key to wake up the program that should be handled
```

5.1.3 Sleep mode wake-up time

When the MCU wakes up from the sleep state, it needs to wait for an oscillation stabilization time (OSC TIME), which can be controlled by programming CONFIG. For details, see Chapter 1.5 about programming CONFIG selection instructions.

6. I/O port

The SC8P115xA has two sets of I/O ports: PORTB (up to 6 I/Os). The readable and writable port data registers provide direct access to these ports.

Port	bit	8 pin	6 pin	Pin description	I/O
PORT B	PB0	7	/	Schmitt trigger input, push-pull output, open-drain output, PWM0	I/O
	PB1	6	/	Schmitt trigger input, push-pull output, open-drain output, PWM1, INT	I/O
	PB2	5	3	Schmitt trigger input, push-pull output, open-drain output, PWM2, T0CKI	I/O
	PB3	4	4	Schmitt trigger input, open drain output, VPP	I/O
	PB4	3	1	Schmitt trigger input, push-pull output, open-drain output, OSCIN	I/O
	PB5	2	6	Schmitt trigger input, push-pull output, open-drain output, OSCOUT	I/O

Table 6-1: Overview of port configuration

6.1 I/O port mode and pull-up and pull-down resistors

Registers PORTB, TRISB, PDCONB, ODCONB, WPUB, IOCB are used to control the working mode of the I/O line.

6.1.1 PORTB port

The PORTB port has 8Bit input and output pins. There are 6 registers related to it, namely IO port data register (PORTB), IO port direction control register (TRISB), IO port pull-up control register (WPUB), IO port pull-down control register (PDCONB), IO port open-drain output control register (ODCONB), IO port level change interrupt control register (IOCB).

PORTB port data register PORTB(05H)

05H	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
PORTB	PB7	PB6	PB5	PB4	PB3	PB2	PB1	PB0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reser value	X	X	X	X	X	X	X	X
definition	I/O	I/O	I/O	I/O	I/O	I/O	I/O	I/O
						PWM2	PWM1	PWM0
			OSCOUT	OSCIN		T0CKI	INT	

PORTB port direction register TRISB(06H)

06H	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
TRISB	TRISB7	TRISB6	TRISB5	TRISB4	TRISB3	TRISB2	TRISB1	TRISB0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset value	1	1	1	1	1	1	1	1

Bit7~Bit0 PORTB direction
0: output;
1: Input.

PORTB port pull-up register WPUB(0DH)

0DH	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
WPUB	WPUB7	WPUB6	WPUB5	WPUB4	WPUB3	WPUB2	WPUB1	WPUB0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset value	0	0	0	0	1	0	0	0

Bit7~Bit0 PORTB pull-up resistor enabled.
0: Disable pull-up resistors;
1: Enable pull-up resistors (must turn off the pull-down enable for the corresponding bit).

PORTB port pull-down register PDCONB(0BH)

0BH	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
PDCONB	PDCONB7	PDCONB6	PDCONB5	PDCONB4	---	PDCONB2	PDCONB1	PDCONB0
R/W	R/W	R/W	R/W	R/W	---	R/W	R/W	R/W
Reset value	0	0	0	0	---	0	0	0

Bit7~Bit0 PORTB pull-down resistor enable.
- 0: Disable pull-down resistors；
- 1: Enable pull-down resistor (must turn off the pull-up enable for the corresponding bit).

PORTB port open-drain output register ODCONB(0CH)

0CH	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
ODCONB	ODCONB7	ODCONB6	ODCONB5	ODCONB4	---	ODCONB2	ODCONB1	ODCONB0
R/W	R/W	R/W	R/W	R/W	---	R/W	R/W	R/W
Reset value	0	0	0	0	---	0	0	0

Bit7~Bit0 PORTB open drain output enable.
- 0: Disable open-drain output；
- 1: Enable open-drain output.

PORTB port level change interrupt register IOCB(0EH)

0EH	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
IOCB	IOCB7	IOCB6	IOCB5	IOCB4	IOCB3	IOCB2	IOCB1	IOCB0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset value	0	0	0	0	0	0	0	0

Bit7~Bit0 PORTB level change interrupt enable.
- 0: Disable level change interrupt；
- 1: Enable level change interrupt.

6.2 The use of I/O ports

6.2.1 Write I/O port

The I/O port registers of SC8P115xA series chips, like general general registers, can be written through data transfer instructions, bit operation instructions, etc.

Example: write I/O port program

```
LD      PORTB, A    ; The ACC value is assigned to the PORTB port
CLRB    PORTB,0     ; PB0 port is cleared to zero
```

6.2.2 Read I/O port

Example: Read I/O port program

```
LD      A, PORTB    ; The value of PORTB is assigned to ACC
SZB     PORTB,1     ; Determine whether the PB1 port is 0, if it is 0, skip the next statement
```

Note: When the user reads the status of an I/O port, if the I/O port is an input port, the data read back by the user will be the state of the external level of this port line. If this I/O port is an output port, then the read value will be the data of the internal output register of this port line.

6.3 I/O port usage precautions

When operating the I/O port, the following aspects should be paid attention to:

1. When the I/O is converted from output to input, wait a few instruction cycles for the I/O port to stabilize.
2. If the internal pull-up resistor is used, when the I/O is converted from output to input, the stabilization time of the internal level is related to the capacitor connected to the I/O port. The user should set the waiting time according to the actual situation to prevent the I/O port from accidentally scanning the level.
3. When the I/O port is an input port, its input level should be between "VDD+0.7V" and "VSS-0.7V". If the input port voltage is not within this range, the method shown in the figure below can be used.

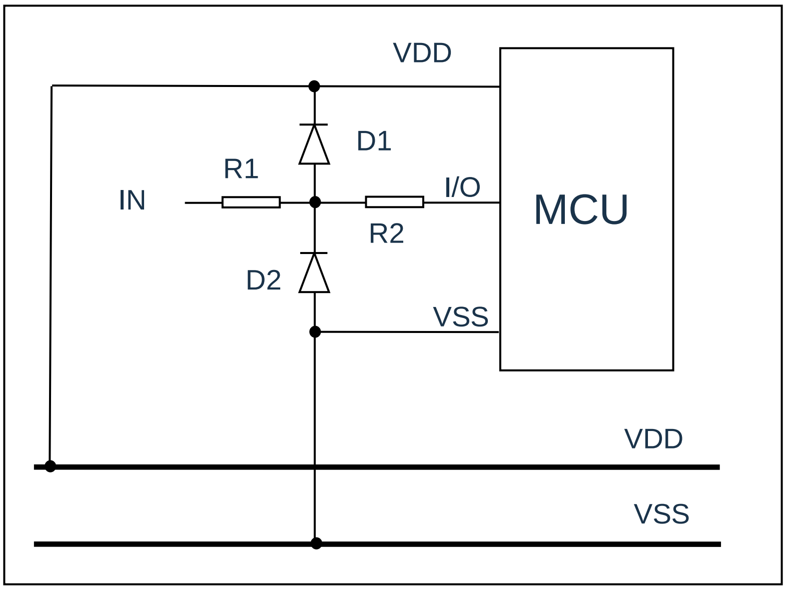


Figure 6-1: Circuit using input voltage not within specified range

4. If the I/O port is connected in series with a long cable, please add a current limiting resistor near the chip I/O to enhance the MCU's EMC resistance.
5. If the PB3 port is used as the signal input port, it is recommended to use the following method to enhance the MCU's resistance to EMC and ESD

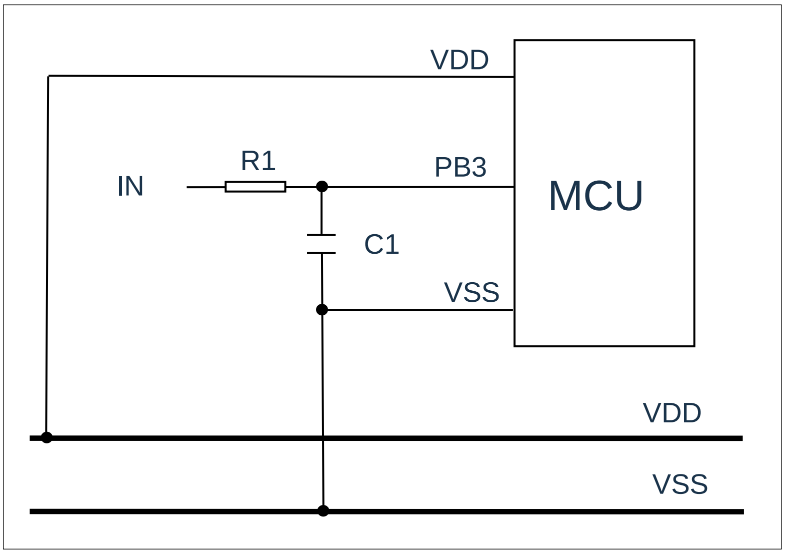


Figure 6-2: PB3 port as signal input port

7. Interrupt

7.1 Interrupt overview

SC8P115xA has 3 interrupt sources: 1 internal interrupt (TMR0) and 2 external interrupts (INT, IOCB). Once the program enters the interrupt, the GIE bit in the INTCON register will be automatically cleared by hardware to avoid re-serving other interrupts. The system exits the interrupt, that is, after executing the RETI instruction, the hardware automatically sets GIE to "1" to respond to the next interrupt. The interrupt request is stored in the register INTCON register.

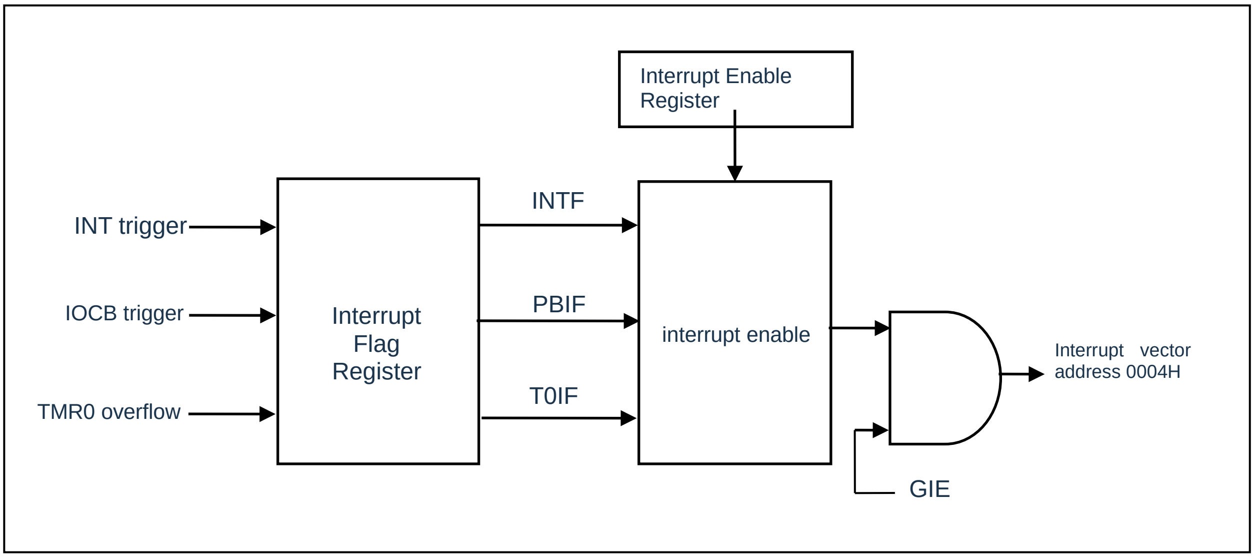


Figure 7-1: Interrupt System

7.2 Interrupt Control Register

The interrupt request control register INTCON includes all interrupt enable control bits and flags bits.

The valid bit of GIE and corresponding interrupt is set to "1", then the system enters the interrupt service routine, the program counter is pushed into the stack, and the program transfers to 0004H, that is, the interrupt program. When the program runs to the instruction RETI, the interrupt ends, and the system exits the interrupt service.

Once an interrupt request occurs, the corresponding interrupt flag will be set to "1". After the request is responded, the program should clear the flag, and the MCU will not automatically clear the interrupt request flag.

Interrupt Control Register INTCON (09H)

09H	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
INTCON	GIE	INTEG	T0IE	INTE	PBIE	T0IF	INTF	PBIF
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset value	0	0	0	0	0	0	0	0

Bit7	GIE:	Global interrupt enable bit enable bit;
	0:	disable all interrupts;
	1:	All unmasked interrupts are enabled.
Bit6	INTEG:	External interrupt edge selection;
	0:	Rising edge trigger;
	1:	Falling edge trigger.
Bit5	T0IE:	TIMER0 overflow interrupt enable bit;
	0:	Disable TIMER0 interrupt;
	1:	Enable TIMER0 interrupt.
Bit4	INTE:	INT external interrupt enable bit;
	0:	Disable INT external interrupt;
	1:	Enable INT external interrupt.
Bit3	PBIE:	PORTB level change interrupt enable bit(1);
	0:	Disable PORTB level change interrupt;
	1:	PORTB level change interrupt enabled.
Bit2	T0IF:	TMR0 overflow interrupt flag bit (2);
	0:	The TMR0 register has not overflowed;
	1:	The TMR0 register has overflowed (must be cleared by software).
Bit1	INTF:	INT external interrupt flag bit;
	0:	No INT external interrupt occurred;
	1:	An INT external interrupt occurred (must be cleared by software).
Bit0	PBIF:	PORTB level change interrupt flag bit;
	0:	None of the PORTB general purpose I/O pins have changed their state;
	1:	At least one pin in the PORTB port has changed state (must be cleared by software).

Note:

1. The IOCB register must also be enabled, and the corresponding port line must be set to the input state.
2. The T0IF bit is set to 1 when TMR0 rolls over to 0. A reset does not change TMR0 and should be initialized before clearing the T0IF bit.

7.3 Protection method of interrupt site

After an interrupt request occurs and is responded to, the program goes to 0004H to execute the interrupt subroutine. Before responding to the interrupt, the contents of ACC and STATUS must be saved. The chip does not provide dedicated push-to-stack save and pop-to-stack restore instructions. Users need to protect the contents of ACC and STATUS by themselves to avoid possible program running errors after the interruption ends.

Example: stack protection for ACC and STATUS

```
            ORG     0000H
            JP      START           ; User program start address
            ORG     0004H
            JP      INT_SERVICE     ; interrupt service routine
            ORG     0010H
START:
            ...
            ...
INT_SERVICE:
PUSH:                               ; Interrupt service routine entry, save ACC and STATUS
            LD      ACC             ; Save the value of ACC, (ACC_BAK needs to be customized)
            SWAPA   STATUS
            LD      STATUS          ; Save the value of STATUS, (STATUS_BAK needs to be customized)
            ...
            ...
POP:                                ; Interrupt service routine exit, restore ACC and STATUS
            SWAPA   STATUS_BAK
            LD      STATUS,A        ; Restore the value of STATUS
            SWAPR   ACC_BAK         ; Restore the value of ACC
            SWAPA   ACC_BAK
            RETI
```

7.3.1 Considerations for External Interrupts

Due to the fast response time of external interrupts, when the peripheral voltage of the system fluctuates, or the system is disturbed by EMC, the MCU may enter the interrupt by mistake, so an RC filter circuit needs to be added, as shown in the figure. The user can select different R1 and C1 according to the signal frequency sampled by the external interrupt to improve the EMC resistance of the system.

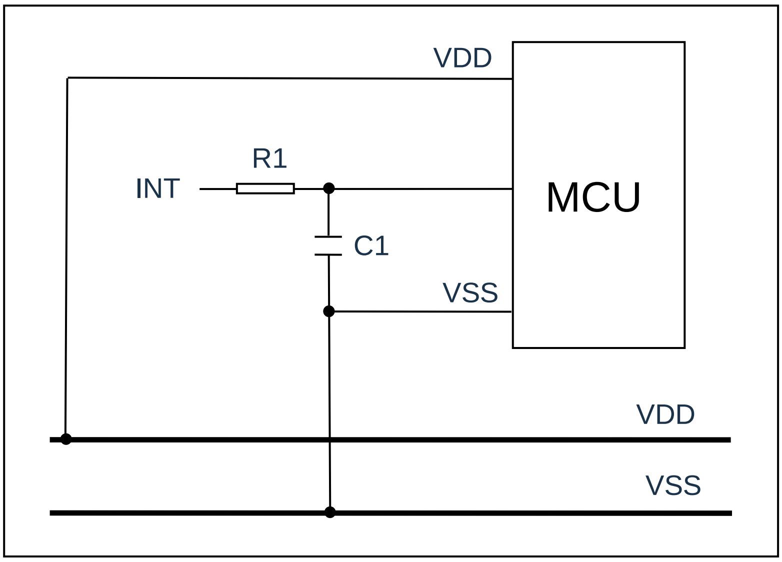


Figure 7-2: External interrupt RC filter circuit

7.4 Interrupt priority and interrupt nesting

At the same time, multiple interrupt requests may appear in the system. At this time, the user must set the priority of each interrupt according to the requirements of the system.

Note: When multiple interrupts occur at the same time, the MCU does not have a preset interrupt priority. First, the priority of each interrupt must be pre-set; secondly, use the interrupt enable bit and interrupt control bit to control whether the system responds to the interrupt. In the program, the interrupt control bit and the interrupt request flag must be checked.

8. Timer TMR0

8.1 TMR0 overview

TMR0 consists of the following functions:

- 8-bit timer/counter register (TMR0);
- 8-bit period register (TMR0PRD);
- 3-bit prescaler (shared with watchdog timer);
- Read and write operations can be performed with programs;
- Choices of working mode of timer or counter;
- Programmable internal or external clock source;
- External clock source can choose 32768 crystal oscillator;
- External clock edge selectable;
- Timer/counter overflow interrupt.

The working mode of TMR0 is selected by T0CS of the TMR0 control register (OPTION_REG):

- When T0CS=0, it works as a timer, adding 1 to each instruction cycle without prescaler. If TMR0 is written, the increment operation is prohibited for two cycles.
- When T0CS=1, XT_EN=1, it works as an external crystal oscillator timer, and the TMR0 module adds 1 to each oscillation cycle.
- When T0CS=1, XT_EN=0, it works as an external pulse counter, and the counter of TMR0 will count the pulses added to the T0CKI port. The rising edge or falling edge is selected by the bit T0SE. When T0SE=0, the rising edge is valid, and when T0SE=1, the falling edge is valid.

8.2 TMR0 related registers

There are three registers related to TMR0, 8-bit timer/counter (TMR0), 8-bit period register (TMR0PRD), and 8-bit programmable control register (OPTION_REG). TMR0 is an 8-bit readable and writable timer/counter, TMR0PRD is an 8-bit period compare register, OPTION_REG register, please refer to 2.6 about the application of prescaler register (OPTION_REG).

8-bit timer/counter TMR0(01H)

01H	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
TMR0								
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset value	X	X	X	X	X	X	X	X

8-bit period compare register TMR0PRD (0FH)

0FH	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
TMR0PRD								
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset value	X	X	X	X	X	X	X	X

8.3 Using an external clock as the clock source for TMR0

When TMR0 is used for external clock counting, the external clock input must meet certain conditions. The external clock is required to be synchronized with the internal phase clock (Tosc). After synchronization, TMR0 will be incremented after a certain delay.

If the prescaler is not used, the external clock is the input to TMR0, and the T0CKI can be synchronized with the internal phase clock by sampling the prescaler output during the Q2 and Q4 cycles of the internal clock. Therefore, the duration of high-level time of the T0CKI pin signal is required to be at least 2 Tosc (plus a short RC delay), and the low level time is at least 2 Tosc (plus a short RC delay).

If a prescaler is used, the external clock input must first be divided by the asynchronous pulse counting type prescaler, so that the output of the prescaler is symmetrical. In order for the external clock to meet the sampling requirements, the effect of the ripple counter must be considered. Therefore, the clock period of T0CKI is at least 4 Tosc (plus a small RC delay) divided by the prescale value.

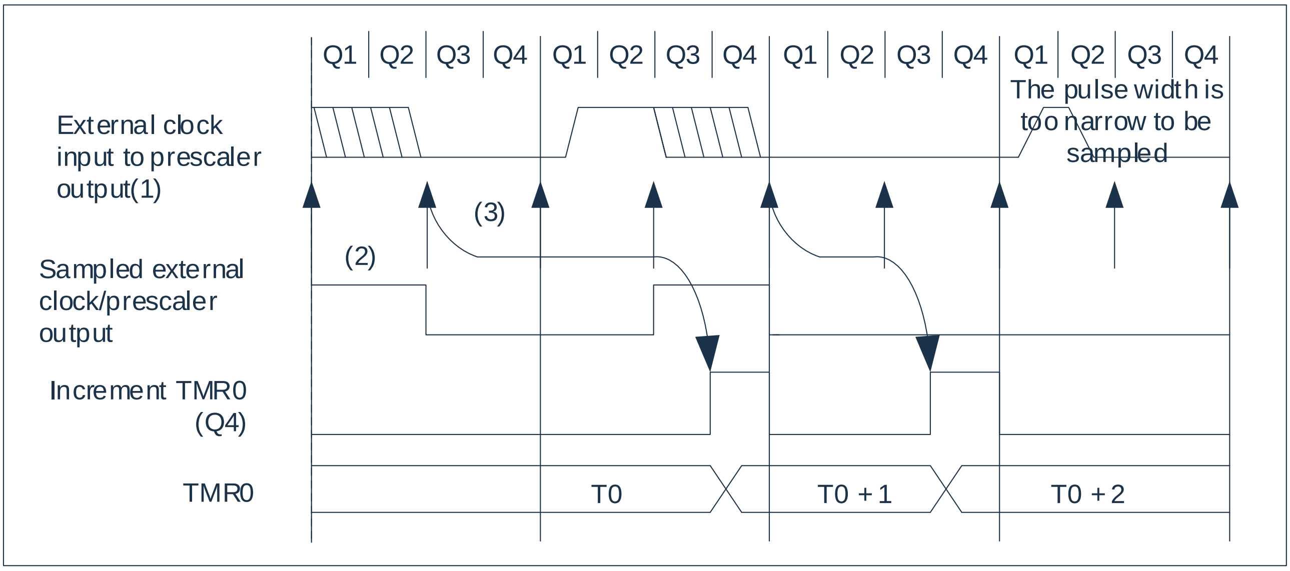


Figure 8-1: TMR0 and external clock timing

Note:

1. It is external clock when no prescaler is selected; otherwise it is the output of the prescaler.
2. The arrow points to the sampling time.
3. There will be a delay of 3 Tosc to 7 Tosc (duration Q=Tosc) when the clock input changes to the increment of TMR0, so when measuring the interval between adjacent pulses of TMR0 input, the maximum error is ±4Tosc.

8.4 Application of TMR0 as Timer

8.4.1 Basic time constant of TMR0

When the Bit3 bit of OPTION_REG is set to 1, the prescaler is used as the frequency division of the WDT timing. At this time, the input clock of TMR0 is the system clock divided by 2. When Bit3 of OPTION_REG is set to 0, the prescaler is used as the frequency division of the TMR0 counter. Its basic time constant is as follows:

OPTION_REG PS2～PS0	Input clock of TMR0 T0CLK	Fcpu=4MHz÷4	
		Maximum overflow interval	TMR0 increment time
000	Fcpu/2	512μs	2μs
001	Fcpu/4	1024μs	4μs
010	Fcpu/8	2048μs	8μs
011	Fcpu/16	4096μs	16μs
100	Fcpu/32	8192μs	32μs
101	Fcpu/64	16384μs	64μs
110	Fcpu/128	32768μs	128μs
111	Fcpu/256	65536μs	256μs

8.5 TMR0 operation procedure

The operation procedure of TMR0 is:

- Set the period of TMR0, TMR0PRD register (TMR0PRD will be latched after TMR0EN is enabled, so TMR0PRD must be written before TMR0 is enabled);
- Set the initial value of the TMR counter (this register is incremented from 0 by default, if the initial value is assigned and incremented by 1, the value of the counter will be automatically cleared after overflow, so the initial value needs to be assigned again);
- If T0CS=1, the system clock can be configured through IRCF[2:0] to change the period of TMR0;
- If PSA=1, the prescaler is used by TMR0, PS2: PS0 is assigned, and the prescaler ratio is selected;
- Enable TMR0, TMR0EN=1.

Example: TMR0 timing setting program

```
        LDIA        03FH
        LD          TMR0PRD,A           ;  set period register
        LDIA        000H
        LD          OPTION_REG,A        ;  Set TMR0 clock=Fcpu/2
        CLR         TMR0                ;  Initialize TMR0
        SETB        OSCCON,TMR0EN
```

Note:

1. The initial value of TMR0 will not be automatically loaded every time TMR0 overflows, so the user needs to reload the initial value of TMR0 every time TMR0 overflows; due to the write operation to TMR0, TMR0 will have a clock that does not increment, and the user to avoid this problem, enter the correction value yourself.
2. Continue to compare the TMR0 and TMR0PRD values to determine when they match. TMR0 will increment from 00h until it matches the value in TMR0PRD. When a match occurs, the following two events occur:
    1) TMR0 on the next increment cycle.
    2) TMR0 postscaler increments.

9. Genaral PWM

9.1 General PWM overview

General PWM consists of the following functions:

- 3 channels of general PWM share a 10-bit period register;
- 3 common PWMs share a prescaler;
- 3 channels of general PWM each have a 10-bit duty cycle register;

The normal PWM enable of SC8P115xA is controlled by the ENx (x=0-2) bits in PWMCTR0. After changing the duty cycle register of PWM, the duty cycle will be changed in the next cycle.

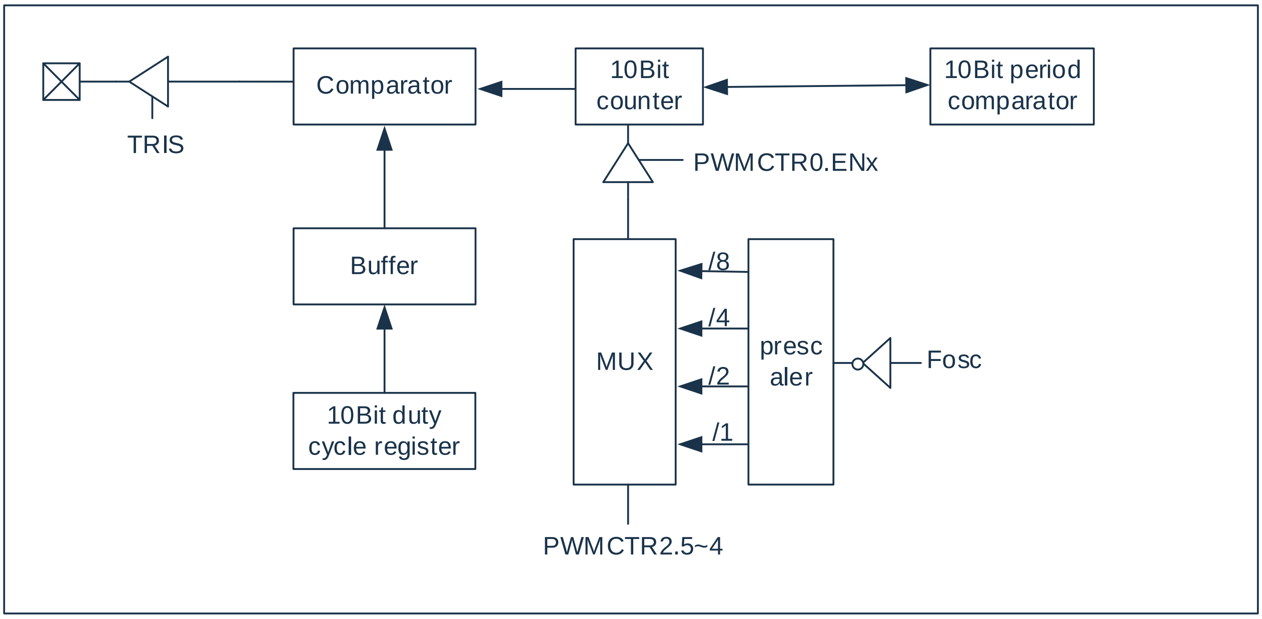


Figure 9-1: General PWM Block Diagram

9.2 Registers related to general PWM

There are 3 registers related to general PWM, PWMCTR0-2 (PWM control register), PWMPRD (PWM period register) and PWMR (PWM duty cycle register). Where PWMR is an indirect access register.

PWM control register 0PWMCTR0(10H)

10H	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
PWMCTR0	---	---	---	---	---	PWMEN2	PWMEN1	PWMEN0
R/W	---	---	---	---	---	R/W	R/W	R/W
Reset value	---	---	---	---	---	0	0	0

Bit7-Bit5 Not used.
Bit4-Bit1 PWMENx: PWM enable bit (x=0-2);
0: Disable PWMx function;
1: Enable PWMx function.

PWM Control Register 1 PWMCTR1(11H)

11H	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
PWMCTR1	---	---	PWMR29	PWMR28	PWMR19	PWMR18	PWMR09	PWMR08
R/W	---	---	R/W	R/W	R/W	R/W	R/W	R/W
Reset value	---	---	0	0	0	0	0	0

Bit7-Bit6 Not used.
Bit5-Bit0 PWMRx9-8: The upper 2 bits of the PWMx duty cycle register (bits 9 and 8, x=0-2).

PWM Control Register 2 PWMCTR2(12H)

12H	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
PWMCTR2	PWMPRD9	PWMPRD8	PWMCK1	PWMCK0	---	PWMS2	PWMS1	PWMS0
R/W	R/W	R/W	R/W	R/W	---	R/W	R/W	R/W
Reset value	0	0	0	0	---	0	0	0

Bit7-Bit6 PWMPRD9-8: The upper 2 bits of the PWM period register (bits 9 and 8, x=0-2);
Bit5-Bit4 PWMCK1-0: PWM clock divider selection bits.
00: FOSC
01: FOSC/2
10: FOSC/4
11: FOSC/8
Bit3 Not used.
Bit2-Bit0 PWMS2-0: PWMR register address select bits.
000: Select the lower 8 bits of the PWM0 channel duty cycle register;
001: Select the lower 8 bits of the PWM1 channel duty cycle register;
010: Select the lower 8 bits of the PWM2 channel duty cycle register;
Others: Not used.

PWM duty cycle register lower 8 bits PWMR (13H)

13H	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
PWMR								
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset value	0	0	0	0	0	0	0	0

PWMR duty cycle register lower 8 bits (3-way shared, indirect access)

Before accessing the PWMR duty cycle register, please set the PWMS2-0 bit in PWMCTR2 to select the PWM channel to be accessed.

PWM period register lower 8 bits PWMPRD (14H)

14H	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
PWMPRD								
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset value	0	0	0	0	0	0	0	0

9.3 9.3 Period and duty cycle of general PWM

9.3.1 General PWM output period

The general PWM output cycle is determined by the system main frequency (Fosc), the PWM clock frequency division ratio, and the PWM cycle register. The calculation formula is as follows:

PWM modulation period = (PWMPRD+1) ×PWM Frequency division ratio÷F_{OSC}

PWMPRD is a 10Bit register, the upper two bits are Bit7-6 of the PWMCTR2 register, PWMPRD9-8. The lower 8 bits are the PWMPRD register.

9.3.2 General PWM duty cycle algorithm

The duty cycle of the general PWM output is related to the value of PWMRx. On the whole, its duty cycle is approximately equal to PWMRx÷(PWMPRD+1). The specific calculation method is that when the PWM counter is greater than the PWMRx register, the PWM output is 0; when the PWM counter is less than or equal to the PWMRx register, the PWM output is 1. PWM counter is cleared when equal to period register.

9.4 General PWM Applications

The operation flow required for the application setting of PWM is as follows:

- Set the PWM prescaler value, PWMCK1 and PWMCK0;
- Set the PWM period, PWM_PRD9-8, PWM_PRD[7:0] a total of 10Bit.
- Set PWMS2-0 in PWMCTR2 to select PWM channel;
- Set the PWMR duty cycle: PWMRx9-8 (x=2-0), PWMR[7:0] has a total of 10Bit, because PWMR is an indirect access register, so the PWM channel to be set must be set first;
- Turn on the corresponding PWM enable bit: PWMCTR0.2-0. After turning on any enable, the PWM cycle will be latched, so the PWM cycle must be set before any channel of PWM is enabled.

10.Electrical parameters

10.1 DC characteristic

(VDD=5V, TA=25℃, except PB3 port, unless otherwise specified)

Symbol	Parameter	Test condition		min	typical	max	unit
		VDD	Condition				
VDD	Operating voltage	-	Fsys=8M	2.0	-	5.5	V
I_{DD}	Operating current	5V	----	-	1.5	-	mA
		3V	----	-	1	-	mA
I_{STB}	Quiescent current (LVR、WDT Disable)	5V	----	0.1	1.0	2.0	uA
		3V	----	0.01	0.1	1.0	uA
	Quiescent current (LVR、WDT Enable)	5V	----	5.0	6.0	10.0	uA
		3V	----	3.0	4.0	6.0	uA
V_{IL}	Low-level input voltage (pull-up on)	5V	----	-	1.2	-	V
	Low-level input voltage (pull-up off)	5V		-	1.4	-	V
V_{IH}	High-level input voltage (pull-up on)	5V	----	-	2.0	-	V
	High-level input voltage (pull-up off)	5V			2.2		V
V_{OH}	High level output voltage	-	No load	0.9VDD	-	-	V
V_{OL}	Low level output voltage	-	No load	-	-	0.1VDD	V
R_{PH}	Pull-up resistor value	5V	0.7VDD	-	40	-	K
		3V	0.7VDD	-	60	-	K
R_{PL}	Pull-down resistor value	5V	0.3VDD	-	50	-	K
		3V	0.3VDD	-	85	-	K
I_{OL}	Output port sink current	5V	V_{OL}=0.3VDD	-	40	-	mA
		3V	V_{OL}=0.3VDD	-	20	-	mA
I_{OH}	output port pull current	5V	V_{OH}=0.7VDD	-	10	-	mA
		3V	V_{OH}=0.7VDD	-	6	-	mA

10.2 LVR Electrical Characteristics

(T_A= 25℃, unless otherwise indicated)

Symbol	Parameters	Test condition	min	typical	max	unit
V_{LVR1}	LVR set voltage 1	VDD=1.8~5.5V	1.7	1.8	1.9	V
V_{LVR2}	LVR set voltage 2	VDD=1.98~5.5V	2.7	2.8	2.9	V

10.3 AC Characteristics

（T_A=25℃，unless otherwise indicated）

<table>
<tr><th rowspan="2">Symbol</th><th rowspan="2">Parameters</th><th colspan="2">Test condition</th><th rowspan="2">min</th><th rowspan="2">typical</th><th rowspan="2">max</th><th rowspan="2">unit</th></tr>
<tr><th>VDD</th><th>condition</th></tr>
<tr><td>Fsys</td><td>working frequency (RC)</td><td>-</td><td>2.5V~5.5V</td><td>-</td><td>8</td><td>-</td><td>MHz</td></tr>
<tr><td rowspan="2">T_{WDT}</td><td rowspan="2">WDT reset time</td><td>5V</td><td>-</td><td>-</td><td>18</td><td>-</td><td>ms</td></tr>
<tr><td>3V</td><td>-</td><td>-</td><td>30</td><td>-</td><td>ms</td></tr>
<tr><td rowspan="2">F_{RC}</td><td rowspan="2">Internal vibration frequency stability</td><td colspan="2">VDD=4.0~5.5V　T_A= - 20~85℃</td><td>-2%</td><td>8</td><td>+2%</td><td>MHz</td></tr>
<tr><td colspan="2">VDD=2.5~5.5V</td><td>-4%</td><td>8</td><td>+4%</td><td>MHz</td></tr>
</table>

11.Instructions

11.1 List of instructions

mnemonic	operation	instruction cycle	symbol
Control			
NOP	no-op	1	None
STOP	enter sleep mode	1	TO,PD
CLRWDT	Clear the watchdog counter	1	TO,PD
Data transfer			
LD [R],A	Transfer ACC content to R	1	NONE
LD A,[R]	Transfer R content to ACC	1	Z
TESTZ [R]	Transfer data memory contents to data memory	1	Z
LDIA i	Send immediate i to ACC	1	NONE
logic operation			
CLRA	clear ACC	1	Z
SET [R]	Set data memory R	1	NONE
CLR [R]	Clear data memory R	1	Z
ORA [R]	R and contents of ACC do "OR" operation, and the result is stored in ACC	1	Z
ORR [R]	R and contents of ACC do "OR" operation, and the result is stored in R	1	Z
ANDA [R]	R and contents of ACC do "AND" operation, and the result is stored in ACC	1	Z
ANDR [R]	R and contents of ACC do "AND" operation, and the result is stored in R	1	Z
XORA [R]	R and contents of ACC do "XOR" operation, and the result is stored in ACC	1	Z
XORR [R]	R and contents of ACC do "XOR" operation, and the result is stored in R	1	Z
SWAPA [R]	The high and low byte swap of the contents of the R register, and the result is stored in the ACC	1	NONE
SWAPR [R]	The high and low byte swap of the contents of the R register, and the result is stored in the R	1	NONE
COMA [R]	The contents of the R register are inverted, and the result is stored in the ACC	1	Z
COMR [R]	The contents of the R register are inverted, and the result is stored in the R	1	Z
XORIA i	ACC and immediate number i do an "XOR" operation, and the result is stored in ACC	1	Z
ANDIA i	ACC and immediate number i do an "AND" operation, and the result is stored in ACC	1	Z
ORIA i	ACC and immediate number i do an "OR" operation, and the result is stored in ACC	1	Z
Shift operation			
RRCA [R]	Rotate data memory right by one bit with carry, and the result is stored in ACC	1	C
RRCR [R]	Rotate data memory right by one bit with carry, and the result is stored in R	1	C
RLCA [R]	Rotate the data memory by one bit to the left with a carry, and store the result in ACC	1	C
RLCR [R]	Rotate the data memory by one bit to the left with a carry, and store the result in R	1	C
RLA [R]	Rotate the data memory by one bit to the left without carry, and store the result in ACC	1	NONE
RLR [R]	Rotate the data memory by one bit to the left without carry, and store the result in R	1	NONE
RRA [R]	Rotate data memory right by one bit without carry, and the result is stored in ACC	1	NONE

mnemonic		operation	instruction cycle	symbol
RRR	[R]	Rotate data memory right by one bit without carry, and the result is stored in R	1	NONE
Increment decrement				
INCA	[R]	Increment data memory R and place result in ACC	1	Z
INCR	[R]	Increment data memory R and place result in R	1	Z
DECA	[R]	Decrement data memory R and place result in ACC	1	Z
DECR	[R]	Decrement data memory R, put result in R	1	Z
Bit manipulation				
CLRB	[R],b	Clear a bit in data memory R	1	NONE
SETB	[R],b	Set a location in data memory R to a	1	NONE
Math computation				
ADDA	[R]	ACC+[R]→ACC	1	C,DC,Z,OV
ADDR	[R]	ACC+[R]→R	1	C,DC,Z,OV
ADDCA	[R]	ACC+[R]+C→ACC	1	Z,C,DC,OV
ADDCR	[R]	ACC+[R]+C→R	1	Z,C,DC,OV
ADDIA	i	ACC+i→ACC	1	Z,C,DC,OV
SUBA	[R]	[R]-ACC→ACC	1	C,DC,Z,OV
SUBR	[R]	[R]-ACC→R	1	C,DC,Z,OV
SUBCA	[R]	[R]-ACC-C→ACC	1	Z,C,DC,OV
SUBCR	[R]	[R]-ACC-C→R	1	Z,C,DC,OV
SUBIA	i	i-ACC→ACC	1	Z,C,DC,OV
Unconditional transfer				
RET		Return from subroutine	2	NONE
RET	i	Return from subroutine and store immediate I in ACC	2	NONE
RETI		Return from interrupt	2	NONE
CALL	ADD	Subroutine call	2	NONE
JP	ADD	Unconditional jump	2	NONE
Conditional transfer				
SZB	[R],b	If the b bit of the data memory R is "0", skip the next instruction	1 or 2	NONE
SNZB	[R],b	If the b bit of the data memory R is "1", skip the next instruction	1 or 2	NONE
SZA	[R]	Data memory R is sent to ACC, if the content is "0", the next instruction will be skipped	1 or 2	NONE
SZR	[R]	The content of data memory R is "0", then skip the next instruction	1 or 2	NONE
SZINCA	[R]	Add "1" to data memory R, put the result into ACC, if the result is "0", skip the next instruction	1 or 2	NONE
SZINCR	[R]	Add "1" to data memory R, put the result into R, if the result is "0", skip the next instruction	1 or 2	NONE
SZDECA	[R]	Decrease "1" from data memory R, put the result into ACC, if the result is "0", skip the next instruction	1 or 2	NONE
SZDECR	[R]	Decrease "1" from data memory R, put the result into R, if the result is "0", skip the next instruction	1 or 2	NONE

11.2 Instruction Description

ADDA **[R]**

operation: Add R to ACC and put the result into ACC

period: 1

flags affected: C, DC, Z, OV

example:

```
LDIA    09H      ;Assign 09H to ACC
LD      R01,A    ; Assign the value of ACC (09H) to the custom register R01
LDIA    077H     ;Assign 77H to ACC
ADDA    R01      ;Execution result: ACC=09H + 77H =80H
```

ADDR **[R]**

operation: Add R to ACC and put the result in R

period: 1

flags affected: C, DC, Z, OV

example:

```
LDIA    09H      ;Assign 09H to ACC
LD      R01,A    ; Assign the value of ACC (09H) to the custom register R01
LDIA    077H     ;Assign 77H to ACC
ADDR    R01      ;Execution result: R01=09H + 77H =80H
```

ADDCA **[R]**

operation: Add R plus ACC plus C bit, the result is put into ACC

period: 1

flags affected: C, DC, Z, OV

example:

```
LDIA    09H      ;Assign 09H to ACC
LD      R01,A    ; Assign the value of ACC (09H) to the custom register R01
LDIA    077H     ;Assign 77H to ACC
ADDCA   R01      ;Execution result: ACC= 09H + 77H + C=80H (C=0)
                                    ACC= 09H + 77H + C=81H (C=1)
```

ADDCR **[R]**

operation: Add R plus ACC plus C bit, the result is put into R

period: 1

flags affected: C, DC, Z, OV

example:

```
LDIA    09H      ;Assign 09H to ACC
LD      R01,A    ; Assign the value of ACC (09H) to the custom register R01
LDIA    077H     ;Assign 77H to ACC
ADDCR   R01      ;Execution result: R01 = 09H + 77H + C=80H (C=0)
                                    R01 = 09H + 77H + C=81H (C=1)
```

ADDIA	**i**		
operation:	Add immediate i to ACC, and put the result into ACC		
period:	1		
flags affected:	C，DC，Z，OV		
example:			
	LDIA	09H	;Assign 09H to ACC
	ADDIA	077H	;Execution result: ACC = ACC(09H) + i(77H)=80H

ANDA	**[R]**		
operation:	Register R and ACC perform logical AND operation, and the result is placed in ACC		
period:	1		
flags	Z		
example:			
	LDIA	0FH	;Assign 0FH to ACC
	LD	R01,A	;Assign the value of ACC (0FH) to register R01
	LDIA	77H	;Assign 77H to ACC
	ANDA	R01	;Execution result: ACC=(0FH and 77H)=07H

ANDR	**[R]**		
operation:	Register R and ACC perform logical AND operation, and the result is placed in R		
period:	1		
flags affected:	Z		
example:			
	LDIA	0FH	;Assign 0FH to ACC
	LD	R01,A	;Assign the value of ACC (0FH) to register R01
	LDIA	77H	;Assign 77H to ACC
	ANDR	R01	;Execution result: R01=(0FH and 77H)=07H

ANDIA	**i**		
operation:	Perform a logical AND operation on the immediate i and ACC, and put the result into ACC		
period:	1		
flags affected:	Z		
example:			
	LDIA	0FH	;Assign 0FH to ACC
	ANDIA	77H	;Execution result: ACC =(0FH and 77H)=07H

CALL	**add**		
operation:	call subroutine		
period:	2		
flags affected:	No		
example:			
	CALL	LOOP	; Call the subroutine address whose name is defined as "LOOP"

CLRA

operation:	ACC is cleared		
period:	1		
flags affected:	Z		
example:			
	CLRA		; Execution result: ACC=0

CLR **[R]**

operation:	Register R is cleared		
period:	1		
flags affected:	Z		
example:			
	CLR	R01	;Execution result：R01=0

CLRB **[R],b**

operation:	Bit b of register R is cleared		
period:	1		
flags affected:	No		
example:			
	CLRB	R01,3	;Execution result：Bit 3 of R01 is zero

CLRWDT

operation:	Clear the watchdog counter		
period:	1		
flags affected:	TO，PD		
example:			
	CLRWDT		; Watchdog counter clear

COMA **[R]**

operation:	Invert register R and put the result into ACC		
period:	1		
flags affected:	Z		
example:			
	LDIA	0AH	;ACC assigns 0AH
	LD	R01,A	;Assign the value of ACC (0AH) to register R01
	COMA	R01	;Execution result: ACC=0F5H

COMR	**[R]**	
operation:	Invert the register R and put the result into R	
period:	1	
flags affected:	Z	
example:		

LDIA	0AH	;ACC assigns 0AH
LD	R01,A	;Assign the value of ACC (0AH) to register R01
COMR	R01	;Execution result: R01=0F5H

DECA	**[R]**	
operation:	Register R is decremented by 1, and the result is placed in ACC	
period:	1	
flags affected:	Z	
example:		

LDIA	0AH	;ACC assigns 0AH
LD	R01,A	;Assign the value of ACC (0AH) to register R01
DECA	R01	;Execution result: ACC=(0AH-1)=09H

DECR	**[R]**	
operation:	Register R is decremented by 1, and the result is placed in R	
period:	1	
flags affected:	Z	
example:		

LDIA	0AH	;ACC assigns 0AH
LD	R01,A	;Assign the value of ACC (0AH) to register R01
DECR	R01	;Execution result: R01=(0AH-1)=09H

INCA	**[R]**	
operation:	Register R is incremented by 1, and the result is placed in ACC	
period:	1	
flags affected:	Z	
example:		

LDIA	0AH	;ACC assigns 0AH
LD	R01,A	; Assign the value of ACC (0AH) to register R01
INCA	R01	;Execution result: ACC=(0AH+1)=0BH

INCR **[R]**

operation: Register R is incremented by 1, and the result is placed in R

period: 1

flags affected: Z

example:

LDIA	0AH	;ACC assigns 0AH
LD	R01,A	;Assign the value of ACC (0AH) to register R01
INCR	R01	;Execution result: R01=(0AH+1)=0BH

JP **add**

operation: Jump to add address

period: 2

flags affected: No

example:

JP	LOOP	; Jump to the subroutine address whose name is defined as "LOOP"

LD **A,[R]**

operation: assign the value of R to ACC

period: 1

flags affected: Z

example:

LD	A,R01	;Assign the value of register R0 to ACC
LD	R02,A	;Assign the value of ACC to register R02 to realize the movement of data from R01→R02

LD **[R],A**

operation: assign the value of ACC to R

period: 1

flags affected: No

example:

LDIA	09H	;Assign 09H to ACC
LD	R01,A	;Execution result: R01=09H

LDIA **i**

operation: Immediate number i is assigned to ACC

period: 1

flags affected: No

example:

LDIA	0AH	; ACC assigns 0AH

NOP

operation:	No operation		
period:	1		
flags affected:	No		
example:			
	NOP		
	NOP		

ORIA **i**

operation:	The immediate value is logically ORed with ACC, and the result is assigned to ACC		
period:	1		
flags affected:	Z		
example:			
	LDIA	0AH	;ACC assigns 0AH
	ORIA	030H	;Execution result: ACC =(0AH or 30H)=3AH

ORA **[R]**

operation:	Register R and ACC perform logical OR operation, and the result is placed in ACC		
period:	1		
flags affected:	Z		
example:			
	LDIA	0AH	;Assign 0AH to ACC
	LD	R01,A	;Assign ACC(0AH) to register R01
	LDIA	30H	;Assign 30H to ACC
	ORA	R01	;Execution result：ACC=(0AH or 30H)=3AH

ORR **[R]**

operation:	The register R is logically ORed with ACC, and the result is placed in R		
period:	1		
flags affected:	Z		
example:			
	LDIA	0AH	;Assign 0AH to ACC
	LD	R01,A	;Assign ACC(0AH) to register R01
	LDIA	30H	;Assign 30H to ACC
	ORR	R01	;Execution result：R01=(0AH or 30H)=3AH

RET

operation: return from subroutine

period: 2

flags affected: No

example:

```
        CALL    LOOP        ;Call subroutine LOOP
        NOP                 ;The statement will be executed after the RET instruction returns
        …                   ; other programs
LOOP:
        …                   ; subroutine
        RET                 ; subroutine return
```

RET i

operation: Return from subroutine with parameters, the parameters are put into ACC

period: 2

flags affected: No

example:

```
        CALL    LOOP        ;Call subroutine LOOP
        NOP                 ;The statement will be executed after the RET instruction returns
        …                   ;Other programs
LOOP:
        …                   ;Subroutine
        RET     35H         ;Subroutine return, ACC=35H
```

RETI

operation: return from interrupt

period: 2

flags affected: No

example:

```
INT_START                   ;Interrupt program entry
        …                   ;Interrupt handler
        RETI                ;Interrupt return
```

RLCA [R]

operation: Register R and C are rotated to the left by one bit, and the result is placed in ACC

period: 1

flags affected: C

example:

```
        LDIA    03H         ;ACC assigns 03H
        LD      R01,A       ;The ACC value is assigned to R01, R01=03H
        RLCA    R01         ;Execution result: ACC=06H(C=0);
                                               ACC=07H(C=1)
                                               C=0
```

RLCR **[R]**

operation: Rotate register R and C to the left by one bit, and put the result into R

period: 1

flags affected: C

example:

```
LDIA    03H     ;ACC assigns 03H
LD      R01,A   ;The ACC value is assigned to R01, R01=03H
RLCR    R01     ;Execution result: R01=06H(C=0);
                                   R01=07H(C=1);
                                   C=0
```

RLA **[R]**

operation: Register R without C is rotated left by one bit, and the result is placed in ACC

period: 1

flags affected: No

example:

```
LDIA    03H     ; ACC assignment 03H
LD      R01,A   ; ACC value is assigned to R01, R01=03H
RLA     R01     ;Execution result: ACC=06H
```

RLR **[R]**

operation: Register R without C is rotated left by one bit, and the result is placed in R

period: 1

flags affected: No

example:

```
LDIA    03H     ;ACC assigns 03H
LD      R01,A   ;The ACC value is assigned to R01, R01=03H
RLR     R01     ;Execution result: R01=06H
```

RRCA **[R]**

operation: Register R and C rotate right by one bit, and the result is placed in ACC

period: 1

flags affected: C

example:

```
LDIA    03H     ;ACC assigns 03H
LD      R01,A   ;The ACC value is assigned to R01, R01=03H
RRCA    R01     ;Execution result: ACC=01H(C=0);
                                   ACC=081H(C=1);
                                   C=1
```

RRCR **[R]**

operation: Register R and C rotate right by one bit, and the result is placed in R

period: 1

flags affected: C

example:

```
LDIA    03H     ;ACC assigns 03H
LD      R01,A   ;The ACC value is assigned to R01, R01=03H
RRCR    R01     ;Execution result: R01=01H(C=0);
                                   R01=81H(C=1);
                                   C=1
```

RRA **[R]**

operation: Register R without C is rotated right by one bit, and the result is placed in ACC

period: 1

flags affected: No

example:

```
LDIA    03H     ;ACC assigns 03H
LD      R01,A   ;The ACC value is assigned to R01, R01=03H
RRA     R01     ;Execution result: ACC=81H
```

RRR **[R]**

operation: Register R without C rotates right by one bit, the result is placed in R

period: 1

flags affected: No

example:

```
LDIA    03H     ;ACC assigns 03H
LD      R01,A   ;The ACC value is assigned to R01, R01=03H
RRR     R01     ;Execution result: R01=81H
```

SET **[R]**

operation: All bits of register R are set to 1

period: 1

flags affected: No

example:

```
SET     R01     ;Execution result: R01=0FFH
```

SETB **[R],b**

operation: Bit b of register R is set to 1

period: 1

flags affected: No

example:

```
CLR     R01     ;R01=0
SETB    R01,3   ;Execution result: R01=08H
```

STOP

operation: go to sleep

period: 1

flags affected: TO， PD

example:

STOP		; The chip enters the power saving mode, the CPU and oscillator stop working, and the IO port remains in the original state.

SUBIA **i**

operation: Subtract ACC from immediate i, and put the result into ACC

period: 1

flags affected: C,DC,Z,OV

example:

LDIA	077H	; ACC assigns 77H
SUBIA	80H	;Execution result：ACC=80H-77H=09H

SUBA **[R]**

operation: Register R minus ACC, the result is placed in ACC

period: 1

flags affected: C,DC,Z,OV

example:

LDIA	080H	;ACC assigns 80H
LD	R01,A	;The value of ACC is assigned to R01, R01=80H
LDIA	77H	;ACC assigns 77H
SUBA	R01	;Execution result：ACC=80H-77H=09H

SUBR **[R]**

operation: Register R minus ACC, the result is placed in R

period: 1

flags affected: C,DC,Z,OV

example:

LDIA	080H	;ACC assigns 80H
LD	R01,A	;The value of ACC is assigned to R01, R01=80H
LDIA	77H	;ACC assigns 77H
SUBR	R01	;Execution result：R01=80H-77H=09H

SUBCA **[R]**

operation: Register R minus ACC minus C, the result is put into ACC

period: 1

flags affected: C,DC,Z,OV

example:

```
LDIA    080H    ;ACC assigns 80H
LD      R01,A   ;The value of ACC is assigned to R01, R01=80H
LDIA    77H     ;ACC assigns 77H
SUBCA   R01     ;Execution result: ACC=80H-77H-C=09H(C=0);
                                   ACC=80H-77H-C=08H(C=1);
```

SUBCR **[R]**

operation: Register R minus ACC minus C, the result is put into R

period: 1

flags affected: C,DC,Z,OV

example:

```
LDIA    080H    ;ACC assigns 80H
LD      R01,A   ;The value of ACC is assigned to R01, R01=80H
LDIA    77H     ;ACC assigns 77H
SUBCR   R01     ;Execution result: R01=80H-77H-C=09H(C=0)
                                   R01=80H-77H-C=08H(C=1)
```

SWAPA **[R]**

operation: The high and low byte of register R are swapped, and the result is placed in ACC

period: 1

flags affected: No

example:

```
LDIA    035H    ;ACC assigns 35H
LD      R01,A   ;The value of ACC is assigned to R01, R01=35H
SWAPA   R01     ;Execution result: ACC=53H
```

SWAPR **[R]**

operation: The high and low byte of register R are swapped, and the result is placed in R

period: 1

flags affected: No

example:

```
LDIA    035H    ;ACC assigns 35H
LD      R01,A   ;The value of ACC is assigned to R01, R01=35H
SWAPR   R01     ;Execution result: R01=53H
```

SZB	**[R],b**		
operation:	Judging the bth bit of register R, it is a 0 jump, otherwise it is executed sequentially		
period:	1 or 2		
flags affected:	No		
example:			
	SZB	R01,3	; Judge the 3rd bit of register R01
	JP	LOOP	;The third bit of R01 is 1 to execute this statement and jump to LOOP
	JP	LOOP1	;The third bit of R01 is 0 time jump, execute this statement, jump to LOOP1

SNZB	**[R],b**		
operation:	Judging the bth bit of register R, it is a jump between 1, otherwise it is executed sequentially		
period:	1 or 2		
flags affected:	No		
example:			
	SNZB	R01,3	; Judge the 3rd bit of register R01
	JP	LOOP	;The third bit of R01 is 0 to execute this statement and jump to LOOP
	JP	LOOP1	;The third bit of R01 is 1 time jump, execute this statement, jump to LOOP1

SZA	**[R]**		
operation:	Assign the value of register R to ACC, skip if R is 0, otherwise execute sequentially		
period:	1 or 2		
flags affected:	No		
example:			
	SZA	R01	;R01→ACC
	JP	LOOP	;Execute this statement when R01 is not 0, jump to LOOP
	JP	LOOP1	;R01 is 0 time jump, execute this statement, jump to LOOP1

SZR	**[R]**		
operation:	Assign the value of register R to R, skip if R is 0, otherwise execute sequentially		
period:	1 or 2		
flags affected:	No		
example:			
	SZR	R01	;R01→R01
	JP	LOOP	;Execute this statement when R01 is not 0, jump to LOOP
	JP	LOOP1	;R01 is 0 time jump to execute this statement, jump to LOOP1

SZINCA **[R]**

operation: Add 1 to register R and put the result into ACC. If the result is 0, skip the next statement, otherwise execute sequentially

period: 1 or 2

flags affected: No

example:

```
SZINCA    R01      ;R01+1→ACC
JP        LOOP     ;Execute this statement when ACC is not 0, jump to LOOP
JP        LOOP1    ;Execute this statement when ACC is 0, jump to LOOP1
```

SZINCR **[R]**

operation: Add 1 to register R, and put the result into R. If the result is 0, skip the next statement, otherwise execute sequentially

period: 1 or 2

flags affected: No

example:

```
SZINCR    R01      ;R01+1→R01
JP        LOOP     ; Execute this statement when R01 is not 0, jump to LOOP
JP        LOOP1    ; Execute this statement when R01 is 0, jump to LOOP1
```

SZDECA **[R]**

operation: Decrement register R by 1, and put the result into ACC. If the result is 0, skip the next statement, otherwise execute sequentially

period: 1 or 2

flags affected: No

example:

```
SZDECA    R01      ;R01-1→ACC
JP        LOOP     ;Execute this statement when ACC is not 0, jump to LOOP
JP        LOOP1    ;Execute this statement when ACC is 0, jump to LOOP1
```

SZDECR **[R]**

operation: Decrement register R by 1, put the result into R, if the result is 0, skip the next statement, otherwise execute sequentially

period: 1 or 2

flags affected: No

example:

```
SZDECR    R01      ;R01-1→R01
JP        LOOP     ; Execute this statement when R01 is not 0, jump to LOOP
JP        LOOP1    ; Execute this statement when R01 is 0, jump to LOOP1
```

TESTZ	**[R]**	
operation:	Assign the value of R to R to affect the Z flag	
period:	1	
flags affected:	Z	
example:		

```
TESTZ    R0          ; Assign the value of register R0 to R0 to affect the Z flag bit
SZB      STATUS,Z    ; Judging the Z flag bit, it is a jump between 0
JP       Add1        ; Jump to address Add1 when register R0 is 0
JP       Add2        ; Jump to address Add1 when register R0 is not 0
```

XORIA	**i**	
operation:	The immediate value is XORed with ACC, and the result is put into ACC	
period:	1	
flags affected:	Z	
example:		

```
LDIA     0AH         ; ACC assigns 0AH
XORIA    0FH         ;Execution result：ACC=05H
```

XORA	**[R]**	
operation:	Register R and ACC perform logical XOR operation, and the result is placed in ACC	
period:	1	
flags affected:	Z	
example:		

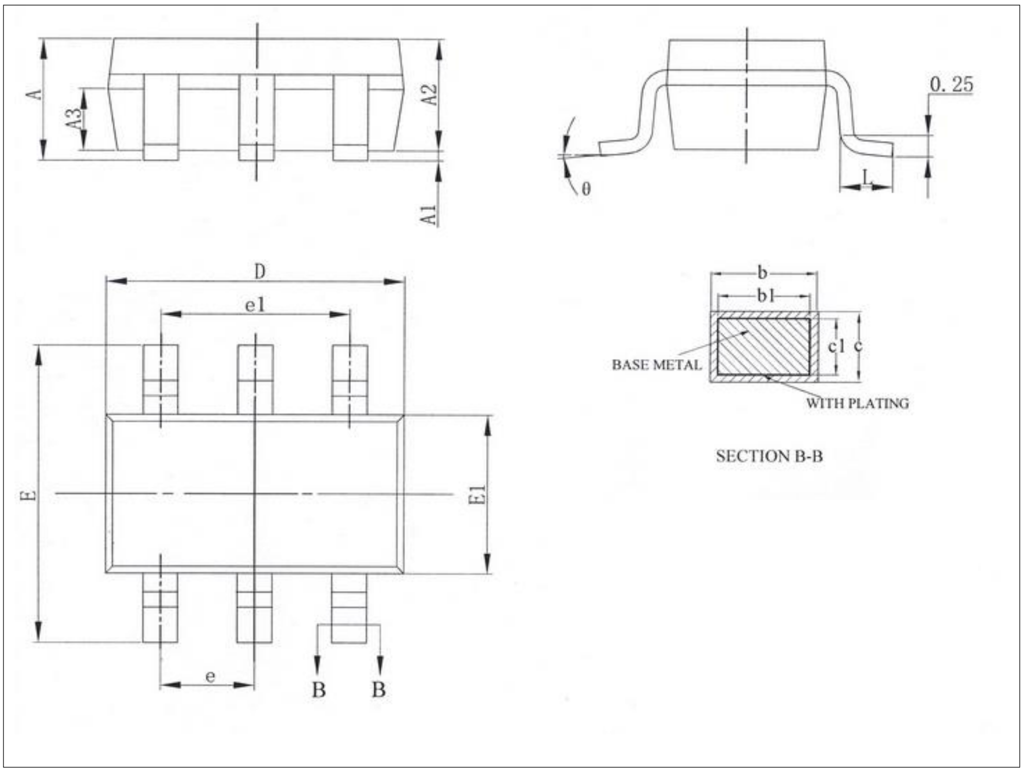
```
LDIA     0AH         ;ACC assigns 0AH
LD       R01,A       ;The ACC value is assigned to R01, R01=0AH
LDIA     0FH         ;ACC assigns 0FH
XORA     R01         ;Execution result：ACC=05H
```

XORR	**[R]**	
operation:	Register R and ACC perform a logical XOR operation, and the result is placed in R	
period:	1	
flags affected:	Z	
example:		

```
LDIA     0AH         ;ACC assigns 0AH
LD       R01,A       ;The ACC value is assigned to R01, R01=0AH
LDIA     0FH         ;ACC assigns 0FH
XORR     R01         ;Execution result：R01=05H
```

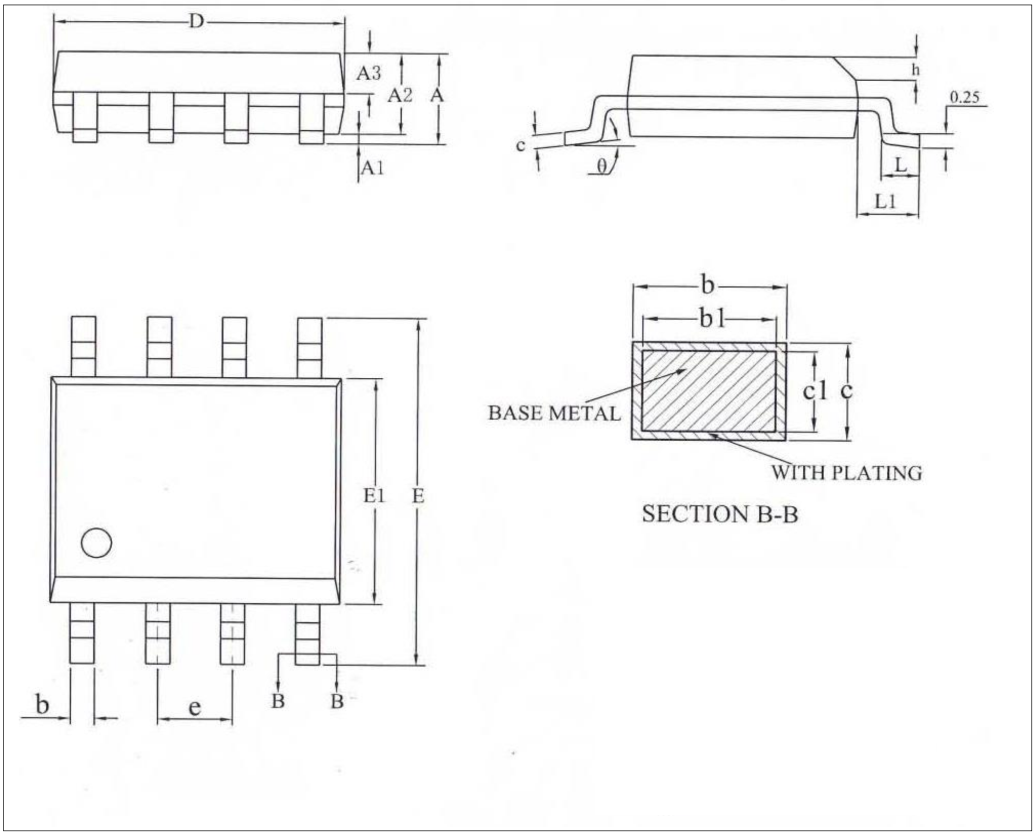
12.Packaging

12.1 SOT23-6



Symbol	Millimeter		
	Min	Nom	Max
A	-	-	1.25
A1	0.04	-	0.10
A2	1.00	1.10	1.20
A3	0.55	0.65	0.75
b	0.38	-	0.48
b1	0.37	0.40	0.43
c	0.11	-	0.21
c1	0.10	0.13	0.16
D	2.72	2.92	3.12
E	2.60	2.80	3.00
E1	1.40	1.60	1.80
e	0.95BSC		
e1	1.9BSC		
L	0.30	-	0.60
θ	0	-	8°

12.2 SOP8



Symbol	Millimeter		
	Min	Nom	Max
A	-	-	1.75
A1	0.10	-	0.225
A2	1.30	1.40	1.50
A3	0.60	0.65	0.70
b	0.39	-	0.47
b1	0.38	0.41	0.44
c	0.20	-	0.24
c1	0.19	0.20	0.21
D	4.80	4.90	5.00
E	5.80	6.00	6.20
E1	3.80	3.90	4.00
e	1.27BSC		
h	0.25	-	0.50
L	0.5	-	0.80
L1	1.05REF		
θ	0	-	8°

13. Version revision instructions

Version	Time	Content modified
V1.0	July 2018	initial version
V1.1	July 2019	Feature version upgrade
V1.2	May 2020	Change to new format
V1.3	Jan 2022	Correct packaging information
V1.3.1	Mar	1) Add DAT2 and PB0 lines to the figure in chapter 1.5 2) Correct the Bit7 name in 3.3 Oscillator Control Register