SN8P2714X_2715

USER'S MANUAL

Preliminary V0.2

SN8P2714
SN8P27142
SN8P27143
SN8P2715

SONiX 8-Bit Micro-Controller

AMENDMENT HISTORY

Version	Date	Description
VER 0.1	Aug. 2004	Preliminary Version first issue
VER 0.2	Jan. 2005	1. Add SN8P27142/ SN8P27143 relative data. 2. Fix ADC clock and Timer clock description. 3. Add LVD36 relative information. 4. Correct the LVD24 bit location from bit 3 to bit 4 in PFLAG register description. 5. Modify LVD code option related description 6. Modify TC0RATE and TC1RATE table. 7. Add TC0X8 and TC1X8 notice. 8. Release the ROM address 0x04 ~ 0x07 as general-purpose area. 9. Remove the instruction limitation at interrupt vector address (0x08) 10. Change IDE support version to M2IDE V1.04 11. Modify pin circuit diagram. 12. There is no Schmitt trigger input in port 4. 13. Add description of P0.3 without wakeup function

Table of Contents

1. PRODUCT OVERVIEW

FEATURES of SN8P2710 Series

- **Memory configuration**
  OTP ROM size: 2K * 16 bits.
  RAM size: 128 * 8 bits
  Eight levels stack buffer

- **I/O pin configuration**
  Input only pin: P0
  Bi-directional: P2, P4, P5
  Wakeup: P0.0, P0.1, P0.2
  External interrupt: P0.0, P0.1
  Pull-up resisters: P0, P2, P4, P5
  P4 pins shared with ADC inputs.
- **Max 8-channel 12-bit ADC.**

- **One channel 7-bit DAC.**

- **Powerful instructions**
  One clocks per instruction cycle (1T)
  Most of instructions are one cycle only
  All ROM area lookup table function (MOVC)

- **Four interrupt sources**
  Two internal interrupts: TC0, TC1
  Two external interrupts: INT0, INT1

- **Two 8-bit Timer/Counter**
  TC0: Auto-reload timer/Counter/PWM0/Buzzer output
  TC1: Auto-reload timer/Counter/PWM1/Buzzer output
- **On chip watchdog timer.**

- **System clocks and Operating modes**
  External high clock: RC type up to 10 MHz
  External high clock: Crystal type up to 16 MHz
  Internal low clock: RC type 16KHz(3V), 32KHz(5V)
  Normal mode: Both high and low clock active
  Slow mode: Low clock only
  Sleep mode: Both high and low clock stop

- **Package (Chip form support)**
  SN8P27142: P-DIP 18 pins, SOP 18pins
  SN8P27143:P-DIP 20 pins, SOP 20 pins, SSOP 20 pins
  SN8P2714: SK-DIP 28 pins, SOP 28pins
  SN8P2715: P-DIP 32 pins, SOP 32 pins

FEATURES SELECTION TABLE

CHIP	ROM	RAM	Stack	Timer			I/O	ADC	DAC	PWM	SIO	Wakeup	Package
				T0	TC0	TC1				Buzzer		Pin no.	
SN8P27142	2K*16	128	8	-	V	V	15	5ch	-	2	-	2	DIP18/SOP18
SN8P27143	2K*16	128	8	-	V	V	16	6ch	-	2	-	2	DIP20/SOP20/SSOP20
SN8P2714	2K*16	128	8	-	V	V	23	8ch	1ch	2	-	3	SKDIP28/SOP28
SN8P2715	2K*16	128	8	-	V	V	27	8ch	1ch	2	-	3	DIP32/SOP32
SN8P2704A	4K*16	256	8	V	V	V	18	5ch	1ch	2	1	8	SKDIP28/SOP28
SN8P2705A	4K*16	256	8	V	V	V	23	8ch	1ch	2	1	9	DIP32/SOP32

Note: For SN8P27143 and SN8P27142 must configure P02R (bit 2 of P0UR) as "1" to avoid sleep mode fail.

Compare SN8P2714/15 to SN8P2704A/05A

Item	SN8P2714/15	SN8P2704A/05A
AC Noise Immunity Capability	Excellent **(Add an 47uF bypass Capacitor between VDD and GND)**	Excellent **(Add an 47uF bypass Capacitor between VDD and GND)**
Memory	ROM: 2K x 16 RAM: 128 x 8	ROM: 4K x 16 RAM: 256 x 8
Maximum I/O pins	23 I/O in 28 pins package 27 I/O in 32 pins package	18 I/O in 28 pins package 23 I/O in 32 pins package
High Speed PWM	PWM Resolution: 8bit/6bit/5bit/4bit 8bit PWM up to 62.5K at 16Mhz 4bit PWM up to 1000K at 16Mhz	PWM Resolution: 8bit/6bit/5bit/4bit 8bit PWM up to 7.8125K at 16Mhz 4bit PWM up to 125K at 16Mhz
Programmable Open-Drain Output	N/A	P1.0 / P1.1 / P5.2 (SO)
B0MOV M, I	No Limitation	“I” can’t be 0E6h or 0E7h
B0XCH A, M	No Limitation	The address of M can’t be 80h ~ FFh
Valid instruction in ROM address 8	No Limitation	JMP or NOP
ADC Interrupt	No	Yes
ADC Clock Frequency	Four kinds of setting (Configuration by ADCKS [1:0])	Seven kinds of setting (Configuration by ADCKS [2:0])
Valid Range of TC0C/TC1C/TC0R/TC1R	0x00 ~ 0xFF	0x00 ~ 0xFE
Green mode	No	Yes
SIO Function	No	Yes
LVD	Level: 2.0V always ON	1.8V always ON
Port 0	Input only port	Bi-direction port
Interrupt Vector Instruction	No Limitation	NOP/JMP only
PUSH/POP Instruction	No	Yes

SYSTEM BLOCK DIAGRAM

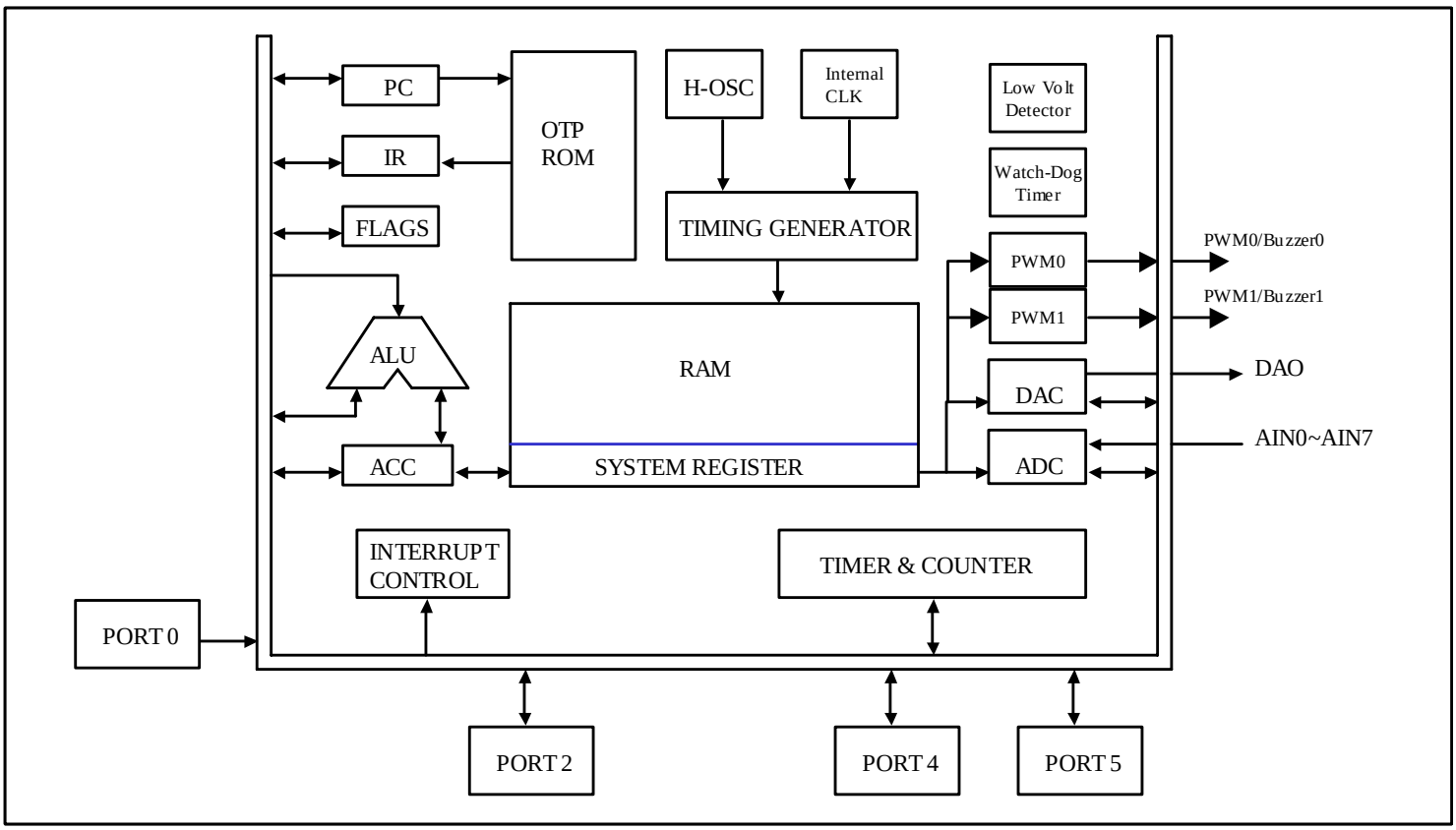
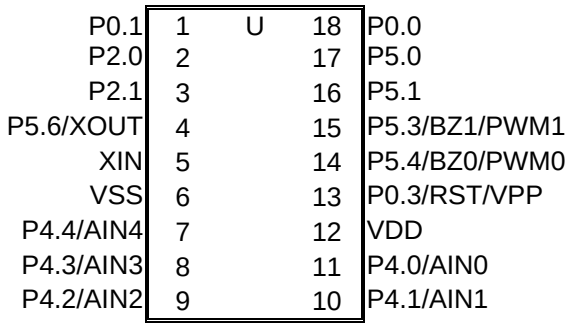


Figure 1-1.Simplified System Block Diagram

PIN ASSIGNMENT

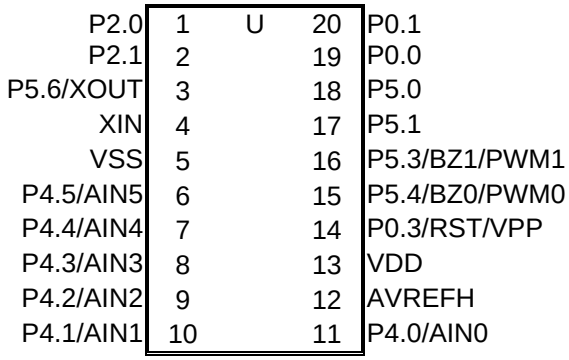
Format Description：SN8P271xY　　Y = K > SK-DIP，P > P-DIP，　S> SOP

SN8P27142 Pin Assignment



SN8P27142P
SN8P27142S

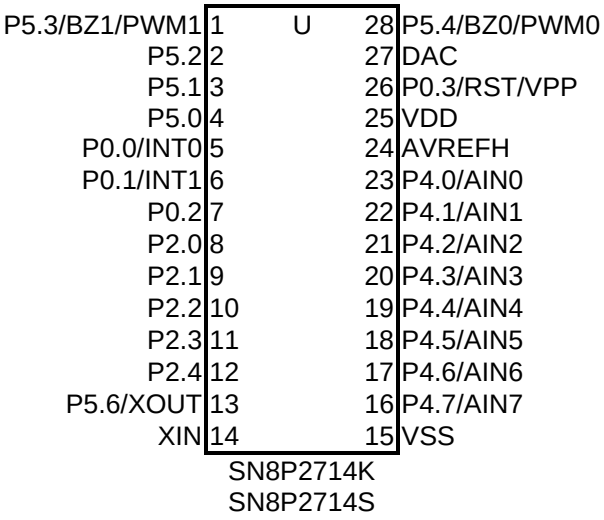
SN8P27143 Pin Assignment



SN8P27143P
SN8P27143S
SN8P27143X

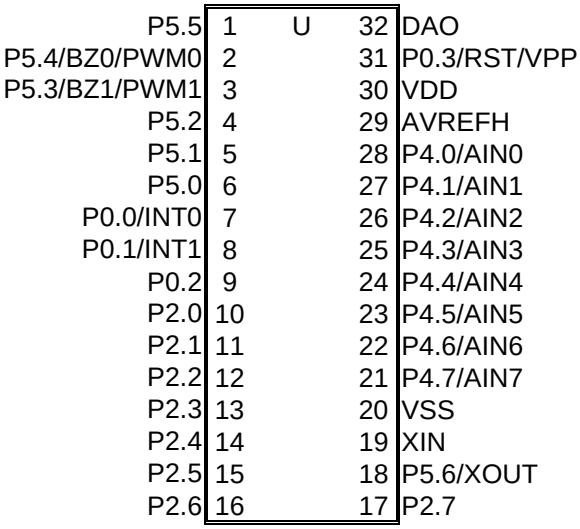
- ***Note: For SN8P27143 and SN8P27142 must configure P02R (bit 2 of P0UR) as "1" to avoid sleep mode fail.***

SN8P2714K Pin Assignment



SN8P2714K
SN8P2714S

SN8P2715P Pin Assignment



SN8P2715P
SN8P2715S

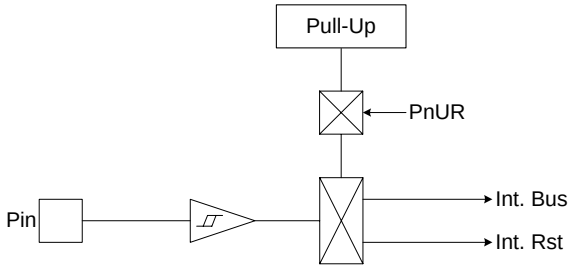
PIN DESCRIPTIONS

PIN NAME	TYPE	DESCRIPTION
P0 [1:0] / INT [1:0]	I	P0 [1:0]: Input only pin/ wakeup/pull-up/Schmitt trigger input INT [1:0]: External interrupt
P0 .2	I	P0.2: Input only pin/wake up/pull-up/Schmitt trigger input
P2 [7:0]	I/O	P2 [7:0] bi-direction pins/pull-up/Schmitt trigger input
P4 [7:0] / AIN [7:0]	I/O	Bi-direction pins/pull-up /ADC input/ **without** Schmitt trigger input
P5 [5:0]	I/O	Bi-direction pins/pull-up/Schmitt trigger input P5.4: PWM0/BZ0, P5.3: PWM1/BZ1
AVREFH	I	ADC highest reference voltage input
DAO	O	Current type DAC output
P0.3/RST/VPP	I/P	P0.3: Schmitt trigger input pin / **NO pull-up, NO wakeup**/ in RC mode RST: External reset, active “low” VPP: OTP programming pin
XIN	I	External oscillator input pin. / External RC oscillator input
XOUT/P5.6	I/O	XOUT: External oscillator output pin. P5.6: Bi-direction pin/pull-up/Schmitt trigger input in RC mode
VDD, VSS	P	Power supply pins.

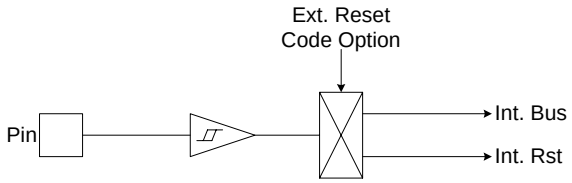
Table 1-1. SN8P2714/15 Pin Description

PIN CIRCUIT DIAGRAMS

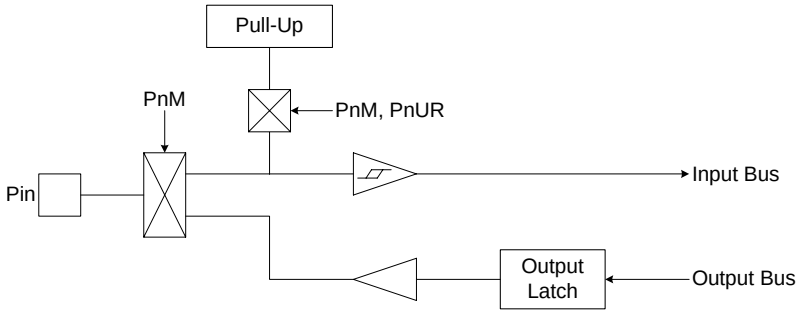
Port 0.1 and P0.2 structure:



Port 0.3 structure:



Port 2, 5 structure:



Port 4 structure:

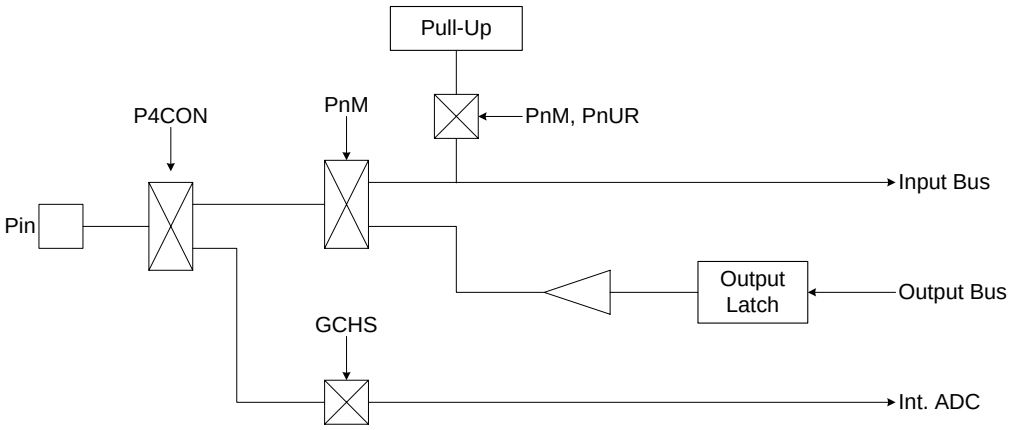


Figure 1-2. Pin Circuit Diagram

2. CODE OPTION TABLE

Code Option	Content	Function Description
High_Clk	Ext_RC	Low cost external RC oscillator for high clock oscillator. Output the Fcpu clock from Xout pin.
	32K_X'tal	Low frequency, power saving crystal (e.g. 32.768KHz) for high clock oscillator.
	12M_X'tal	High speed crystal /resonator (e.g. 12MHz ~ 16MHz) for high clock oscillator.
	4M_X'tal	Middle speed crystal /resonator (e.g. 4MHz ~ 10Mhz) for high clock oscillator.
Noise_Filter	Enable	Enable noise filter to enhance noise immunity performance.
	Disable	Disable noise filter.
Watch_Dog	Always_On	Watchdog timer always on even in sleep (power down) mode.
	Enable	Normal mode: Enable Watchdog timer Sleep mode: Stop Watchdog timer Stop
	Disable	Disable Watchdog function.
Fcpu	Fosc/1	Instruction cycle is oscillator clock.
	Fosc/2	Instruction cycle is 2 oscillator clocks.
	Fosc/4	Instruction cycle is 4 oscillator clocks.
	Fosc/8	Instruction cycle is 8 oscillator clocks.
Security	Enable	Enable ROM code Security function.
	Disable	Disable ROM code Security function.
RST_P0.3	Reset	Enable external reset pin
	P0.3	Enable P0.3 input only pin without pull-up register
LVD	LVD_L	LVD will reset chip if VDD is below 2.0V
	LVD_M	LVD will reset chip if VDD is below 2.0V Enable LVD24 bit of PFLAG register for 2.4V low voltage indicator.
	LVD_H	LVD will reset chip if VDD is below 2.4V Enable LVD36 bit of PFLAG register for 3.6V low voltage indicator.

Table 2-1. Code Option Table of SN8P270xA

Notice:

- ***In high noisy environment, enable "Noise Filter" and set Watch_Dog as "Always_On" is strongly recommended.***
- ***Enable "Noise Filter" will limit the Fcpu = Fosc/4 or Fosc/8***
- ***Fcpu code option is only available for High Clock***
- ***Fosc = Fhosc (External high clock) in normal mode.***
- ***Fosc = Flosc (Internal low RC clock) in slow mode.***
- ***In slow mode, Fcpu = Fosc / 4.***

3. ADDRESS SPACES

PROGRAM MEMORY (ROM)

OVERVIEW

ROM Maps for SN8P2710 devices provide 2K X 16-bit OTP programmable memory. The SN8P2710 program memory is able to fetch instructions through 12-bit wide PC (Program Counter) and can look up ROM data by using ROM code registers (R, X, Y, Z). In standard configuration, the device's 2,048 x 16-bit program memory has four areas:

- 1-word reset vector addresses
- 1-word Interrupt vector addresses
- 4-words reserved area
- 2K words

All of the program memory is partitioned into three coding areas. The 1st area is located from 00H to 07H(The Reset vector area), the 2nd area is for the interrupt vector (0008H) and the 3ed area is user code area from 0009H to 07FBH.

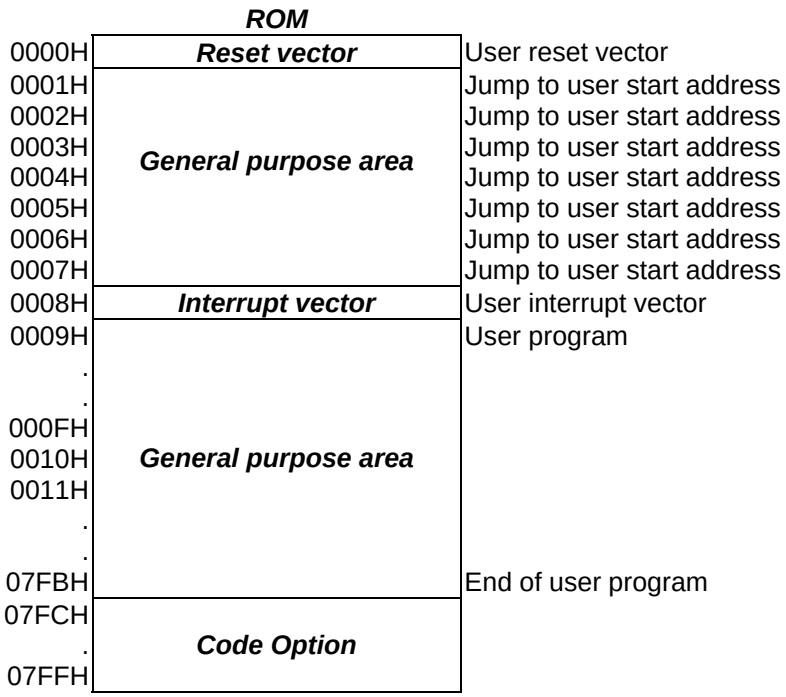


Figure 3-1 ROM Address Structure

USER RESET VECTOR ADDRESS (0000H)

A 1-word vector address area is used to execute system reset. After power on reset or watchdog timer overflow reset, then the chip will restart the program from address 0000h and all system registers will be set as default values. The following example shows the way to define the reset vector in the program memory.

➲ **Example: After power on reset, external reset active or reset by watchdog timer overflow.**

```
                ORG     0               ; 0000H
                JMP     START           ; Jump to user program address.
                .                       ;

                ORG     10H
START:                                  ; 0010H, The head of user program.
                .                       ; User program
                .
                .
                .
                ENDP                    ; End of program
```

INTERRUPT VECTOR ADDRESS (0008H)

A 1-word vector address area is used to execute interrupt request. If any interrupt service is executed, the program counter (PC) value is stored in stack buffer and points to 0008h of program memory to execute the vectored interrupt. Users have to define the interrupt vector and the following example shows the way to define the interrupt vector in the program memory.

➲ **Example 1: This demo program includes interrupt service routine and the user program is behind the interrupt service routine.**

```
                ORG     0               ; 0000H
                JMP     START           ; Jump to user program address.
                .


                ORG     8               ; Interrupt service routine
                B0XCH   A, ACCBUF       ; B0XCH doesn't change C, Z flag
                B0MOV   A, PFLAG
                B0MOV   PFLAGBUF, A     ; Save PFLAG register in a buffer
                .                       ; User code
                .                       ; User code
                B0MOV   A, PFLAGBUF
                B0MOV   PFLAG, A        ; Restore PFLAG register from buffer
                B0XCH   A, ACCBUF
                RETI                    ; End of interrupt service routine

START:                                  ; The head of user program.
                .                       ; User program
                .
                .
                JMP     START           ; End of user program

                ENDP                    ; End of program
```

➲ **Example 2: The demo program includes interrupt service routine and the address of interrupt service routine is in a special address of general-purpose area.**

```
                ORG     0           ; 0000H
                JMP     START       ; Jump to user program address.
                .                   ; 0001H ~ 0007H are reserved

                ORG     08
                JMP     MY_IRQ      ; 0008H, Jump to interrupt service routine address

                ORG     10H
START:                              ; 0010H, The head of user program.
                .                   ; User program
                .
                .
                .
                JMP     START       ; End of user program

MY_IRQ:                             ;The head of interrupt service routine
                B0XCH   A, ACCBUF   ; B0XCH doesn't change C, Z flag
                B0MOV   A, PFLAG
                B0MOV   PFLAGBUF, A ; Save PFLAG register in a buffer
                .                   ; User code
                .                   ; User code
                B0MOV   A, PFLAGBUF
                B0MOV   PFLAG, A    ; Restore PFLAG register from buffer
                RETI                ; End of interrupt service routine

                ENDP                ; End of program
```

➢ ***Note: It is easy to get the rules of SONIX program from demo programs given above. These points are as following.***

1.The address 0000H is a "JMP" instruction to make the program go to general-purpose ROM area.

2. The interrupt service starts from 0008H. Users can put the whole interrupt service routine from 0008H (Example1) or to put a "JMP" instruction in 0008H then place the interrupt service routine in other general-purpose ROM area (Example2) to get more modularized coding style.

GENERAL PURPOSE PROGRAM MEMORY AREA

The ROM locations 0001H~0007H and 0009H~07FBH are used as general-purpose memory. The area is stored instruction's op-code and look-up table data. The SN8P2710 includes jump table function by using program counter (PC) and look-up table function by using ROM code registers (R, Y, Z).

The boundary of program memory is separated by the high-byte program counter (PCH) every 100H. In jump table function and look-up table function, the program counter can't leap over the boundary by program counter automatically. Users need to modify the PCH value to "PCH+1" as the PCL overflow (from 0FFH to 000H).

LOOKUP TABLE DESCRIPTION

In the ROM's data lookup function, Y register to the highest 8-bit and Z register to the lowest 8-bit data of ROM address. After MOVC instruction is executed, the low-byte data of ROM then will be stored in ACC and high-byte data stored in R register.

➲ **Example: To look up the ROM data located "TABLE1".**

```
                B0MOV     Y, #TABLE1$M    ; To set lookup table1's middle address
                B0MOV     Z, #TABLE1$L    ; To set lookup table1's low address.
                MOVC                      ; To lookup data, R = 00H, ACC = 35H
                                          ;
                                          ; Increment the index address for next address
                INCMS     Z               ; Z+1
                JMP       @F              ; Not overflow
                INCMS     Y               ; Z overflow (FFH → 00), → Y=Y+1
                NOP                       ; Not overflow
                                          ;
@@:             MOVC                      ; To lookup data, R = 51H, ACC = 05H.
                .         .               ;
TABLE1:         DW        0035H           ; To define a word (16 bits) data.
                DW        5105H           ; "
                DW        2012H           ; "
```

➢ **CAUSION: The Y register can't increase automatically if Z register cross boundary from 0xFF to 0x00. Therefore, user must take care such situation to avoid loop-up table errors. If Z register overflow, Y register must be added one. The following INC_YZ macro shows a simple method to process Y and Z registers automatically.**

➢ ***Note: Because the program counter (PC) is only 12-bit, the X register is useless in the application. Users can omit "B0MOV X, #TABLE1$H". SONiX ICE support more larger program memory addressing capability. So make sure X register is "0" to avoid unpredicted error in loop-up table operation.***

➲ **Example: INC_YZ Macro**

```
INC_YZ          MACRO
                INCMS     Z               ; Z+1
                JMP       @F              ; Not overflow
                
                INCMS     Y               ; Y+1
                NOP                       ; Not overflow
@@:
                ENDM
```

The other coding style of loop-up table is to add Y or Z index register by accumulator. Be careful if carry happen. Refer following example for detailed information:

➲ **Example: Increase Y and Z register by B0ADD/ADD instruction**

```
        B0MOV   Y, #TABLE1$M    ; To set lookup table's middle address.
        B0MOV   Z, #TABLE1$L    ; To set lookup table's low address.

        B0MOV   A, BUF          ; Z = Z + BUF.
        B0ADD   Z, A

        B0BTS1  FC              ; Check the carry flag.
        JMP     GETDATA         ; FC = 0
        INCMS   Y               ; FC = 1. Y+1.
        NOP

GETDATA:                        ;
        MOVC                    ; To lookup data. If BUF = 0, data is 0x0035
                                ; If BUF = 1, data is 0x5105
                                ; If BUF = 2, data is 0x2012
                                .
                                .
        .       .               ;
TABLE1: DW      0035H           ; To define a word (16 bits) data.
        DW      5105H           ; "
        DW      2012H           ; "
```

JUMP TABLE DESCRIPTION

The jump table operation is one of multi-address jumping function. Add low-byte program counter (PCL) and ACC value to get one new PCL. The new program counter (PC) points to a series jump instructions as a listing table. The way is easy to make a multi-stage program.

When carry flag occurs after executing of "ADD PCL, A", it will not affect PCH register. Users have to check if the jump table leaps over the ROM page boundary or the listing file generated by SONIX assembly software. If the jump table leaps over the ROM page boundary (e.g. from xxFFH to xx00H), move the jump table to the top of next program memory page (xx00H). **Here one page mean 256 words.**

➲ **Example : If PC = 0323H (PCH = 03H、PCL = 23H)**

```
        ORG     0X0100      ; The jump table is from the head of the ROM boundary

        B0ADD   PCL, A      ; PCL = PCL + ACC, the PCH can't be changed.
        JMP     A0POINT     ; ACC = 0, jump to A0POINT
        JMP     A1POINT     ; ACC = 1, jump to A1POINT
        JMP     A2POINT     ; ACC = 2, jump to A2POINT
        JMP     A3POINT     ; ACC = 3, jump to A3POINT
```

In following example, the jump table starts at 0x00FD. When execute B0ADD PCL, A. If ACC = 0 or 1, the jump table points to the right address. If the ACC is larger then 1 will cause error because PCH doesn't increase one automatically. We can see the PCL = 0 when ACC = 2 but the PCH still keep in 0. The program counter (PC) will point to a wrong address 0x0000 and crash system operation. It is important to check whether the jump table crosses over the boundary (xxFFH to xx00H). A good coding style is to put the jump table at the start of ROM boundary (e.g. 0100H).

➲ **Example: If "jump table" crosses over ROM boundary will cause errors.**

```
ROM Address
    .           .
    .           .
    .           .
0X00FD      B0ADD   PCL, A      ; PCL = PCL + ACC, the PCH can't be changed.
0X00FE      JMP     A0POINT     ; ACC = 0
0X00FF      JMP     A1POINT     ; ACC = 1
0X0100      JMP     A2POINT     ; ACC = 2   ← jump table cross boundary here
0X0101      JMP     A3POINT     ; ACC = 3
    .           .
    .           .
```

SONIX provides a macro for safe jump table function. This macro will check the ROM boundary and move the jump table to the right position automatically. The side effect of this macro is maybe wasting some ROM size. Notice the maximum jmp table number for this macro is limited under 254.

```
@JMP_A      MACRO   VAL
            IF      (($+1) !& 0XFF00) !!= (($+(VAL)) !& 0XFF00)
            JMP     ($ | 0XFF)
            ORG     ($ | 0XFF)
            ENDIF
            ADD     PCL, A
            ENDM
```

➢ ***Note: "VAL" is the number of the jump table listing number.***

➲ **Example: "@JMP_A" application in SONIX macro file called "MACRO3.H".**

```
B0MOV     A, BUF0       ; "BUF0" is from 0 to 4.
@JMP_A    5             ; The number of the jump table listing is five.
JMP       A0POINT       ; If ACC = 0, jump to A0POINT
JMP       A1POINT       ; ACC = 1, jump to A1POINT
JMP       A2POINT       ; ACC = 2, jump to A2POINT
JMP       A3POINT       ; ACC = 3, jump to A3POINT
JMP       A4POINT       ; ACC = 4, jump to A4POINT
```

If the jump table position is from 00FDH to 0101H, the "@JMP_A" macro will make the jump table to start from 0100h.

DATA MEMORY (RAM)

OVERVIEW

The SN8P2710 has internally built-in the data memory up to 128 bytes for storing the general-purpose data.

For SN8P2710
- 128 * 8-bit general purpose area in bank 0
- 128 * 8-bit system special register area

The memory is located in bank 0. The bank 0, using the first 128-byte location assigned as general-purpose area, and the remaining 128-byte in bank 0 as system register.

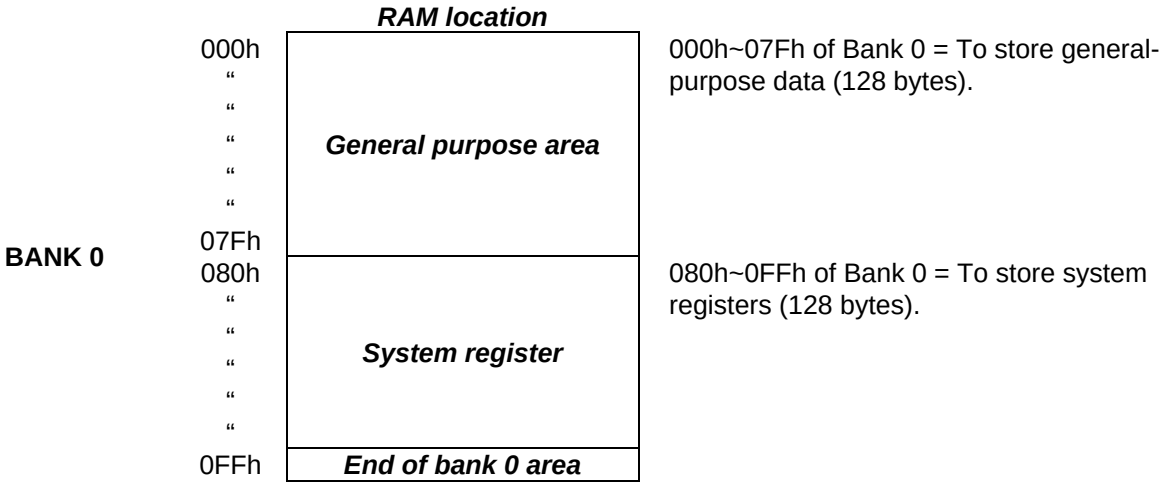


Figure 3-2 RAM Location of SN8P2710

- ***Note: The undefined locations of system register area are logic “high” after executing read instruction “MOV A, M”.***

WORKING REGISTERS

The locations 82H to 84H of RAM bank 0 in data memory stores the specially defined registers such as register R, Y, Z, respectively shown in the following table. These registers can use as the general purpose of working buffer and be used to access ROM's and RAM's data. For instance, all of the ROM's table can be looked-up with R, Y and Z registers. The data of RAM memory can be indirectly accessed with Y and Z registers.

	80H	81H	82H	83H	84H	85H	
RAM	-	-	R	Z	Y	-	
	-	-	R/W	R/W	R/W	-	

Y, Z REGISTERS

The Y and Z registers are the 8-bit buffers. There are three major functions of these registers. First, Y and Z registers can be used as working registers. Second, these two registers can be used as data pointers for @YZ register. Third, the registers can be address ROM location in order to look-up ROM data.

Y initial value = XXXX XXXX

084H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Y	YBIT7	YBIT6	YBIT5	YBIT4	YBIT3	YBIT2	YBIT1	YBIT0
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Z initial value = XXXX XXXX

083H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Z	ZBIT7	ZBIT6	ZBIT5	ZBIT4	ZBIT3	ZBIT2	ZBIT1	ZBIT0
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

The @YZ that is data point_1 index buffer located at address E7H in RAM bank 0. It employs Y and Z registers to addressing RAM location in order to read/write data through ACC. The Lower 4-bit of Y register is pointed to RAM bank number and Z register is pointed to RAM address number, respectively. The higher 4-bit data of Y register is truncated in RAM indirectly access mode.

➲ **Example: If want to read a data from RAM address 25H of bank 0, it can use indirectly addressing mode to access data as following.**

```
B0MOV   Y, #00H     ; To set RAM bank 0 for Y register
B0MOV   Z, #25H     ; To set location 25H for Z register
B0MOV   A, @YZ      ; To read a data into ACC
```

➲ **Example: Clear general-purpose data memory area of bank 0 using @YZ register.**

```
        MOV     A, #0
        B0MOV   Y, A            ; Y = 0, bank 0
        MOV     A, #07FH
        B0MOV   Z, A            ; Z = 7FH, the last address of the data memory area

CLR_YZ_BUF:
        CLR     @YZ             ; Clear @YZ to be zero

        DECMS   Z               ; Z – 1, if Z= 0, finish the routine
        JMP     CLR_YZ_BUF      ; Not zero

        CLR     @YZ
END_CLR:                        ; End of clear general purpose data memory area of bank 0
```

Note: Please consult the "LOOK-UP TABLE DESCRIPTION" about Y, Z register look-up table application.

R REGISTERS

There are two major functions of the R register. First, R register can be used as working registers. Second, the R registers can be store high-byte data of look-up ROM data. After MOVC instruction executed, the high-byte data of a ROM address will be stored in R register and the low-byte data stored in ACC.

R initial value = XXXX XXXX

082H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
R	RBIT7	RBIT6	RBIT5	RBIT4	RBIT3	RBIT2	RBIT1	RBIT0
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

➢ ***Note: Please consult the "LOOK-UP TABLE DESCRIPTION" about R register look-up table application.***

PROGRAM FLAG

The PFLAG includes reset flag, low voltage detect flag, carry flag, decimal carry flag (DC) and zero flag (Z). If the result of operating is zero or there is carry, borrow occurrence, then these flags will be set to PFLAG register.

PFLAG initial value = 00xx,x000

086H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
PFLAG	NT0	NPD	LVD36	LVD24	-	C	DC	Z
	R/W	R/W	R/W	R/W	-	R/W	R/W	R/W

RESET FLAG

NT0	NPD	Reset Status
0	0	Watch-dog time out
0	1	Reserved
1	0	Reset by LVD
1	1	Reset by external Reset Pin

LVD 2.4V FLAG

LVD24	VDD Status
1	VDD <= 2.4V
0	VDD > 2.4V

Note: This bit is only valid when code option LVD=LVD_M

LVD 3.6V FLAG

LVD36	VDD Status
1	VDD <= 3.6V
0	VDD > 3.6V

Note: This bit is only valid when code option LVD=LVD_H

CARRY FLAG

C = 1: If executed arithmetic addition with occurring carry signal or executed arithmetic subtraction without borrowing signal or executed rotation instruction with shifting out logic “1”.
C = 0: If executed arithmetic addition without occurring carry signal or executed arithmetic subtraction with borrowing signal or executed rotation instruction with shifting out logic “0”.

DECIMAL CARRY FLAG

DC = 1: If executed arithmetic addition with occurring carry signal from low nibble or executed arithmetic subtraction without borrow signal from high nibble.
DC = 0: If executed arithmetic addition without occurring carry signal from low nibble or executed arithmetic subtraction with borrow signal from high nibble.

ZERO FLAG

Z = 1: After operation, the content of ACC is zero.
Z = 0: After operation, the content of ACC is not zero.

ACCUMULATOR

The ACC is an 8-bits data register responsible for transferring or manipulating data between ALU and data memory. If the result of operating is zero (Z) or there is carry (C or DC) occurrence, then these flags will be set to PFLAG register.

ACC is not in data memory (RAM), so ACC can't be access by "B0MOV" instruction during the instant addressing mode.

➲ **Example: Read and write ACC value.**

```
; Read ACC data and store in BUF data memory

                MOV         BUF, A
                .           .

; Write a immediate data into ACC

                MOV         A, #0FH
                .           .

; Write ACC data from BUF data memory

                MOV         A, BUF
                .           .
```

The ACC value don't store in any interrupt service executed. ACC must be exchanged to another data memory defined by users. Thus, once interrupt occurs, these data must be stored in the data memory based on the user's program as follows.

➲ **Example: ACC and working registers protection.**

```
ACCBUF          EQU         00H             ; ACCBUF is ACC data buffer in bank 0.

INT_SERVICE:
                B0XCH       A, ACCBUF       ; B0XCH doesn't change C, Z flag
                B0MOV       A, PFLAG
                B0MOV       PFLAGBUF, A     ; Save PFLAG register in a buffer
                .
                .
                B0MOV       A, PFLAGBUF
                B0MOV       PFLAG, A        ; Restore PFLAG register from buffer
                B0XCH       A, ACCBUF       ; Re-load ACC

                RETI                        ; Exit interrupt service vector
```

➢ ***Notice: To save and re-load ACC data must be used "B0XCH" instruction, or the PLAGE value maybe modified by ACC.***

STACK OPERATIONS

OVERVIEW

The stack buffer of SN8P2710 has 8-level high area and each level is 11-bits length. This buffer is designed to save and restore program counter's (PC) data when interrupt service is executed. The STKP register is a pointer designed to point active level in order to save or restore data from stack buffer for kernel circuit. The STKnH and STKnL are the 12-bit stack buffers to store program counter (PC) data.

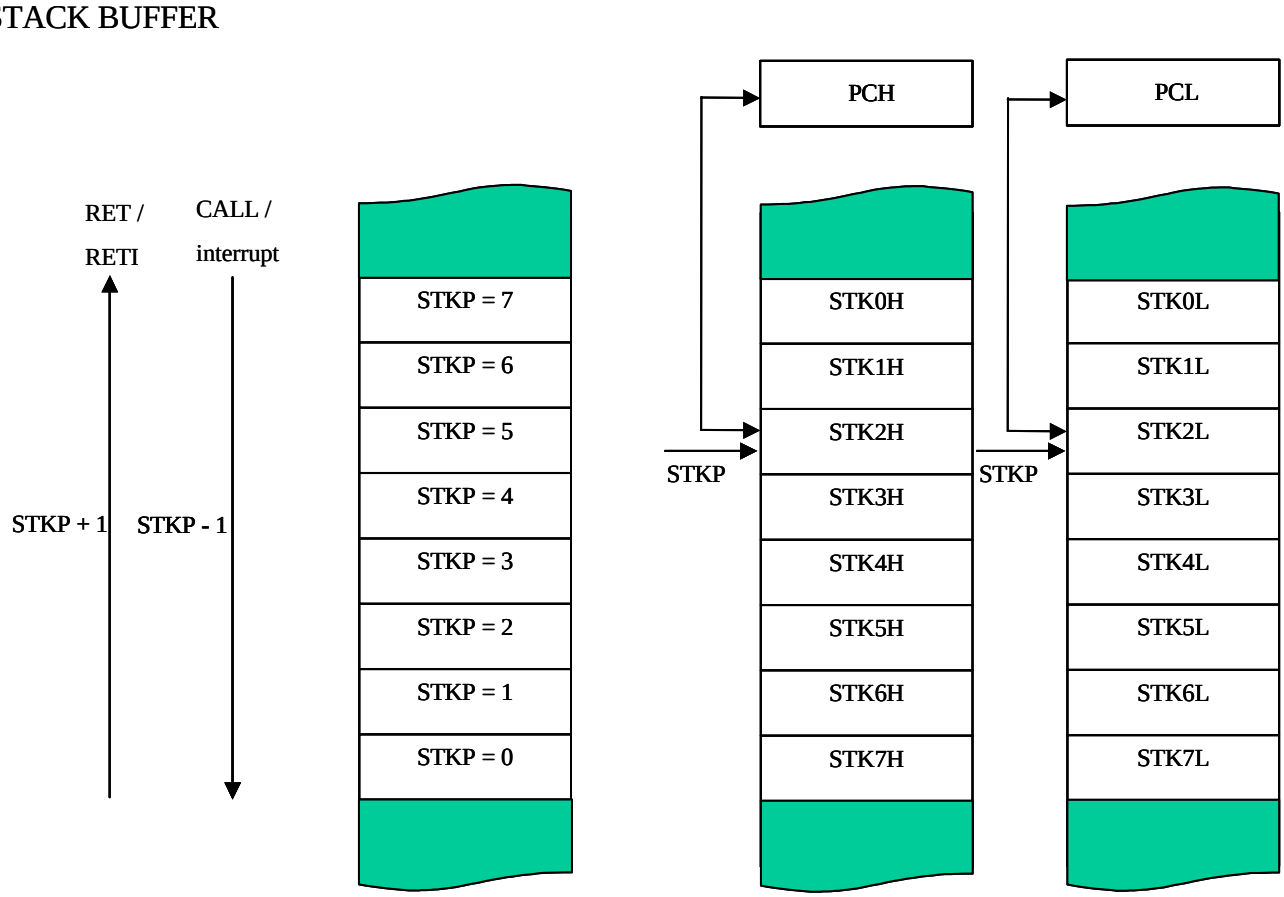


Figure 3-3 Stack Operation

STACK REGISTERS

The stack pointer (STKP) is a 4-bit register to store the address used to access the stack buffer, 11-bits data memory (STKnH and STKnL) set aside for temporary storage of stack addresses.

The two stack operations are writing to the top of the stack (Stack-Save) and reading (Stack-Restore) from the top of stack. Stack-Save operation decrements the STKP and the Stack-Resotre operation increments one time. That makes the STKP always points to the top address of stack buffer and writes the last program counter value (PC) into the stack buffer.

The program counter (PC) value is stored in the stack buffer before a CALL instruction executed or during interrupt service routine. Stack operation is a LIFO type (Last in and first out). The stack pointer (STKP) and stack buffer (STKnH and STKnL) are located in the system register area bank 0.

STKP (stack pointer) initial value = 0xxx x111

0DFH	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
STKP	GIE	-	-	-	-	STKPB2	STKPB1	STKPB0
	R/W	-	-	-	-	R/W	R/W	R/W

STKPBn: Stack pointer. (n = 0 ~ 3)
GIE: Global interrupt control bit. 0 = disable, 1 = enable. More detail information is in interrupt chapter.

➲ **Example: Stack pointer (STKP) reset routine.**

```
MOV     A, #00000111B
B0MOV   STKP, A
```

STKn (stack buffer) initial value = xxxx xxxx xxxx xxxx, STKn = STKnH + STKnL (n = 7 ~ 0)

0F0H~0FFH	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
STKnH	-	-	-	-	-	SnPC10	SnPC9	SnPC8
	-	-	-	-	-	R/W	R/W	R/W

0F0H~0FFH	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
STKnL	SnPC7	SnPC6	SnPC5	SnPC4	SnPC3	SnPC2	SnPC1	SnPC0
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

STKnH: Store PCH data as interrupt or call executing. The n expressed 0 ~7.
STKnL: Store PCL data as interrupt or call executing. The n expressed 0 ~7.

STACK OPERATION EXAMPLE

The two kinds of Stack-Save operations to reference the stack pointer (STKP) and write the program counter contents (PC) into the stack buffer are CALL instruction and interrupt service. Under each condition, the STKP is decremented and points to the next available stack location. The stack buffer stores the program counter about the op-code address. The Stack-Save operation is as following table.

Stack Level	STKP Register			Stack Buffer		Description
	STKPB2	STKPB1	STKPB0	High Byte	Low Byte	
0	1	1	1	STK0H	STK0L	-
1	1	1	0	STK1H	STK1L	-
2	1	0	1	STK2H	STK2L	-
3	1	0	0	STK3H	STK3L	-
4	0	1	1	STK4H	STK4L	-
5	0	1	0	STK5H	STK5L	-
6	0	0	1	STK6H	STK6L	-
7	0	0	0	STK7H	STK7L	-
>8	-	-	-	-	-	**Stack Overflow**

Table 3-1. STKP, STKnH and STKnL relative of Stack-Save Operation

The RETI instruction is for interrupt service routine. The RET instruction is for CALL instruction. When a Stack-Restore operation occurs, the STKP is incremented and points to the next free stack location. The stack buffer restores the last program counter (PC) to the program counter registers. The Stack-Restore operation is as following table.

Stack Level	STKP Register			Stack Buffer		Description
	STKPB2	STKPB1	STKPB0	High Byte	Low Byte	
7	0	0	0	STK7H	STK7L	-
6	0	0	1	STK6H	STK6L	-
5	0	1	0	STK5H	STK5L	-
4	0	1	1	STK4H	STK4L	-
3	1	0	0	STK3H	STK3L	-
2	1	0	1	STK2H	STK2L	-
1	1	1	0	STK1H	STK1L	-
0	1	1	1	STK0H	STK0L	-

Table 3-2. STKP, STKnH and STKnL relative of Stack-Restore Operation

PROGRAM COUNTER

The program counter (PC) is a 11-bit binary counter separated into the high-byte 3 bits and the low-byte 8 bits. This counter is responsible for pointing a location in order to fetch an instruction for kernel circuit. Normally, the program counter is automatically incremented with each instruction during program execution.

Besides, it can be replaced with specific address by executing CALL or JMP instruction. When JMP or CALL instruction is executed, the destination address will be inserted to bit 0 ~ bit 10.

PC Initial value = xxxx 0000 0000 0000

	Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
PC	-	-	-	-	-	0	0	0	0	0	0	0	0	0	0	0
	PCH								PCL							

PCH Initial value = xxxx x000

0CFH	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
PCH	-	-	-	-	-	PC10	PC9	PC8
	-	-	-	-	-	R/W	R/W	R/W

PCL Initial value = 0000 0000

0CEH	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
PCL	PC7	PC6	PC5	PC4	PC3	PC2	PC1	PC0
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

ONE ADDRESS SKIPPING

There are 7 instructions (CMPRS, INCS, INCMS, DECS, DECMS, B0BTS0, B0BTS1) with one address skipping function. If the result of these instructions is matched, the PC will add 2 steps to skip next instruction.

If the condition of bit test instruction is matched, the PC will add 2 steps to skip next instruction.

```
            B0BTS1    FC          ; Skip next instruction, if Carry_flag = 1
            JMP       C0STEP      ; Else jump to C0STEP.
            .
C0STEP:     NOP

            B0MOV     A, BUF0     ; Move BUF0 value to ACC.
            B0BTS0    FZ          ; Skip next instruction, if Zero flag = 0.
            JMP       C1STEP      ; Else jump to C1STEP.
            .
C1STEP:     NOP
```

If the ACC is equal to the immediate data or memory, the PC will add 2 steps to skip next instruction.

```
            CMPRS     A, #12H     ; Skip next instruction, if ACC = 12H.
            JMP       C0STEP      ; Else jump to C0STEP.
            .
C0STEP:     NOP
```

If the result after increasing 1 or decreasing 1 is 0xffh (for DECS and DECMS) or 0x00h (for INCS and INCMS) , the PC will add 2 steps to skip next instruction.

INCS instruction:

```
            INCS      BUF0        ; Skip next instruction, if BUF0 = 0X00H.
            JMP       C0STEP      ; Else jump to C0STEP.
            .
C0STEP:     NOP
```

INCMS instruction:

```
            INCMS     BUF0        ; Skip next instruction, if BUF0 = 0X00H.
            JMP       C0STEP      ; Else jump to C0STEP.
            .
C0STEP:     NOP
```

DECS instruction:

```
            DECS      BUF0        ; Skip next instruction, if BUF0 = 0XFFH.
            JMP       C0STEP      ; Else jump to C0STEP.
            .
C0STEP:     NOP
```

DECMS instruction:

```
            DECMS     BUF0        ; Skip next instruction, if BUF0 = 0XFFH.
            JMP       C0STEP      ; Else jump to C0STEP.
            .
C0STEP:     NOP
```

MULTI-ADDRESS JUMPING

Users can jump round multi-address by either JMP instruction or ADD M, A instruction (M = PCL) to activate multi-address jumping function. If carry signal occurs after execution of ADD PCL, A, the carry signal will not affect PCH register.

➲ Example: If PC = 0323H (PCH = 03H、PCL = 23H)

```
; PC = 0323H
                MOV     A, #28H
                B0MOV   PCL, A          ; Jump to address 0328H
                .       .
                .       .
; PC = 0328H     .       .
                MOV     A, #00H
                B0MOV   PCL, A          ; Jump to address 0300H
```

➲ Example: If PC = 0323H (PCH = 03H、PCL = 23H)

```
; PC = 0323H
                B0ADD   PCL, A          ; PCL = PCL + ACC, the PCH cannot be changed.
                JMP     A0POINT         ; If ACC = 0, jump to A0POINT
                JMP     A1POINT         ; ACC = 1, jump to A1POINT
                JMP     A2POINT         ; ACC = 2, jump to A2POINT
                JMP     A3POINT         ; ACC = 3, jump to A3POINT
                .       .               ;
```

4. ADDRESSING MODE

OVERVIEW

The SN8P2710 provides three addressing modes to access RAM data, including immediate addressing mode, directly addressing mode and indirectly address mode. The main purpose of the three different modes is described in the following:

IMMEDIATE ADDRESSING MODE

The immediate addressing mode uses an immediate data to set up the location (MOV A, #I, B0MOV M,#I) in ACC or specific RAM.

Immediate addressing mode

```
MOV     A, #12H         ; To set an immediate data 12H into ACC
```

DIRECTLY ADDRESSING MODE

The directly addressing mode uses address number to access memory location (MOV A,12H, MOV 12H,A).

Directly addressing mode

```
B0MOV   A, 12H          ; To get a content of location 12H of bank 0 and save in ACC
```

INDIRECTLY ADDRESSING MODE

The indirectly addressing mode is to set up an address in data pointer registers (Y/Z) and uses MOV instruction to read/write data between ACC and @YZ register (MOV A,@YZ, MOV @YZ,A).

➲ **Example: Indirectly addressing mode with @YZ register**

```
CLR     Y               ; To clear Y register to access RAM bank 0.
B0MOV   Z, #12H         ; To set an immediate data 12H into Z register.
B0MOV   A, @YZ          ; Use data pointer @YZ reads a data from RAM location
                        ; 012H into ACC.
```

TO ACCESS DATA in RAM BANK 0

In the RAM bank 0, this area memory can be read/written by these twoaccess methods.

➲ **Example 1: To use RAM bank0 dedicate instruction (Such as B0xxx instruction).**

```
B0MOV    A, 12H        ; To move content from location 12H of RAM bank 0 to ACC
```

➲ **Example 2: To use indirectly addressing mode with @YZ register.**

```
CLR      Y             ; To clear Y register for accessing RAM bank 0.
B0MOV    Z, #12H       ; To set an immediate data 12H into Z register.
B0MOV    A, @YZ        ; Use data pointer @YZ reads a data from RAM location
                       ; 012H into ACC.
```

5. SYSTEM REGISTER

OVERVIEW

The system special register is located at 80h~FFh. The main purpose of system registers is to control the peripheral hardware of the chip. Using system registers can control I/O ports, ADC, PWM, timers and counters by programming. The Memory map provides an easy and quick reference source for writing application program.

SYSTEM REGISTER ARRANGEMENT (BANK 0)

BYTES of SYSTEM REGISTER

SN8P2710

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
8	-	-	R	Z	Y	-	PFLAG	-	-	-	-	-	-	-	-	-
9	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
A	-	-	-	-	-	-	-	-	-	-	-	-	-	-	P4CON	-
B	DAM	ADM	ADB	ADR	-	-	-	-	-	-	-	-	-	-	-	PEDGE
C	-	-	P2M	-	P4M	P5M	-	-	INTRQ	INTEN	OSCM	-	WDTR	TC0R	PCL	PCH
D	P0	-	P2	-	P4	P5	-	-	T0M	-	TC0M	TC0C	TC1M	TC1C	TC1R	STKP
E	P0UR	-	P2UR	-	P4UR	P5UR	-	@YZ	-	-	-	-	-	-	-	-
F	STK7L	STK7H	STK6L	STK6H	STK5L	STK5H	STK4L	STK4H	STK3L	STK3H	STK2L	STK2H	STK1L	STK1H	STK0L	STK0H

Table 5-1. System Register Arrangement of SN8P2710

Description

PFLAG = ROM page and special flag register.
DAM = DAC's mode register.
ADB = ADC's data buffer.
PnM = Port n input/output mode register.
INTRQ = Interrupts' request register.
OSCM = Oscillator mode register.
T0M = Timer/ Counter 0, Timer/ Counter 1 speed selection .
TC1M = Timer/Counter 1 mode register.
TC1C = Timer/Counter 1 counting register.
STKP = Stack pointer buffer.
@HL = RAM HL indirect addressing index pointer.
P4CON= Port 4 configuration setting

R = Working register and ROM lookup data buffer.
Y, Z = Working, @YZ and ROM addressing register.
ADM = ADC's mode register.
ADR = ADC's resolution selects register.
Pn = Port n data buffer.
INTEN = Interrupts' enable register.
PCH, PCL = Program counter.
TC0M = Timer/Counter 0 mode register.
TC0C = Timer/Counter 0 counting register.
TC0R = Timer/Counter 0 auto-reload data buffer.
TC1R = Timer/Counter 1 auto-reload data buffer.
STK0~STK7 = Stack 0 ~ stack 7 buffer.
@YZ = RAM YZ indirect addressing index pointer.

➢ ***Note:***

a). All of register names had been declared in SONiX 8-bit MCU assembler.
b). One-bit name had been declared in SONiX 8-bit MCU assembler with "F" prefix code.
c). It will get logic "H" data, when use instruction to check empty location.
d). The low nibble of ADR register is read only.
e). "b0bset", "b0bclr", "bset", "bclr" instructions only support "R/W" registers.

BITS of SYSTEM REGISTER

SN8P2710 System register table

Address	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	R/W	Remarks
080H										
081H										
082H	RBIT7	RBIT6	RBIT5	RBIT4	RBIT3	RBIT2	RBIT1	RBIT0	R/W	R
083H	ZBIT7	ZBIT6	ZBIT5	ZBIT4	ZBIT3	ZBIT2	ZBIT1	ZBIT0	R/W	Z
084H	YBIT7	YBIT6	YBIT5	YBIT4	YBIT3	YBIT2	YBIT1	YBIT0	R/W	Y
085H										
086H	NT0	NPD	LVD36	LVD24	-	C	DC	Z	R/W	PFLAG
087H	-	-	-	-	-	-	-	-	-	-
0AEH	P4CON7	P4CON6	P4CON 5	P4CON4	P4CON3	P4CON2	P4CON1	P4CON0	W	P4CON
0B0H	DAENB	DAB6	DAB5	DAB4	DAB3	DAB2	DAB1	DAB0	R/W	DAM data register
0B1H	ADENB	ADS	EOC	GCHS	-	CHS2	CHS1	CHS0	R/W	ADM mode register
0B2H	ADB11	ADB10	ADB9	ADB8	ADB7	ADB6	ADB5	ADB4	R	ADB data buffer
0B3H	-	ADCKS1	-	ADCKS0	ADB3	ADB2	ADB1	ADB0	R/W	ADR register
0B4H	-	-	-	-	-	-	-	-	-	
0B5H	-	-	-	-	-	-	-	-	-	
0B6H	-	-	-	-	-	-	-	-	-	
0B8H	-	-	-	-	-	-	-	-	-	
0BFH	-	-	-	P00G1	P00G0	-	-	-	R/W	PEDGE
0C0H	-	-	-	-	-	-	-	-	-	
0C1H	-	-	-	-	-	-	-	-	-	
0C2H	P27M	P26M	P25M	P24M	P23M	P22M	P21M	P20M	R/W	P2M I/O direction
0C3H	-	-	-	-	-	-	-	-	-	
0C4H	P47M	P46M	P45M	P44M	P43M	P42M	P41M	P40M	R/W	P4M I/O direction
0C5H	-	P56M	P55M	P54M	P53M	P52M	P51M	P50M	R/W	P5M I/O direction
0C8H	-	TC1IRQ	TC0IRQ	-	-	-	P01IRQ	P00IRQ	R/W	INTRQ
0C9H	-	TC1IEN	TC0IEN	-	-	-	P01IEN	P00IEN	R/W	INTEN
0CAH	-	-	-	-	CPUM0	CLKMD	STPHX	-	R/W	OSCM
0CCH	WDTR7	WDTR6	WDTR5	WDTR4	WDTR3	WDTR2	WDTR1	WDTR0	W	WDTR
0CDH	TC0R7	TC0R6	TC0R5	TC0R4	TC0R3	TC0R2	TC0R1	TC0R0	W	TC0R
0CEH	PC7	PC6	PC5	PC4	PC3	PC2	PC1	PC0	R/W	PCL
0CFH	-	-	-	-	-	PC10	PC9	PC8	R/W	PCH
0D0H	-	-	-	-	P03	P02	P01	P00	R	P0 data buffer
0D1H	-	-	-	-	-	-	-	-	-	
0D2H	P27	P26	P25	P24	P23	P22	P21	P20	R/W	P2 data buffer
0D3H	-	-	-	-	-	-	-	-	-	
0D4H	P47	P46	P45	P44	P43	P42	P41	P40	R/W	P4 data buffer
0D5H	-	P56	P55	P54	P53	P52	P51	P50	R/W	P5 data buffer
0D8H	-	-	-	-	TC1X8	TC0X8	-	-	R/W	T0M
0D9H	-	-	-	-	-	-	-	-	-	
0DAH	TC0ENB	TC0rate2	TC0rate1	TC0rate0	TC0CKS	ALOAD0	TC0OUT	PWM0OUT	R/W	TC0M
0DBH	TC0C7	TC0C6	TC0C5	TC0C4	TC0C3	TC0C2	TC0C1	TC0C0	R/W	TC0C
0DCH	TC1ENB	TC1rate2	TC1rate1	TC1rate0	TC1CKS	ALOAD1	TC1OUT	PWM1OUT	R/W	TC1M
0DDH	TC1C7	TC1C6	TC1C5	TC1C4	TC1C3	TC1C2	TC1C1	TC1C0	R/W	TC1C
0DEH	TC1R7	TC1R6	TC1R5	TC1R4	TC1R3	TC1R2	TC1R1	TC1R0	W	TC1R
0DFH	GIE	-	-	-	-	STKPB2	STKPB1	STKPB0	R/W	STKP stack pointer
0E0H	-	-	-	-	-	P02R	P01R	P00R	W	P0UR
0E1H	-	-	-	-	-	-	-	-	-	
0E2H	P27R	P26R	P25R	P24R	P23R	P22R	P21R	P20R	W	P2UR
0E3H	-	-	-	-	-	-	-	-	-	
0E4H	P47R	P46R	P45R	P44R	P43R	P42R	P41R	P40R	W	P4UR
0E5H	-	P56R	P54R	P54R	P53R	P52R	P51R	P50R	W	P5UR
0E6H	-	-	-	-	-	-	-	-	-	
0E7H	@YZ7	@YZ6	@YZ5	@YZ4	@YZ3	@YZ2	@YZ1	@YZ0	R/W	@YZ index pointer
0E9H	-	-	-	-	-	-	-	-	-	
0F0H	S7PC7	S7PC6	S7PC5	S7PC4	S7PC3	S7PC2	S7PC1	S7PC0	R/W	STK7L
0F1H	-	-	-	-	-	S7PC10	S7PC9	S7PC8	R/W	STK7H
0F2H	S6PC7	S6PC6	S6PC5	S6PC4	S6PC3	S6PC2	S6PC1	S6PC0	R/W	STK6L
0F3H	-	-	-	-	-	S6PC10	S6PC9	S6PC8	R/W	STK6H

Address	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	R/W	Remarks
0F4H	S5PC7	S5PC6	S5PC5	S5PC4	S5PC3	S5PC2	S5PC1	S5PC0	R/W	STK5L
0F5H	-	-	-	-	-	S5PC10	S5PC9	S5PC8	R/W	STK5H
0F6H	S4PC7	S4PC6	S4PC5	S4PC4	S4PC3	S4PC2	S4PC1	S4PC0	R/W	STK4L
0F7H	-	-	-	-	-	S4PC10	S4PC9	S4PC8	R/W	STK4H
0F8H	S3PC7	S3PC6	S3PC5	S3PC4	S3PC3	S3PC2	S3PC1	S3PC0	R/W	STK3L
0F9H	-	-	-	-	-	S3PC10	S3PC9	S3PC8	R/W	STK3H
0FAH	S2PC7	S2PC6	S2PC5	S2PC4	S2PC3	S2PC2	S2PC1	S2PC0	R/W	STK2L
0FBH	-	-	-	-	-	S2PC10	S2PC9	S2PC8	R/W	STK2H
0FCH	S1PC7	S1PC6	S1PC5	S1PC4	S1PC3	S1PC2	S1PC1	S1PC0	R/W	STK1L
0FDH	-	-	-	-	-	S1PC10	S1PC9	S1PC8	R/W	STK1H
0FEH	S0PC7	S0PC6	S0PC5	S0PC4	S0PC3	S0PC2	S0PC1	S0PC0	R/W	STK0L
0FFH	-	-	-	-	-	S0PC10	S0PC9	S0PC8	R/W	STK0H

Table. Bit System Register Table of SN8P2710

Note:
a). To avoid system error, please be sure to put all the "0" as it indicates in the above table
b). All of register name had been declared in SONiX 8-bit MCU assembler.
c). One-bit name had been declared in SONiX 8-bit MCU assembler with "F" prefix code.
d). "b0bset", "b0bclr", "bset", "bclr" instructions only support "R/W" registers.
e). For detail description please refer file of "System Register Quick Reference Table"

6. POWER ON RESET

OVERVIEW

The MCU resets in power on reset, external reset and watchdog reset. Under one of reset status, system registers keep initial status and program stops. After reset status released, the system inserts normal mode and operates. The reset timing diagram is as following.

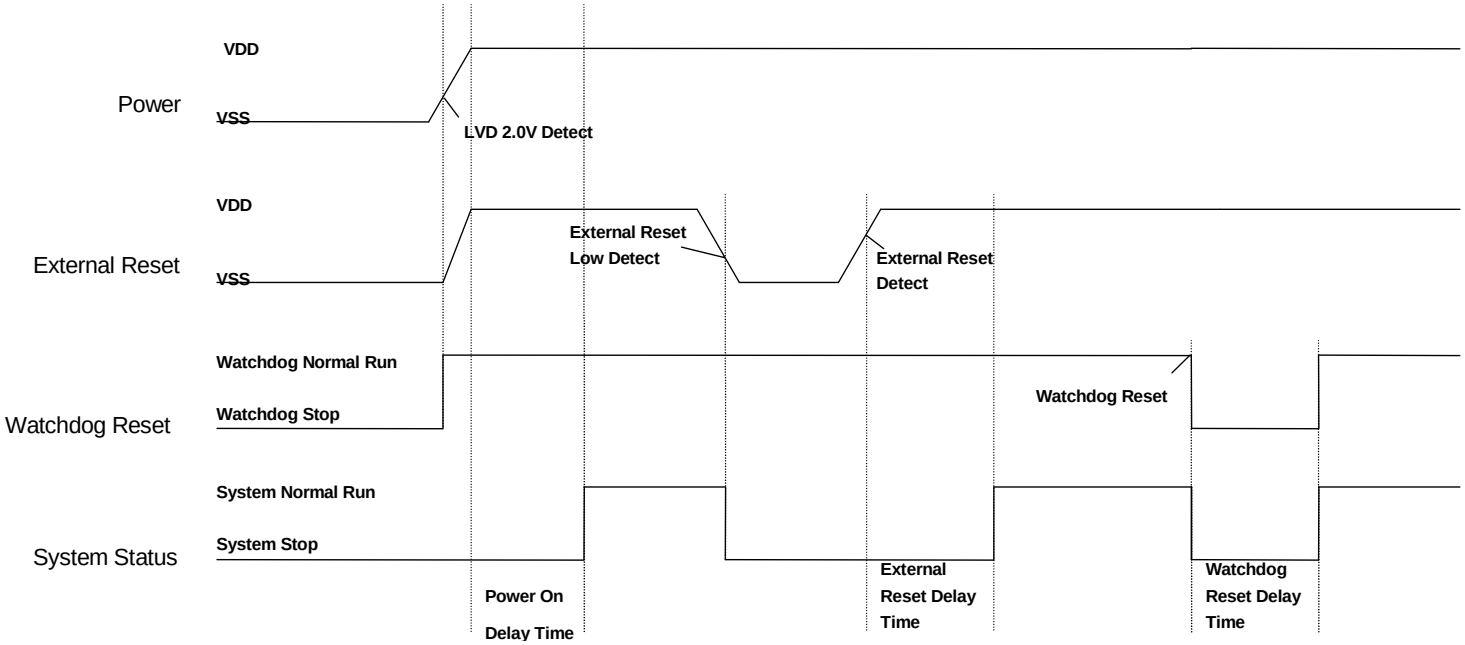


Figure 6-1 Reset Timing Chart

POWER ON RESET

Turn on power and the power rises from VSS to VDD. The system is reset and inserts normal mode. The period from power on to system running is called “Power On Delay Time”. System works well after power on delay time.

VDD	*Crystal*	*Power On Delay Time*
3V	4MHz	About 129ms
5V	4MHz	About 65ms
3V	32768Hz	About 237ms
5V	32768Hz	About 173ms

EXTERNAL RESET

External reset function is enabled when external reset pin is set as reset mode. The external reset pin is Schmitt Trigger structure. External reset is a low level active device. The reset pin receives the low voltage (less than 0.3VDD) and resets the system. When the voltage detects high level (higher than 0.7VDD), it stops reset the system. Users can use an external reset circuit to control system operation.

VDD	*Crystal*	*External Reset Delay Time*
3V	4MHz	About 129ms
5V	4MHz	About 65ms
3V	32768Hz	About 237ms
5V	32768Hz	About 173ms

EXTERNAL RESET CIRCUIT

The external reset circuit is a simple RC circuit as the following diagram.

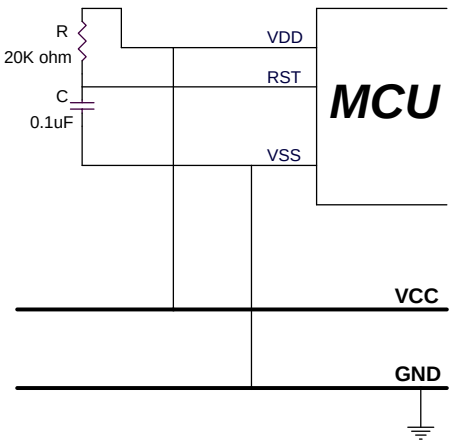


Figure 6-2 **Simply RC Reset Circuit**

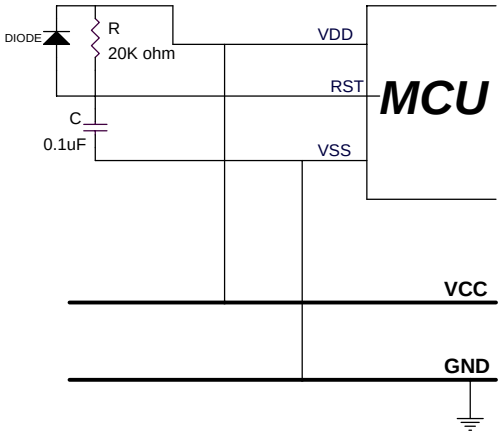


Figure 6-3 **Diode Reset Circuit for Brownout Reset**

In high noisy environment, remove capacitor between RST pin and ground to improve EFT (Electrical Fast Transients) protection capability.

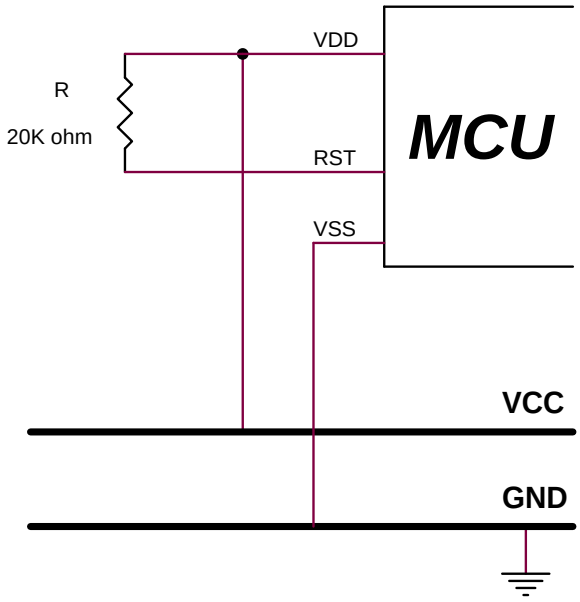


Figure 6-4 Reset circuit in High Noisy Environment

WATCHDOG RESET

Watchdog reset is a system protection. In normal condition, system works well and clears watchdog timer by program. Under error condition, system is in unknown situation and watchdog can't be clear by program before watchdog timer overflow. Watchdog timer overflow occurs and the system is reset. After watchdog reset, the system restarts and returns normal mode.

VDD	*Crystal*	*Watchdog Reset Delay Time*
3V	4MHz	About 145ms
5V	4MHz	About 73ms
3V	32768Hz	About 253ms
5V	32768Hz	About 181ms

LOW VOLTAGE DETECTOR (LVD)

There is LVD built in system. LVD is low power detector. The detect level is about **2.0V**. If VDD drops to 2.0V lower, system will reset. If VDD returns higher than 2.0V, system start to execute power on routine and system works well after power on reset delay time. It is called Brown Out Reset. Under high noisy situation, the power might be unstable. LVD avoids noise to effect system. When VDD keeps higher than 2.0V, system keeps working well status. If VDD lower than 2.0V by noise, LVD makes system reset and waits VDD return to normal level. LVD reset delay is power on reset delay time.

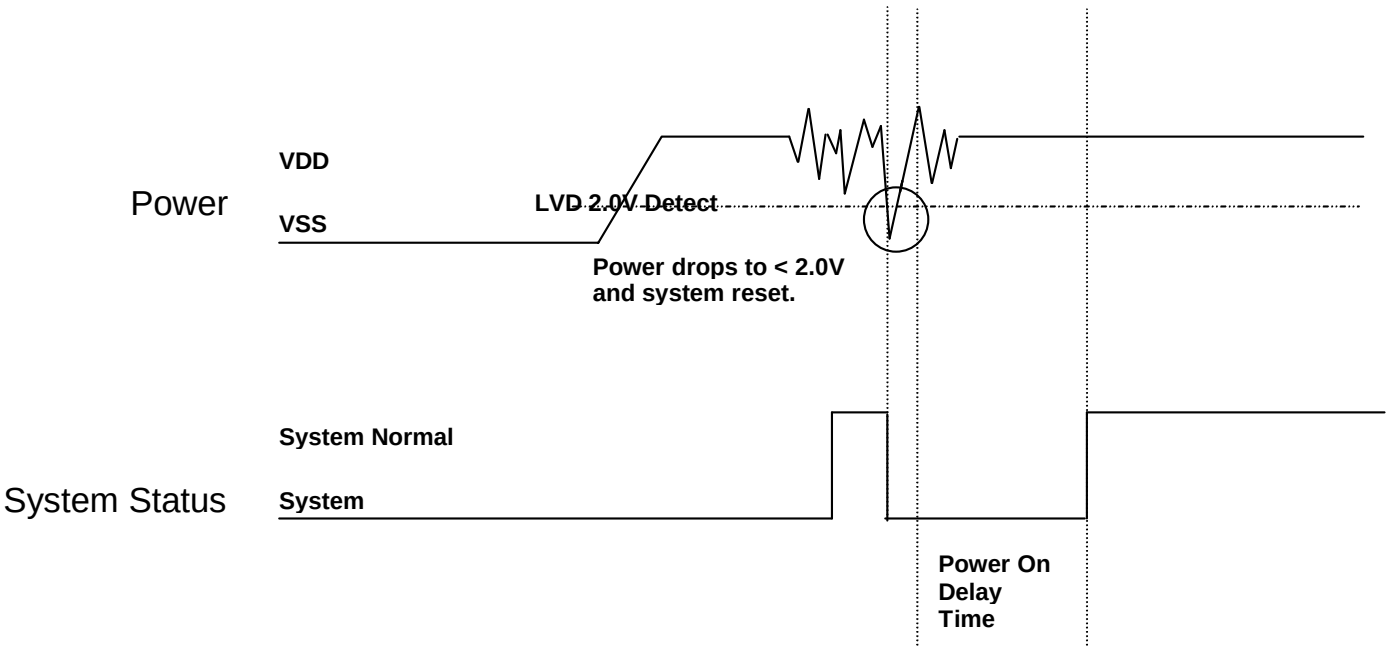


Figure 6-5 LVD Reset Timing Chart

The LVD is three levels design (2.0V/2.4V/3.6V) and controlled by LVD code option. The 2.0V LVD is always enable for power on reset and Brown Out reset. The 2.4V LVD includes the LVD reset function and flag function to indicate VDD status function. The 3.6V include flag function to indicate VDD status. LVD flag function can be an **easy low battery detector.**

LVD	LVD Code Option		
	LVD_L	LVD_M	LVD_H
2.0V Reset	Available	Available	Available
2.4V Flag	-	Available	-
2.4V Reset	-	-	Available
3.6V Flag	-	-	Available

LVD_L
If VDD<2.0V, system will be reset
Disable LVD24 and LVD36 bit of PFLAG register.

LVD_M
If VDD<2.0V, system will be reset
Enable LVD24 bit of PFLAG register. If VDD > 2.4V, LVD24 is “0”. If VDD < 2.4V, LVD24 flag is “1”.
Disable LVD36 bit of PFLAG register.

LVD_H
If VDD<2.4V, system will be reset
Enable LVD24 bit of PFLAG register. If VDD > 2.4V, LVD24 is “0”. If VDD < 2.4V, LVD24 flag is “1”.
Enable LVD36 bit of PFLAG register. If VDD > 3.6V, LVD36 is “0”. If VDD < 3.6V, LVD36 flag is “1”.

LVD24, LVD36 flags are real time indicating VDD voltage status. For low battery detect application, only checking LVD24, LVD36 status to be battery status. This is a cheap and easy solution.

✷ **Note:**
1. **After any LVD reset, LVD24, LVD36 flags are cleared.**
2. ***The voltage level of LVD 2.4V or 3.6V is for design reference only. Don’t use the LVD indicator as precision VDD measurement.***

7. OSCILLATORS

OVERVIEW

The SN8P2710 highly performs the dual clock micro-controller system. The dual clocks are high-speed clock and low-speed clock. The high-speed clock frequency is supplied through the external oscillator circuit. The low-speed clock frequency is supplied through on-chip RC oscillator circuit.

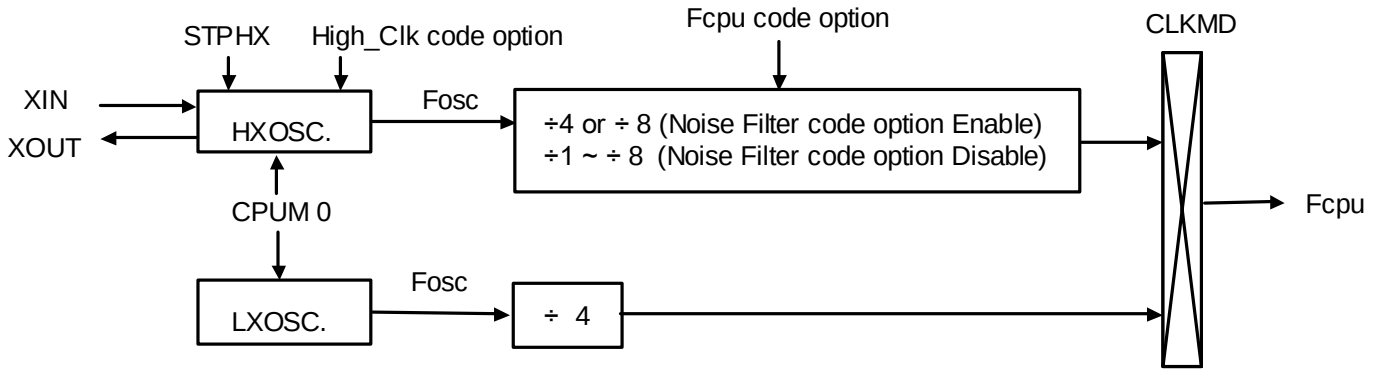


Figure 7-1 System Clock Block Diagram

- Fosc = Fhosc (External high clock) in normal mode
- Fosc = Flosc (Internal low RC clock) in slow mode
- Fosc is system clock, Fcpu is instruction cycle clock
- Fcpu = Fosc/1 ~ Fosc/8 in normal mode
- Fcpu = Fosc/4 in slow mode

The system clock is required by the following peripheral modules:

- ✓ ***Timer (TC0 / TC1 / Watchdog timer)***
- ✓ ***PWM output (PWM0, PWM1)***
- ✓ ***Buzzer output (TC0OUT, TC1OUT)***
- ✓ ***ADC converter***

OSCM REGISTER DESCRIPTION

The OSCM register is an oscillator control register. It can control oscillator select, system mode.

OSCM initial value = xxx0 000x

0CAH	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
OSCM	-	-	-	-	CPUM0	CLKMD	STPHX	-
	-	-	-	-	R/W	R/W	R/W	-

Bit [4:3] **CPUM 0:** CPU operating mode control bit.
0 = normal mode
1 = sleep (power down) mode. Always enter normal mode after wakeup.

Bit 2 **CLKMD:** Select instruction cycle clock (Fcpu) source
0 Fcpu came form External high clock (Fhosc). Operation mode is normal mode.
1 Fcpu came from Internal low clock (Flosc). Operation mode is slow mode.

Bit1 **STPHX**: Stop High clock control bit.
0=High clock free running.
1=High clock stop.

EXTERNAL HIGH-SPEED OSCILLATOR

The high clock oscillator of SN8P2710 can be configured as four different oscillator types. There are external RC oscillator modes, high crystal/resonator mode (12M code option), standard crystal/resonator mode (4M code option) and low crystal mode (32K code option). For different application, the users can select one of suitable oscillator mode by programming “High_Clk” code option to generate system high-speed clock source after reset.

➲ **Example: Stop external high-speed oscillator.**

```
B0BSET    FSTPHX      ; To stop external high-speed oscillator only.

B0BSET    FCPUM0      ; To stop external high-speed oscillator and internal low-speed
                      ; oscillator called power down mode (sleep mode).
```

HIGH CLOCK OSCILLATOR CODE OPTION

SN8P2710 provide four oscillator modes for different applications. These modes are 4M, 12M, 32K and RC. The main purpose is to support different oscillator types and frequencies. High-speed crystal needs more current but the low one doesn’t. For crystals, there are three steps to select. User can select oscillator mode from Code Option table before compiling. The table is as follow.

Code Option	Content	Function Description
High_Clk	Ext_RC	Low cost external RC oscillator for high clock oscillator. Output the Fcpu clock from Xout pin.
	32K_X’tal	Low frequency, power saving crystal (e.g. 32.768KHz) for high clock oscillator.
	12M_X’tal	High speed crystal /resonator (e.g. 12MHz ~ 16MHz) for high clock oscillator.
	4M_X’tal	Middle speed crystal /resonator (e.g. 4MHz ~ 10Mhz) for high clock oscillator.

SYSTEM OSCILLATOR CIRCUITS

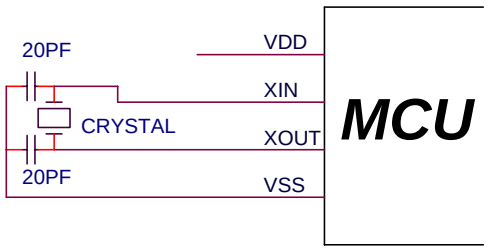


Figure 7-2. Crystal/Ceramic Oscillator

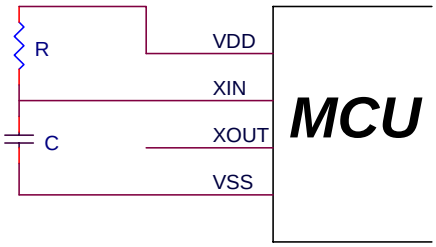


Figure 7-3. RC Oscillator

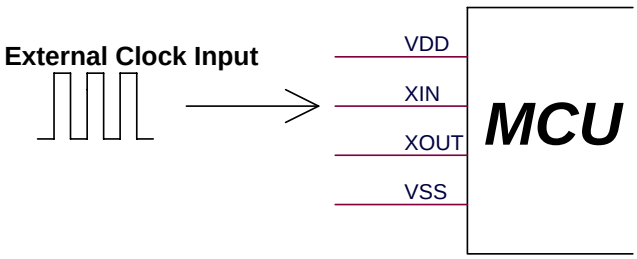


Figure 7-4. External clock input

- ***Note1: The VDD and VSS of external oscillator circuit must be from the micro-controller. Don't connect them from the neighbor power terminal.***
- ***Note2: The external clock input mode can select RC type oscillator or crystal type oscillator of the code option and input the external clock into XIN pin.***
- ***Note3: The power and ground of external oscillator circuit must be connected from the micro-controller's VDD and VSS. It is necessary to step up the performance of the whole system.***

External RC Oscillator Frequency Measurement

There is one way to get the Fosc frequency of external RC oscillator by instruction cycle (Fcpu). We can get the Fosc frequency of external RC from the Fcpu frequency. The sub-routine to get Fcpu frequency of external oscillator is as the following.

➲ Example: Fcpu instruction cycle of external oscillator

```
        B0BSET    P2M.0       ; Set P2.0 to be output mode for outputting Fcpu toggle
                                  signal.

@@:
        B0BSET    P2.0        ; Output Fcpu toggle signal in low-speed clock mode.
        B0BCLR    P2.0        ; Measure the Fcpu frequency by oscilloscope.
        JMP       @B
```

INTERNAL LOW-SPEED OSCILLATOR

The internal low-speed oscillator is built in the micro-controller. The low-speed clock's source is a RC type oscillator circuit. The low-speed clock can supplies clock for system clock, timer counter, watchdog timer, and so on.

➲ **Example: Stop internal low-speed oscillator.**

```
B0BSET    FCPUM0      ; To stop external high-speed oscillator and internal low-speed
                      ; oscillator called power down mode (sleep mode).
```

➢ ***Note: The internal low-speed clock can't be turned off individually. It is controlled by CPUM0 bit of OSCM register.***

The low-speed oscillator uses RC type oscillator circuit. The frequency is affected by the voltage and temperature of the system. In common condition, the frequency of the RC oscillator is about 16KHz at 3V and 32KHz at 5V. The relative between the RC frequency and voltage is as following.

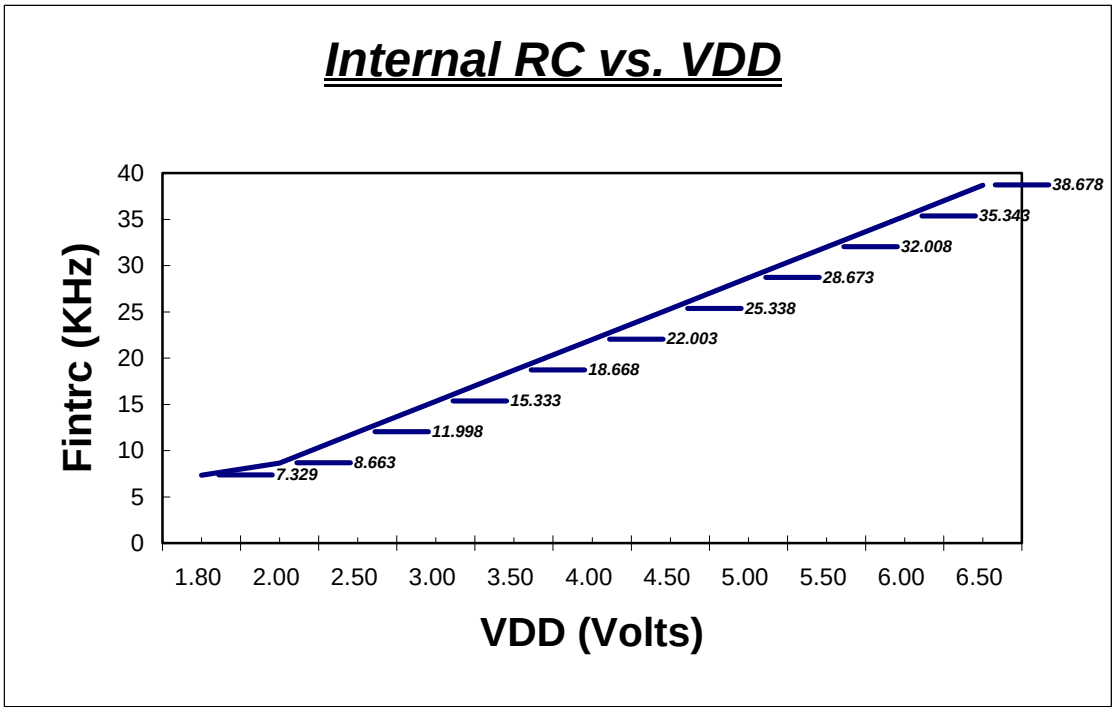


Figure 7-5. Internal RC vs. VDD Diagram

➲ **Example: To measure the internal RC frequency is by instruction cycle (Fcpu). The internal RC frequency is the Fcpu multiplied by 4. So we can get the Fosc frequency of internal RC from the Fcpu frequency.**

```
        B0BSET    P2M.0       ; Set P2.0 to be output mode for outputting Fcpu toggle signal.

        B0BSET    FCLKMD      ; Switch the system clock to internal low-speed clock mode.
@@:
        B0BSET    P2.0        ; Output Fcpu toggle signal in low-speed clock mode.
        B0BCLR    P2.0        ; Measure the Fcpu frequency by oscilloscope.
        JMP       @B
```

SYSTEM MODE DESCRIPTION

OVERVIEW

The chip is featured with low power consumption by switching around three different modes as following.
In actual application, the user can adjust the chip's controller to work in these three modes by using OSCM register.

NORMAL MODE

In normal mode, the system clock source is external high-speed clock. After power on, the system works under normal mode. All software and hardware are executed and working. In normal mode, system can get into power down mode and slow mode.

SLOW MODE

In slow mode, the system clock source is internal low-speed RC clock. To set CLKMD = 1, the system switch to slow mode. In slow mode, the system works as normal mode but the slower clock. The system in slow mode can get into normal mode and power down mode. To set STPHX = 1 to stop the external high-speed oscillator, and then the system consumes less power.

POWER DOWN (SLEEP) MODE

The power down mode is also called sleep mode. The chip stops working as sleeping status. The power consumption is very less almost to zero. The power down mode is usually applied to low power consuming system as battery power productions. To set CUPM0 = 1, the system gets into power down mode. The external high-speed and low-speed oscillators are turned off. The system can be waked up by P0 (P0.0, P0.1, P0.2) wakeup trigger signal.

Note:

- ***Watch_Dog code option = "Enable"***
  - ***Stop in power down (sleep) mode.***
  - ***Enable in normal mode mode.***
- ***Watch_Dog code option = "Always_ON".***
  - ***Enable in normal mode and power down (sleep) mode.***

SYSTEM MODE CONTROL

SN8P2710 SYSTEM MODE BLOCK DIAGRAM

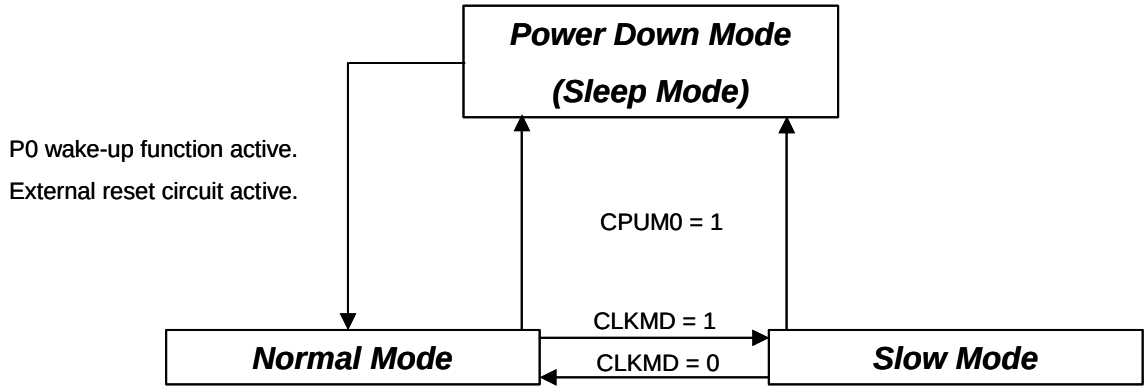


Figure 7-6. SN8P2710 System Mode Block Diagram

MODE	NORMAL	SLOW	POWER DOWN (SLEEP)	REMARK
HX osc.	Running	By STPHX	Stop	
LX osc.	Running	Running	Stop	
CPU instruction	Executing	Executing	Stop	
TC0/TC1	*Active	*Active	Inactive	* Active by program
Watchdog timer	Active	Active	By Watchdog code option	
Internal interrupt	All active	All active	All inactive	
External interrupt	All active	All active	All inactive	
Wakeup source	-	-	P0, Reset by RST, LVD, *Watchdog	* Watchdog code option must be “Always_ON”

Table 7-1. Operating Mode Description

SYSTEM MODE SWITCHING

Switch normal/slow mode to power down (sleep) mode.
CPUM0 = 1

```
        B0BSET      FCPUM0          ; Set CPUM0 = 1.
```

During the sleep, only the wakeup pin and reset can wakeup the system back to the normal mode.

Switch normal mode to slow mode.

```
        B0BSET      FCLKMD          ;To set CLKMD = 1, Change the system into slow mode
        B0BSET      FSTPHX          ;To stop external high-speed oscillator for power saving.
```

- ***Note: To stop high-speed oscillator is not necessary and user can omit it.***

Switch slow mode to normal mode (The external high-speed oscillator is still running)

```
        B0BCLR      FCLKMD          ;To set CLKMD = 0
```

Switch slow mode to normal mode (The external high-speed oscillator stops)
If external high clock stop and program want to switch back normal mode. It is necessary to delay at least 10mS for external clock stable.

```
        B0BCLR      FSTPHX          ; Turn on the external high-speed oscillator.

        B0MOV       Z, #27          ; If VDD = 5V, internal RC=32KHz (typical) will delay
@@:     DECMS       Z               ; 0.125ms X 81 = 10.125ms for external clock stable
        JMP         @B
                                    ;
        B0BCLR      FCLKMD          ; Change the system back to the normal mode
```

WAKEUP TIME

OVERVIEW

The external high-speed oscillator needs a delay time from stopping to operating. The delay is very necessary and makes the oscillator to work stably. Some conditions during system operating, the external high-speed oscillator often runs and stops. Under these conditions, the delay time for external high-speed oscillator restart is called wakeup time.

There are two conditions need wakeup time. One is power down mode to normal mode. The other one is slow mode to normal mode. For the first case, SN8P2710 provides 4096 oscillator clocks to be the wakeup time. However, in the last case, users need to make the wakeup time by themselves.

HARDWARE WAKEUP

When the system is in power down mode (sleep mode), the external high-speed oscillator stops. For wakeup into normal, SN8P2710 provides 4096 external high-speed oscillator clocks to be the wakeup time for warming up the oscillator circuit. After the wakeup time, the system goes into the normal mode. The value of the wakeup time is as following.

The Wakeup time = 1/Fosc * 3584 (sec) + X'tal settling time

The x'tal settling time is depended on the x'tal type. Typically, it is about 2~4mS.

➲ **Example: In power down mode (sleep mode), the system is waked up by P0 trigger signal. After the wakeup time, the system goes into normal mode. The wakeup time of P0 wakeup function is as following.**

*The wakeup time = 1/Fosc * 3584 = 1.001 ms (Fosc = 3.58MHz)*

The total wakeup time = 1.001 ms + x'tal settling time

Under power down mode (sleep mode), there are only I/O ports with wakeup function making the system to return normal mode. Port 0.0 and Port 0.1 wakeup function always enables.

8. TIMERS COUNTERS

WATCHDOG TIMER (WDT)

This built-in WDT watchdog 4-bit binary up counter designed for monitoring program execution. If the program is operated into the unknown status by noise interference or program dead lock, WDT's overflow signal will reset this chip to restart operation. In normal operation flow, the user must clear watchdog timer before overflow occurs to prevent the program from unexpected system reset. The clock source of watchdog timer (WDT) always comes from internal low speed RC oscillator. The overflow time of WDT is about:

1 / (16K ÷ 512 ÷ 16) ~ 0.5s @ 3V
1 / (32K ÷ 512 ÷ 16) ~ 0.25s @ 5V

0CCH	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
WDTR	WDTR7	WDTR6	WDTR5	WDTR4	WDTR3	WDTR2	WDTR1	WDTR0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
After reset	-	-	-	-	-	-	-	-

Write WDTR with "0x5A" to clear the Watchdog Timer.

Note:
- ***The watchdog timer can be set "Always_ON","Enable" and "Disabled" at the code option.***
- ***Watch_Dog code option = "Enable"***
  - ***Stop in power down (sleep) mode.***
  - ***Enable in normal mode.***
- ***Watch_Dog code option = "Always_ON".***
  - ***Enable in normal mode and power down (sleep) mode.***

➲ **Example: An operation of watchdog timer is as following. To clear the watchdog timer counter in the top of the main routine of the program.**

```
Main:
        MOV     A,#0x5A
        MOV     WDTR,A          ; Clear the watchdog timer counter.
        .       .
        CALL    SUB1
        CALL    SUB2
        .       .
        .       .
        .       .
        JMP     MAIN
```

- ***Note: Please use "@RST_WDT" macro to clear the watchdog timer successfully both in S8KD-2 ICE emulation and real chip.***

TIMER COUNTER 0 (TC0)

OVERVIEW

The TC0 is an 8-bit binary up timer and event counter for general-purpose timer, buzzer and PWM output. TC0 has a auto re-loadable counter that consists of two parts: an 8-bit reload register (TC0R) into which you write the counter reference value, and an 8-bit counter register (TC0C) whose value is automatically incremented by counter logic.

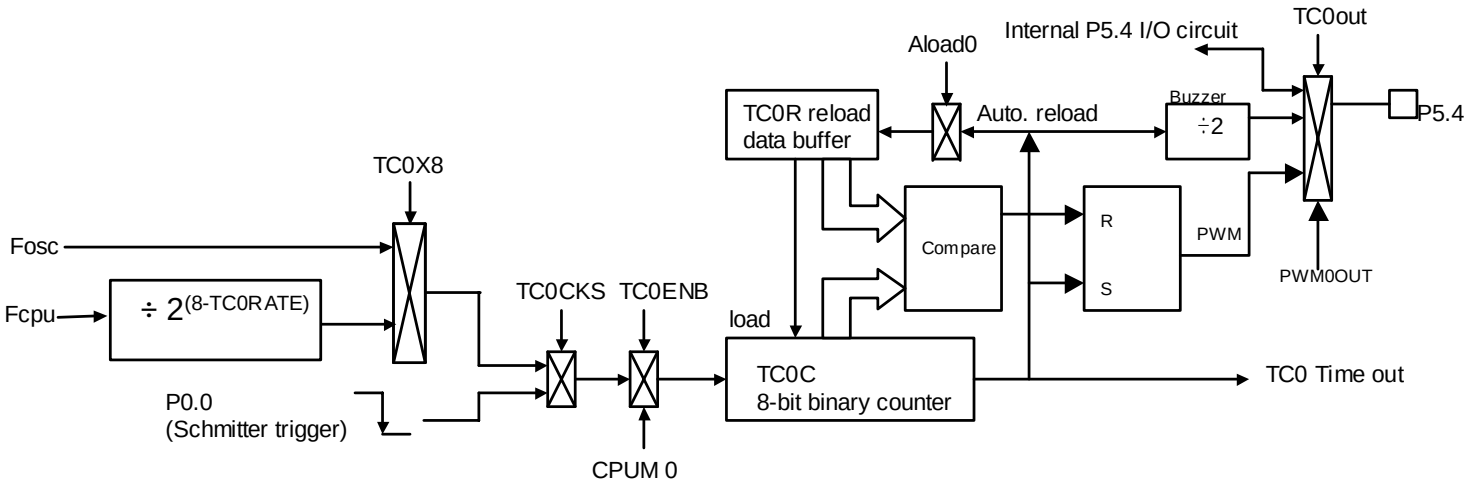


Figure 8-1. Timer Count TC0 Block Diagram

The main purposes of the TC0 timer counter is as following.

- **8-bit programmable timer:** Generates interrupts at specific time intervals based on the selected clock frequency.
- **Arbitrary frequency output (Buzzer output):** Outputs selectable clock frequencies to the BZ0 pin (P5.4).
- **PWM function:** PWM output can be generated by the PWM1OUT bit and output to PWM0OUT pin (P5.4).

TC0M MODE REGISTER

The TC0M is the timer counter mode register, which is an 8-bit read/write register. By loading different value into the TC0M register, users can modify the timer counter clock frequency dynamically when program executing.

Eight rates for TC0 timer can be selected by TC0RATE0 ~ TC0RATE2 and TC0X8 bits. If TC0X8=1 the TC0 clock will come from Fosc and the range is from Fosc/2 to Fosc/256, if TC0X8=0 (Initial), the range is from Fcpu/2 to Fcpu/256. The TC0M initial value is zero and the rate is Fcpu/256. The bit7 of TC0M named TC0ENB is the control bit to start TC0 timer. The combination of these bits is to determine the TC0 timer frequency.

T0M initial value = xxxx 00xx

0D8H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
T0M	-	-	-	-	TC1X8	TC0X8		-
	-	-	-	-	R/W	R/W		-

Bit3 **TC1X8**: Multiple TC1 timer speed eight times. Refer TC1M register for detailed information.
0 = TC1 clock came from Fcpu
1 = TC1 clock came from Fosc

Bit2 **TC0X8**: Multiple TC0 timer speed eight times. Refer TC0M register for detailed information.
0 = TC0 clock came from Fcpu
1 = TC0 clock came from Fosc

✸ ***Note: Under TC0 event counter mode (TC0CKS=1), TC0X8 bit and TC0RATE are useless.***

TC0M initial value = 0000 0000

0DAH	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
TC0M	TC0ENB	TC0RATE2	TC0RATE1	TC0RATE0	TC0CKS	ALOAD0	TC0OUT	PWM0OUT
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Bit 7 **TC0ENB:** TC0 counter enable bit.
0 = disable
1 = enable

Bit [6:4] **TC0RATE [2:0]:** TC0 internal clock rate select bits. Only for TC0CKS = 0

TC0Rate	TC0X8=0	TC0X8=1
111	Fcpu/2	Fosc/2
110	Fcpu/4	Fosc/4
101	Fcpu/8	Fosc/8
100	Fcpu/16	Fosc/16
011	Fcpu/32	Fosc/32
010	Fcpu/64	Fosc/64
001	Fcpu/128	Fosc/128
000	Fcpu/256	Fosc/256

Bit 3 **TC0CKS:** TC0 clock source select bit.
0 = Internal clock source (Fcpu or Fosc)
1 = External clock source input from P0.0 (INT0) pin.

Bit 2 **ALOAD0:** Auto-reload control bit.
0 = None auto-reload
1 = Auto-reload.

Bit 1 **TC0OUT:** TC0 time-out toggle signal output control bit. **Only valid when PWM0OUT = 0**
0 = Disable TC0OUT signal output and enable P5.4's I/O function,
1 = Enable TC0OUT signal output and disable P5.4's I/O function.

Bit 0 **PWM0OUT:** PWM output control bit. Refer “PWM Function Description” section for detailed information.
0 = Disable the PWM output
1 = Enable the PWM output (Auto-disable the TC0OUT function.)

PWM0OUT = 1, TC0X8=0

ALOAD0	TC0OUT	TC0 Overflow boundary	PWM duty range	Max PWM Frequency (Fosc = 16M) (Fcpu = 4M)	Note
0	0	FFh to 00h	0/256 ~ 255/256	7.8125K	Overflow per 256 count
0	1	3Fh to 40h	0/64 ~ 63/64	31.25K	Overflow per 64 count
1	0	1Fh to 20h	0/32 ~ 31/32	62.5K	Overflow per 32 count
1	1	0Fh to 10h	0/16 ~ 15/16	125K	Overflow per 16 count

PWM0OUT = 1, TC0X8=1

ALOAD0	TC0OUT	TC0 Overflow boundary	PWM duty range	Max PWM Frequency (Fosc = 16M) (Fcpu = 4M)	Note
0	0	FFh to 00h	0/256 ~ 255/256	62.5K	Overflow per 256 count
0	1	3Fh to 40h	0/64 ~ 63/64	250K	Overflow per 64 count
1	0	1Fh to 20h	0/32 ~ 31/32	500K	Overflow per 32 count
1	1	0Fh to 10h	0/16 ~ 15/16	1000K	Overflow per 16 count

✸ ***Note: When TC0CKS=1, TC0 became an external event counter and TC0RATE is useless. No more P0.0 interrupt request will be raised. (P0.0IRQ will be always 0).***

TC0C COUNTING REGISTER

TC0C is an 8-bit counter register for the timer counter (TC0). TC0C must be reset whenever the TC0ENB is set "1" to start the timer counter. TC0C is incremented by one with a clock pulse which the frequency is determined by TC0RATE0 ~ TC0RATE2. When TC0C has incremented to "0FFH", it is will be cleared to "00H" in next clock and an overflow is generated. Under TC0 interrupt service request (TC0IEN) enable condition, the TC0 interrupt request flag will be set "1" and the system executes the interrupt service routine.

TC0C initial value = xxxx xxxx

0DBH	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
TC0C	TC0C7	TC0C6	TC0C5	TC0C4	TC0C3	TC0C2	TC0C1	TC0C0
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

The equation of TC0C initial value is as following.

TC0C initial value = 256 - (TC0 interrupt interval time * input clock)

➲ **Example: To set 10ms interval time for TC0 interrupt at 3.58MHz high-speed mode. TC0C value (74H) = 256 - (10ms * fcpu/256)**

$$\begin{aligned} \textit{TC0C initial value} &= 256 - (\textit{TC0 interrupt interval time} * \textit{input clock}) \\ &= 256 - (10ms * 3.58 * 10^6 / 256) \\ &= 256 - (10^{-2} * 3.58 * 10^6 / 256) \\ &= 116 \\ &= 74H \end{aligned}$$

TC0_Counter=8-bit, TC0X8=0

TC0RATE	TC0CLOCK	High speed mode (fcpu = 3.58MHz / 4)		Low speed mode (fcpu = 32768Hz / 4)	
		Max overflow interval	One step = max/256	Max overflow interval	One step = max/256
000	fcpu/256	73.2 ms	286us	8000 ms	31.25 ms
001	fcpu/128	36.6 ms	143us	4000 ms	15.63 ms
010	fcpu/64	18.3 ms	71.5us	2000 ms	7.8 ms
011	fcpu/32	9.15 ms	35.8us	1000 ms	3.9 ms
100	fcpu/16	4.57 ms	17.9us	500 ms	1.95 ms
101	fcpu/8	2.28 ms	8.94us	250 ms	0.98 ms
110	fcpu/4	1.14 ms	4.47us	125 ms	0.49 ms
111	fcpu/2	0.57 ms	2.23us	62.5 ms	0.24 ms

TC0_Counter=6-bit , TC0X8=0

TC0RATE	TC0CLOCK	High speed mode (fcpu = 3.58MHz / 4)		Low speed mode (fcpu = 32768Hz / 4)	
		Max overflow interval	One step = max/256	Max overflow interval	One step = max/256
000	fcpu/256	18.3 ms	71.5us	2000 ms	7.8 ms
001	fcpu/128	9.15 ms	35.8us	1000 ms	3.9 ms
010	fcpu/64	4.57 ms	17.9us	500 ms	1.95 ms
011	fcpu/32	2.28 ms	8.94us	250 ms	0.98 ms
100	fcpu/16	1.14 ms	4.47us	125 ms	0.49 ms
101	fcpu/8	0.57 ms	2.23us	62.5 ms	0.24 ms
110	fcpu/4	0.285 ms	1.11us	31.25 ms	0.12 ms
111	fcpu/2	0.143 ms	0.56 us	15.63 ms	0.06 ms

TC0_Counter=5-bit, TC0X8=0

TC0RATE	TC0CLOCK	High speed mode (fcpu = 3.58MHz / 4)		Low speed mode (fcpu = 32768Hz / 4)	
		Max overflow interval	One step = max/256	Max overflow interval	One step = max/256
000	fcpu/256	9.15 ms	35.8us	1000 ms	3.9 ms
001	fcpu/128	4.57 ms	17.9us	500 ms	1.95 ms
010	fcpu/64	2.28 ms	8.94us	250 ms	0.98 ms
011	fcpu/32	1.14 ms	4.47us	125 ms	0.49 ms
100	fcpu/16	0.57 ms	2.23us	62.5 ms	0.24 ms
101	fcpu/8	0.285 ms	1.11us	31.25 ms	0.12 ms
110	fcpu/4	0.143 ms	0.56 us	15.63 ms	0.06 ms
111	fcpu/2	71.25 us	0.278 us	7.81 ms	0.03 ms

TC0_Counter=4-bit, TC0X8=0

TC0RATE	TC0CLOCK	High speed mode (fcpu = 3.58MHz / 4)		Low speed mode (fcpu = 32768Hz / 4)	
		Max overflow interval	One step = max/256	Max overflow interval	One step = max/256
000	fcpu/256	4.57 ms	17.9us	500 ms	1.95 ms
001	fcpu/128	2.28 ms	8.94us	250 ms	0.98 ms
010	fcpu/64	1.14 ms	4.47us	125 ms	0.49 ms
011	fcpu/32	0.57 ms	2.23us	62.5 ms	0.24 ms
100	fcpu/16	0.285 ms	1.11us	31.25 ms	0.12 ms
101	fcpu/8	0.143 ms	0.56 us	15.63 ms	0.06 ms
110	fcpu/4	71.25 us	0.278 us	7.81 ms	0.03 ms
111	fcpu/2	35.63 us	0.139 us	3.91 ms	0.015 ms

Table 8-1. The Timing Table of Timer Count TC0

TC0R AUTO-LOAD REGISTER

TC0R is an 8-bit register for the TC0 auto-reload function. TC0R's value applies to TC0OUT and PWM0OUT functions.. Under TC0OUT application, users must enable and set the TC0R register. The main purpose of TC0R is as following.

- Store the auto-reload value and set into TC0C when the TC0C overflow. (ALOAD0 = 1).
- Store the duty value of PWM0OUT function.

TC0R initial value = xxxx xxxx

0CDH	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
TC0R	TC0R7	TC0R6	TC0R5	TC0R4	TC0R3	TC0R2	TC0R1	TC0R0
	W	W	W	W	W	W	W	W

The equation of TC0R initial value is like TC0C as following.

TC0R initial value = 256 - (TC0 interrupt interval time * input clock)

Note: The TC0R is write-only register can't be process by INCMS, DECMS instructions.

TC0 TIMER COUNTER OPERATION SEQUENCE

The TC0 timer counter's sequence of operation can be following.

- Set the TC0C initial value to setup the interval time.
- Set the TC0ENB to be "1" to enable TC0 timer counter.
- TC0C is incremented by one with each clock pulse which frequency is corresponding to T0M selection.
- TC0C overflow when TC0C from FFH to 00H.
- When TC0C overflow occur, the TC0IRQ flag is set to be "1" by hardware.
- Execute the interrupt service routine.
- Users reset the TC0C value and resume the TC0 timer operation.

➲ Example: Setup the TC0M and TC0C without auto-reload function.

```
B0BCLR      FTC0IEN     ; To disable TC0 interrupt service
B0BCLR      FTC0ENB     ; To disable TC0 timer
MOV         A,#00H      ;
B0MOV       TC0M,A      ; To set TC0 clock = fcpu / 256
MOV         A,#74H      ; To set TC0C initial value = 74H
B0MOV       TC0C,A      ;(To set TC0 interval = 10 ms)

B0BSET      FTC0IEN     ; To enable TC0 interrupt service
B0BCLR      FTC0IRQ     ; To clear TC0 interrupt request
B0BSET      FTC0ENB     ; To enable TC0 timer
```

➲ Example: Setup the TC0M and TC0C with auto-reload function.

```
B0BCLR      FTC0IEN     ; To disable TC0 interrupt service
B0BCLR      FTC0ENB     ; To disable TC0 timer
MOV         A,#00H      ;
B0MOV       TC0M,A      ; To set TC0 clock = fcpu / 256
MOV         A,#74H      ; To set TC0C initial value = 74H
B0MOV       TC0C,A      ; (To set TC0 interval = 10 ms)
B0MOV       TC0R,A      ; To set TC0R auto-reload register

B0BSET      FTC0IEN     ; To enable TC0 interrupt service
B0BCLR      FTC0IRQ     ; To clear TC0 interrupt request
B0BSET      FTC0ENB     ; To enable TC0 timer
B0BSET      ALOAD0      ; To enable TC0 auto-reload function.
```

➲ **Example: TC0 interrupt service routine without auto-reload function.**

```
                ORG         8               ; Interrupt vector
                JMP         INT_SERVICE
INT_SERVICE:

                B0XCH       A, ACCBUF       ; B0XCH doesn't change C, Z flag
                B0MOV       A, PFLAG        ;
                B0MOV       PFLAGBUF, A     ; Save PFLAG register in a buffer                ; Save

                B0BTS1      FTC0IRQ         ; Check TC0IRQ
                JMP         EXIT_INT        ; TC0IRQ = 0, exit interrupt vector

                B0BCLR      FTC0IRQ         ; Reset TC0IRQ
                MOV         A,#74H          ; Reload TC0C
                B0MOV       TC0C,A
                .           .               ; TC0 interrupt service routine
                .           .
                JMP         EXIT_INT        ; End of TC0 interrupt service routine and exit interrupt
                                            vector
                .           .
                .           .
EXIT_INT:
                                            ;
                B0MOV       A, PFLAGBUF
                B0MOV       PFLAG, A        ; Restore PFLAG register from buffer
                B0XCH       A, ACCBUF       ; Restore ACC value.

                RETI                        ; Exit interrupt vector
```

➲ **Example: TC0 interrupt service routine with auto-reload.**

```
                ORG         8               ; Interrupt vector
                JMP         INT_SERVICE
INT_SERVICE:

                B0XCH       A, ACCBUF       ; B0XCH doesn't change C, Z flag
                B0MOV       A, PFLAG        ;
                B0MOV       PFLAGBUF, A     ; Save PFLAG register in a buffer


                B0BTS1      FTC0IRQ         ; Check TC0IRQ
                JMP         EXIT_INT        ; TC0IRQ = 0, exit interrupt vector

                B0BCLR      FTC0IRQ         ; Reset TC0IRQ
                .           .               ; TC0 interrupt service routine
                .           .
                JMP         EXIT_INT        ; End of TC0 interrupt service routine and exit interrupt
                                            vector
                .           .
                .           .
EXIT_INT:
                B0MOV       A, PFLAG        ;
                B0MOV       PFLAGBUF, A     ; Save PFLAG register in a buffer
                B0XCH       A, ACCBUF       ; Restore ACC value.

                RETI                        ; Exit interrupt vector
```

TC0 CLOCK FREQUENCY OUTPUT (BUZZER)

TC0 timer counter provides a frequency output function. By setting the TC0 clock frequency, the clock signal is output to P5.4 and the P5.4 general purpose I/O function is auto-disable. The TC0 output signal divides by 2. The TC0 clock has many combinations and easily to make difference frequency. This function applies as buzzer output to output multi-frequency.

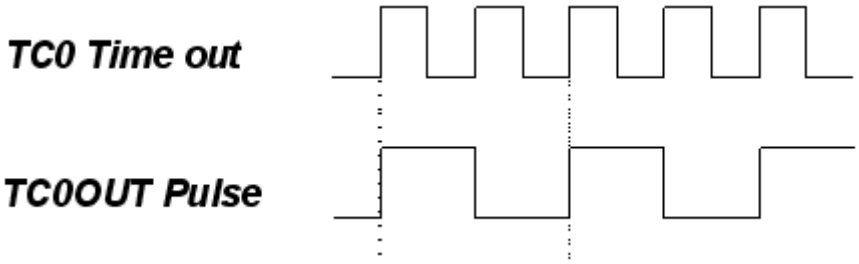


Figure 8-2. The TC0OUT Pulse Frequency

➲ **Example: Setup TC0OUT output from TC0 to TC0OUT (P5.4). The Fcpu is 4MHz. The TC0OUT frequency is 1KHz. Because the TC0OUT signal is divided by 2, set the TC0 clock to 2KHz. The TC0 clock source is from external oscillator clock. T0C rate is Fcpu/4. The TC0RATE2~TC0RATE1 = 110. TC0C = TC0R = 131, TC0X8=1.**

```
MOV     A,#01100000B
B0MOV   TC0M,A          ; Set the TC0 rate to Fcpu/4

MOV     A,#131          ; Set the auto-reload reference value
B0MOV   TC0C,A
B0MOV   TC0R,A
B0BCLR  FTC0X8

B0BSET  FTC0OUT         ; Enable TC0 output to P5.4 and disable P5.4 I/O function
B0BSET  FALOAD0         ; Enable TC0 auto-reload function
B0BSET  FTC0ENB         ; Enable TC0 timer
```

TIMER COUNTER 1 (TC1)

OVERVIEW

The timer counter 1 (TC1) is used to generate an interrupt request when a specified time interval has elapsed. TC1 has a auto re-loadable counter that consists of two parts: an 8-bit reload register (TC1R) into which you write the counter reference value, and an 8-bit counter register (TC1C) whose value is automatically incremented by counter logic.

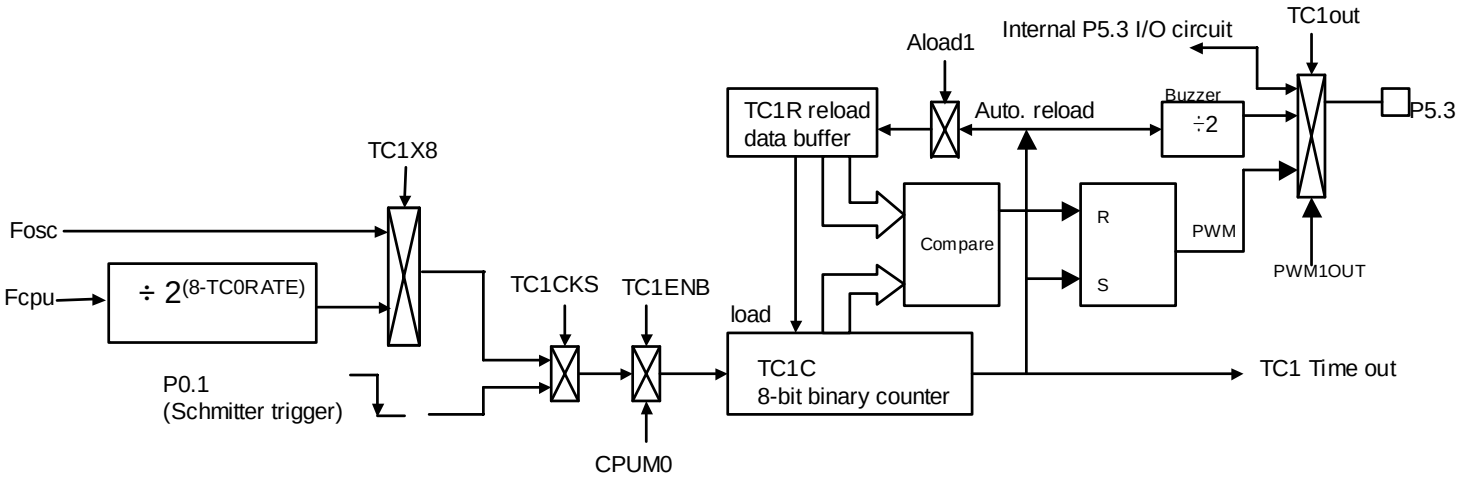


Figure 8-3. Timer Count TC1 Block Diagram

The main purposes of the TC1 timer is as following.

- **8-bit programmable timer:** Generates interrupts at specific time intervals based on the selected clock frequency.
- **Arbitrary frequency output (Buzzer output):** Outputs selectable clock frequencies to the BZ1 pin (P5.3).
- **PWM function:** PWM output can be generated by the PWM1OUT bit and output to PWM1OUT pin (P5.3).

TC1M MODE REGISTER

The TC1M is an 8-bit read/write timer mode register. By loading different value into the TC1M register, users can modify the timer clock frequency dynamically as program executing.

Eight rates for TC1 timer can be selected by TC1RATE0 ~ TC1RATE2 bits. If TC1X8=1 the TC1 clock will come from Fosc and the range is from Fosc/2 to Fosc/256. if TC1X8=0 (Initial), the range is from Fcpu(Fosc)/2 to Fcpu(Fosc)/256. The TC1M initial value is zero and the rate is Fcpu/256. The bit7 of TC1M called TC1ENB is the control bit to start TC1 timer. The combination of these bits is to determine the TC1 timer clock frequency and the intervals.

T0M initial value = xxxx 00xx

0D8H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
T0M	-	-	-	-	TC1X8	TC0X8		-
	-	-	-	-	R/W	R/W		-

Bit3 **TC1X8**: Multiple TC1 timer speed eight times. Refer TC1M register for detailed information.
0 = TC1 clock came from Fcpu
1 = TC1 clock came from Fosc

Bit2 **TC0X8**: Multiple TC0 timer speed eight times. Refer TC0M register for detailed information.
0 = TC0 clock came from Fcpu
1 = TC0 clock came from Fosc

✸ ***Note: Under TC1 event counter mode (TC1CKS=1), TC1X8 bit and TC1RATE are useless.***

TC1M initial value = 0000 0000

0DCH	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
TC1M	TC1ENB	TC1RATE2	TC1RATE1	TC1RATE0	TC1CKS	ALOAD1	TC1OUT	PWM1OUT
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Bit 7 **TC1ENB:** TC1 counter enable bit.
0 = disable
1 = enable

Bit [6:4] **TC1RATE [2:0]:** TC1 internal clock rate select bits. Only for TC1CKS = 0

TC1Rate	TC1X8=0	TC1X8=1
111	Fcpu/2	Fosc/2
110	Fcpu/4	Fosc/4
101	Fcpu/8	Fosc/8
100	Fcpu/16	Fosc/16
011	Fcpu/32	Fosc/32
010	Fcpu/64	Fosc/64
001	Fcpu/128	Fosc/128
000	Fcpu/256	Fosc/256

Bit 3 **TC1CKS:** TC1 clock source select bit.
0 = Internal clock source (Fcpu/Fosc)
1 = External clock source input from P0.1 (INT1) pin.

Bit 2 **ALOAD1:** Auto-reload control bit.
0 = none auto-reload
1 = auto-reload.

Bit 1 **TC1OUT:** TC1 time-out toggle signal output control bit. **Only valid when PWM1OUT = 0**
0 = Disable TC1OUT signal output and enable P5.3's I/O function,
1 = Enable TC1OUT signal output and disable P5.3's I/O function.

Bit 0 **PWM1OUT:** PWM output control bit. Refer “PWM Function Description” section for detailed information.
0 = Disable the PWM output
1 = Enable the PWM output (Auto-disable the TC1OUT function.)

PWM1OUT = 1, TC1X8=0

ALOAD1	TC1OUT	TC1 Overflow boundary	PWM duty range	Max PWM Frequency (Fosc = 16M) (Fcpu = 4M)	Note
0	0	FFh to 00h	0/256 ~ 255/256	7.8125K	Overflow per 256 count
0	1	3Fh to 40h	0/64 ~ 63/64	31.25K	Overflow per 64 count
1	0	1Fh to 20h	0/32 ~ 31/32	62.5K	Overflow per 32 count
1	1	0Fh to 10h	0/16 ~ 15/16	125K	Overflow per 16 count

PWM0OUT = 1, TC1X8=1

ALOAD0	TC0OUT	TC0 Overflow boundary	PWM duty range	Max PWM Frequency (Fosc = 16M) (Fcpu = 4M)	Note
0	0	FFh to 00h	0/256 ~ 255/256	62.5K	Overflow per 256 count
0	1	3Fh to 40h	0/64 ~ 63/64	250K	Overflow per 64 count
1	0	1Fh to 20h	0/32 ~ 31/32	500K	Overflow per 32 count
1	1	0Fh to 10h	0/16 ~ 15/16	1000K	Overflow per 16 count

✷ ***Note: When TC1CKS=1, TC1 became an external event counter and TC1RATE is useless. No more P0.1 interrupt request will be raised. (P0.1IRQ will be always 0).***

TC1C COUNTING REGISTER

TC1C is an 8-bit counter register for the timer counter (TC1). TC1C must be reset whenever the TC1ENB is set "1" to start the timer. TC1C is incremented by one with a clock pulse which the frequency is determined by TC1RATE0 ~ TC1RATE2. When TC1C has incremented to "0FFH", it is will be cleared to "00H" in next clock and an overflow is generated. Under TC1 interrupt service request (TC1IEN) enable condition, the TC1 interrupt request flag will be set "1" and the system executes the interrupt service routine.

TC1C initial value = xxxx xxxx

0DDH	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
TC1C	TC1C7	TC1C6	TC1C5	TC1C4	TC1C3	TC1C2	TC1C1	TC1C0
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

The interval time of TC1 basic timer table.

The equation of TC1C initial value is as following.

TC1C initial value = 256 - (TC1 interrupt interval time * input clock)

➲ **Example: To set 10ms interval time for TC1 interrupt at 3.58MHz high-speed mode. TC1C value (74H) = 256 - (10ms * fcpu/256)**

$$\begin{aligned} \textit{TC1C initial value} &= 256 - (\textit{TC1 interrupt interval time} * \textit{input clock}) \\ &= 256 - (10\text{ms} * 3.58 * 10^6 / 256) \\ &= 256 - (10^{-2} * 3.58 * 10^6 / 256) \\ &= 116 \\ &= 74\text{H} \end{aligned}$$

TC1_Counter=8-bit, TC1X8=0

TC1RATE	TC1CLOCK	High speed mode (fcpu = 3.58MHz / 4)		Low speed mode (fcpu = 32768Hz / 4)	
		Max overflow interval	One step = max/256	Max overflow interval	One step = max/256
000	fcpu/256	73.2 ms	286us	8000 ms	31.25 ms
001	fcpu/128	36.6 ms	143us	4000 ms	15.63 ms
010	fcpu/64	18.3 ms	71.5us	2000 ms	7.8 ms
011	fcpu/32	9.15 ms	35.8us	1000 ms	3.9 ms
100	fcpu/16	4.57 ms	17.9us	500 ms	1.95 ms
101	fcpu/8	2.28 ms	8.94us	250 ms	0.98 ms
110	fcpu/4	1.14 ms	4.47us	125 ms	0.49 ms
111	fcpu/2	0.57 ms	2.23us	62.5 ms	0.24 ms

TC1_Counter=6-bit, TC1X8=0

TC1RATE	TC1CLOCK	High speed mode (fcpu = 3.58MHz / 4)		Low speed mode (fcpu = 32768Hz / 4)	
		Max overflow interval	One step = max/256	Max overflow interval	One step = max/256
000	fcpu/256	18.3 ms	71.5us	2000 ms	7.8 ms
001	fcpu/128	9.15 ms	35.8us	1000 ms	3.9 ms
010	fcpu/64	4.57 ms	17.9us	500 ms	1.95 ms
011	fcpu/32	2.28 ms	8.94us	250 ms	0.98 ms
100	fcpu/16	1.14 ms	4.47us	125 ms	0.49 ms
101	fcpu/8	0.57 ms	2.23us	62.5 ms	0.24 ms
110	fcpu/4	0.285 ms	1.11us	31.25 ms	0.12 ms
111	fcpu/2	0.143 ms	0.56 us	15.63 ms	0.06 ms

TC1_Counter=5-bit, TC1X8=0

TC1RATE	TC1CLOCK	High speed mode (fcpu = 3.58MHz / 4)		Low speed mode (fcpu = 32768Hz / 4)	
		Max overflow interval	One step = max/256	Max overflow interval	One step = max/256
000	Fcpu/256	9.15 ms	35.8us	1000 ms	3.9 ms
001	Fcpu/128	4.57 ms	17.9us	500 ms	1.95 ms
010	fcpu/64	2.28 ms	8.94us	250 ms	0.98 ms
011	fcpu/32	1.14 ms	4.47us	125 ms	0.49 ms
100	fcpu/16	0.57 ms	2.23us	62.5 ms	0.24 ms
101	fcpu/8	0.285 ms	1.11us	31.25 ms	0.12 ms
110	fcpu/4	0.143 ms	0.56 us	15.63 ms	0.06 ms
111	fcpu/2	71.25 us	0.278 us	7.81 ms	0.03 ms

TC1_Counter=4-bit, TC1X8=0

TC1RATE	TC1CLOCK	High speed mode (fcpu = 3.58MHz / 4)		Low speed mode (fcpu = 32768Hz / 4)	
		Max overflow interval	One step = max/256	Max overflow interval	One step = max/256
000	Fcpu/256	4.57 ms	17.9us	500 ms	1.95 ms
001	Fcpu/128	2.28 ms	8.94us	250 ms	0.98 ms
010	fcpu/64	1.14 ms	4.47us	125 ms	0.49 ms
011	fcpu/32	0.57 ms	2.23us	62.5 ms	0.24 ms
100	fcpu/16	0.285 ms	1.11us	31.25 ms	0.12 ms
101	fcpu/8	0.143 ms	0.56 us	15.63 ms	0.06 ms
110	fcpu/4	71.25 us	0.278 us	7.81 ms	0.03 ms
111	fcpu/2	35.63 us	0.139 us	3.91 ms	0.015 ms

Table 8-2. The Timing Table of Timer Count TC1

TC1R AUTO-LOAD REGISTER

TC1R is an 8-bit register for the TC1 auto-reload function. TC1R's value applies to TC1OUT and PWM1OUT functions. Under TC1OUT application, users must enable and set the TC1R register. The main purpose of TC1R is as following.

- Store the auto-reload value and set into TC1C when the TC1C overflow. (ALOAD1 = 1).
- Store the duty value of PWM1OUT function.

TC1R initial value = xxxx xxxx

0DEH	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
TC1R	TC1R7	TC1R6	TC1R5	TC1R4	TC1R3	TC1R2	TC1R1	TC1R0
	W	W	W	W	W	W	W	W

The equation of TC1R initial value is like TC1C as following.

TC1R initial value = 256 - (TC1 interrupt interval time * input clock)

Note: The TC1R is write-only register can't be process by INCMS, DECMS instructions.

TC1 TIMER COUNTER OPERATION SEQUENCE

The TC1 timer's sequence of operation can be following.

- Set the TC1C initial value to setup the interval time.
- Set the TC1ENB to be "1" to enable TC1 timer counter.
- TC1C is incremented by one with each clock pulse which frequency is corresponding to TC1M selection.
- TC1C overflow if TC1C from FFH to 00H.
- When TC1C overflow occur, the TC1IRQ flag is set to be "1" by hardware.
- Execute the interrupt service routine.
- Users reset the TC1C value and resume the TC1 timer operation.

➲ Example: Setup the TC1M and TC1C without auto-reload function.

```
B0BCLR      FTC1IEN     ; To disable TC1 interrupt service
B0BCLR      FTC1ENB     ; To disable TC1 timer
B0BCLR      FTC1X8      ;
MOV         A,#00H      ;
B0MOV       TC1M,A      ; To set TC1 clock = fcpu / 256
MOV         A,#74H      ; To set TC1C initial value = 74H
B0MOV       TC1C,A      ;(To set TC1 interval = 10 ms)

B0BSET      FTC1IEN     ; To enable TC1 interrupt service
B0BCLR      FTC1IRQ     ; To clear TC1 interrupt request
B0BSET      FTC1ENB     ; To enable TC1 timer
```

➲ Example: Setup the TC1M and TC1C with auto-reload function.

```
B0BCLR      FTC1IEN     ; To disable TC1 interrupt service
B0BCLR      FTC1ENB     ; To disable TC1 timer
B0BCLR      FTC1X8      ;
MOV         A,#00H      ;
B0MOV       TC1M,A      ; To set TC1 clock = fcpu / 256
MOV         A,#74H      ; To set TC1C initial value = 74H
B0MOV       TC1C,A      ; (To set TC1 interval = 10 ms)
B0MOV       TC1R,A      ; To set TC1R auto-reload register

B0BSET      FTC1IEN     ; To enable TC1 interrupt service
B0BCLR      FTC1IRQ     ; To clear TC1 interrupt request
B0BSET      FTC1ENB     ; To enable TC1 timer
B0BSET      ALOAD1      ; To enable TC1 auto-reload function.
```

➲ **Example: TC1 interrupt service routine without auto-reload function.**

```
            ORG     8               ; Interrupt vector
            JMP     INT_SERVICE
INT_SERVICE:

            B0XCH   A, ACCBUF       ; B0XCH doesn't change C, Z flag
            B0MOV   A, PFLAG
            B0MOV   PFLAGBUF, A     ; Save PFLAG register in a buffer

            B0BTS1  FTC1IRQ         ; Check TC1IRQ
            JMP     EXIT_INT        ; TC1IRQ = 0, exit interrupt vector

            B0BCLR  FTC1IRQ         ; Reset TC1IRQ
            MOV     A,#74H          ; Reload TC1C
            B0MOV   TC1C,A
            .       .               ; TC1 interrupt service routine
            .       .
            JMP     EXIT_INT        ; End of TC1 interrupt service routine and exit interrupt
                                    vector
            .       .
            .       .
EXIT_INT:
            B0MOV   A, PFLAGBUF
            B0MOV   PFLAG, A        ; Restore PFLAG register from buffer
            B0XCH   A, ACCBUF       ; Restore ACC value.

            RETI                    ; Exit interrupt vector
```

➲ **Example: TC1 interrupt service routine with auto-reload.**

```
            ORG     8               ; Interrupt vector
            JMP     INT_SERVICE
INT_SERVICE:

            B0XCH   A, ACCBUF       ; B0XCH doesn't change C, Z flag
            B0MOV   A, PFLAG
            B0MOV   PFLAGBUF, A     ; Save PFLAG register in a buffer

            B0BTS1  FTC1IRQ         ; Check TC1IRQ
            JMP     EXIT_INT        ; TC1IRQ = 0, exit interrupt vector

            B0BCLR  FTC1IRQ         ; Reset TC1IRQ
            .       .               ; TC1 interrupt service routine
            .       .
            JMP     EXIT_INT        ; End of TC1 interrupt service routine and exit interrupt
                                    vector
            .       .
            .       .
EXIT_INT:
            B0MOV   A, PFLAGBUF
            B0MOV   PFLAG, A        ; Restore PFLAG register from buffer
            B0XCH   A, ACCBUF       ; Restore ACC value.

            RETI                    ; Exit interrupt vector
```

TC1 CLOCK FREQUENCY OUTPUT (BUZZER)

TC1 timer counter provides a frequency output function. By setting the TC1 clock frequency, the clock signal is output to P5.3 and the P5.3 general purpose I/O function is auto-disable. The TC1 output signal divides by 2. The TC1 clock has many combinations and easily to make difference frequency. This function applies as buzzer output to output multi-frequency.

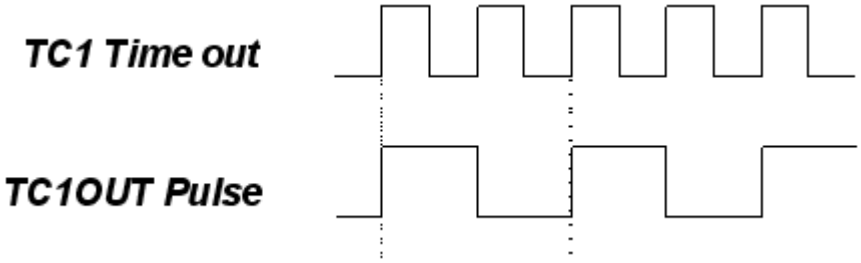


Figure 8-4. The TC1OUT Pulse Frequency

➲ **Example: Setup TC1OUT output from TC1 to TC1OUT (P5.3). The Fcpu is 4MHz. The TC1OUT frequency is 1KHz. Because the TC1OUT signal is divided by 2, set the TC1 clock to 2KHz. The TC1 clock source is from external oscillator clock. TC1 rate is Fcpu/4. The TC1RATE2~TC1RATE1 = 110. TC1X8=1, TC1C = TC1R = 131.**

```
B0BCLR      FTC1X8          ;
MOV         A,#01100000B
B0MOV       TC1M,A          ; Set the TC1 rate to Fcpu/4

MOV         A,#131          ; Set the auto-reload reference value
B0MOV       TC1C,A
B0MOV       TC1R,A

B0BSET      FTC1OUT         ; Enable TC1 output to P5.3 and disable P5.3 I/O function
B0BSET      FALOAD1         ; Enable TC1 auto-reload function
B0BSET      FTC1ENB         ; Enable TC1 timer
```

➢ ***Note: The TC1OUT frequency table is as TC0OUT frequency table. Please consult TC0OUT frequency table. (Table 7-2~7-5)***

PWM FUNCTION DESCRIPTION

OVERVIEW

PWM function is generated by TC0/TC1 timer counter and output the PWM signal to PWM0OUT pin (P5.4)/ PWM1OUT pin (P5.3). The 8-bit counter counts modulus 256, from 0-255, inclusive. The value of the 8-bit counter is compared to the contents of the reference register TC0R/TC1R. When the reference register value (TC0R/TC1R) is equal to the counter value TC0C/TC1C, the PWM output goes low. When the counter reaches zero, the PWM output is forced high.

Initial PWM output level is low until the counter value cross boundary (e.g. TC0C changes from FFH back to 00H), the PWM outputs are forced to high level. The pulse width ratio (duty cycle) is defined by TC0R/TC1R registers and the overflow boundary is defined by ALOAD0/ALOAD1 and TC0OUT/TC1OUT bits. PWM output can be held at low level by continuously loading the TC0R/TC1R with 00H. By continuously loading the TC0R with boundary value (e.g. FFH), you can hold the PWM output to high level, except for the last pulse of the clock source, which sends the output low.

PWM0OUT = 1, TC0X8=0

ALOAD0 ALOAD1	TC0OUT TC1OUT	TC0 Overflow boundary TC1 Overflow boundary	PWM duty range	Max PWM Frequency (Fcpu = 4M)	Note
0	0	FFh to 00h	0/256 ~ 255/256	7.8125K	Overflow per 256 count
0	1	3Fh to 40h	0/64 ~ 63/64	31.25K	Overflow per 64 count
1	0	1Fh to 20h	0/32 ~ 31/32	62.5K	Overflow per 32 count
1	1	0Fh to 10h	0/16 ~ 15/16	125K	Overflow per 16 count

ALOAD0 ALOAD1	TC0OUT TC1OUT	TC0R TC1R	PWM Duty Range
0	0	00000000 to 11111111	0/256 to 255/256
0	1	XX000000 to XX111111	0/64 to 63/64
1	0	XXX00000 to XXX11111	0/32 to 31/32
1	1	XXXX0000 to XXXX1111	0/16 to 15/16

Table 8-2. The PWM Duty Cycle Table

NOTE: If PWM0OUT or TC0OUT is enabled, P5.4 mode will be forced as output mode automatically. When PWM0OUT or TC0OUT is disabled, P5.4 mode will be defined by P54M bit.

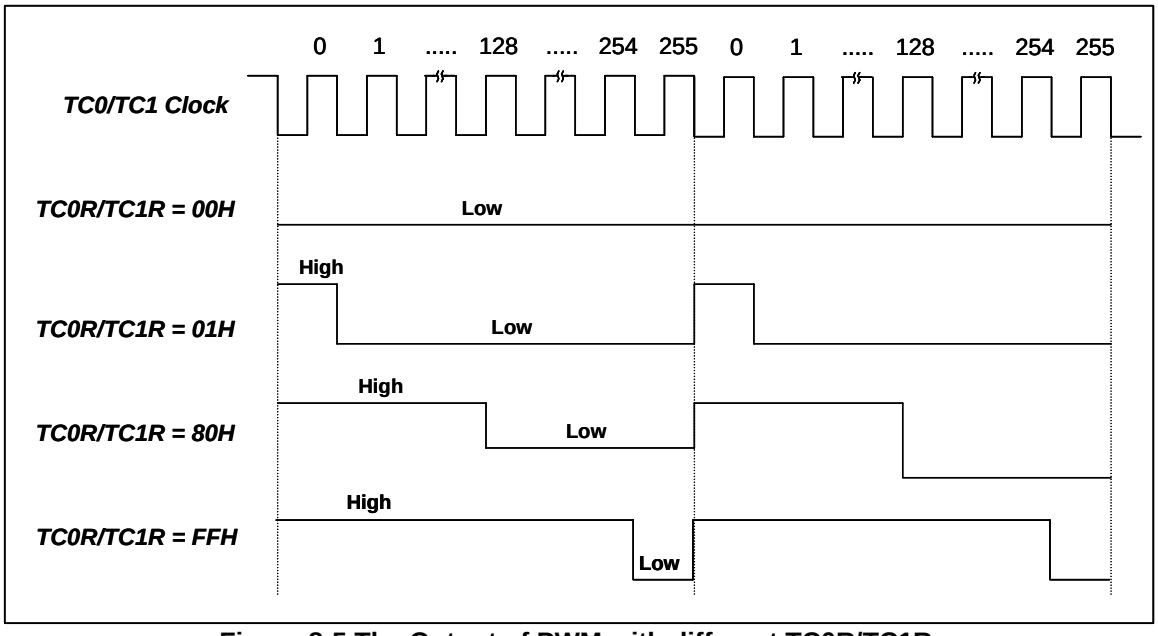


Figure 8-5 The Output of PWM with different TC0R/TC1R.

PWM PROGRAM DESCRIPTION

➲ **Example: Setup PWM0 output from TC0 to PWM0OUT (P5.4). The Fcpu is 4MHz. The duty of PWM is 30/256. The PWM frequency is about 1KHz. The PWM clock source is from external oscillator clock. TC0 rate is Fcpu/4. The TC0RATE2~TC0RATE1 = 110. TC0C = TC0R = 30.**

```
B0BCLR      FTC0X8          ;
B0BCLR      FTC1X8          ;
MOV         A,#01100000B
B0MOV       TC0M,A          ; Set the TC0 rate to Fcpu/4
B0MOV       TC0M,A          ; Set the TC0 rate to Fcpu/4
MOV         A,#0x00         ;First Time Initial TC0

MOV         A,#30           ; Set the PWM duty to 30/256
B0MOV       TC0R,A

B0BCLR      FTC0OUT         ; Disable TC0OUT function.
B0BSET      FPWM0OUT        ; Enable PWM0 output to P5.4 and disable P5.4 I/O function
B0BSET      FTC0ENB         ; Enable TC0 timer
```

- ***Note1: The TC0R and TC1R are write-only registers. Don't process them using INCMS, DECMS instructions.***
- ***Note2: Set TC0C at initial is to make first duty-cycle correct.***

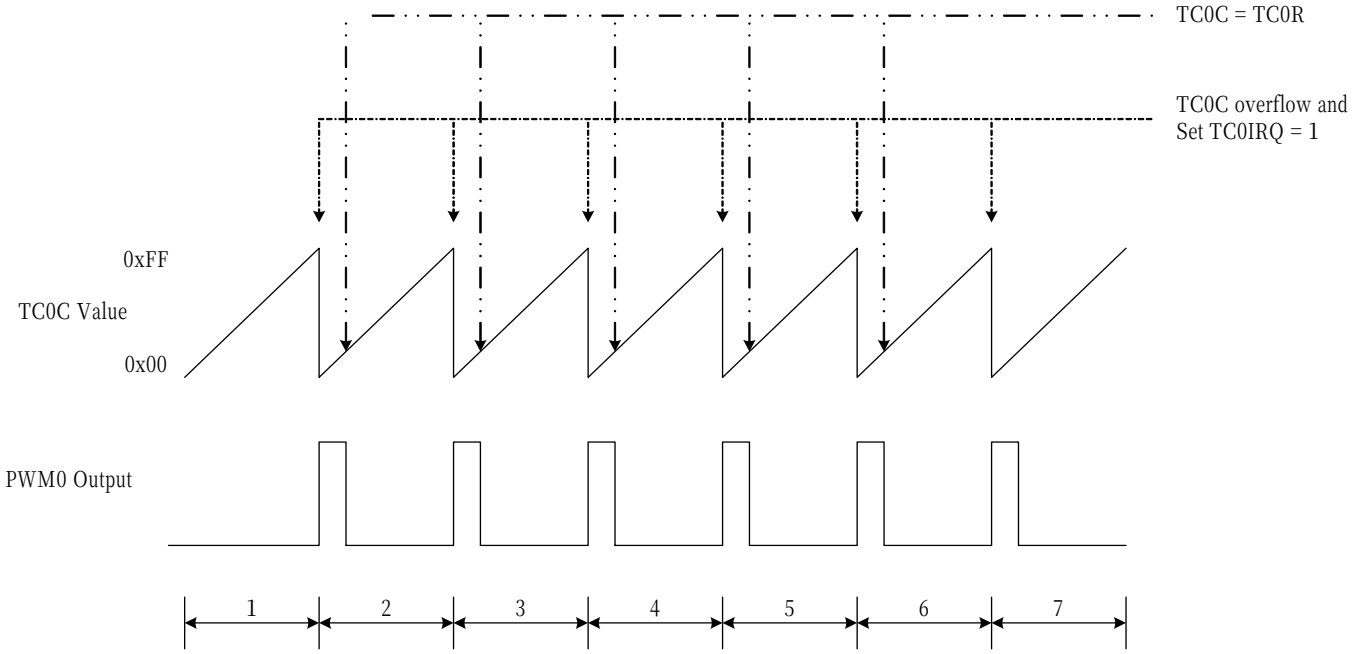
➲ **Example: Modify TC0R/TC1R registers' value.**

```
MOV         A, #30H         ; Input a number using B0MOV instruction.
B0MOV       TC0R, A

INCMS       BUF0            ; Get the new TC0R value from the BUF0 buffer defined by
B0MOV       A, BUF0         ; programming.
B0MOV       TC0R, A
```

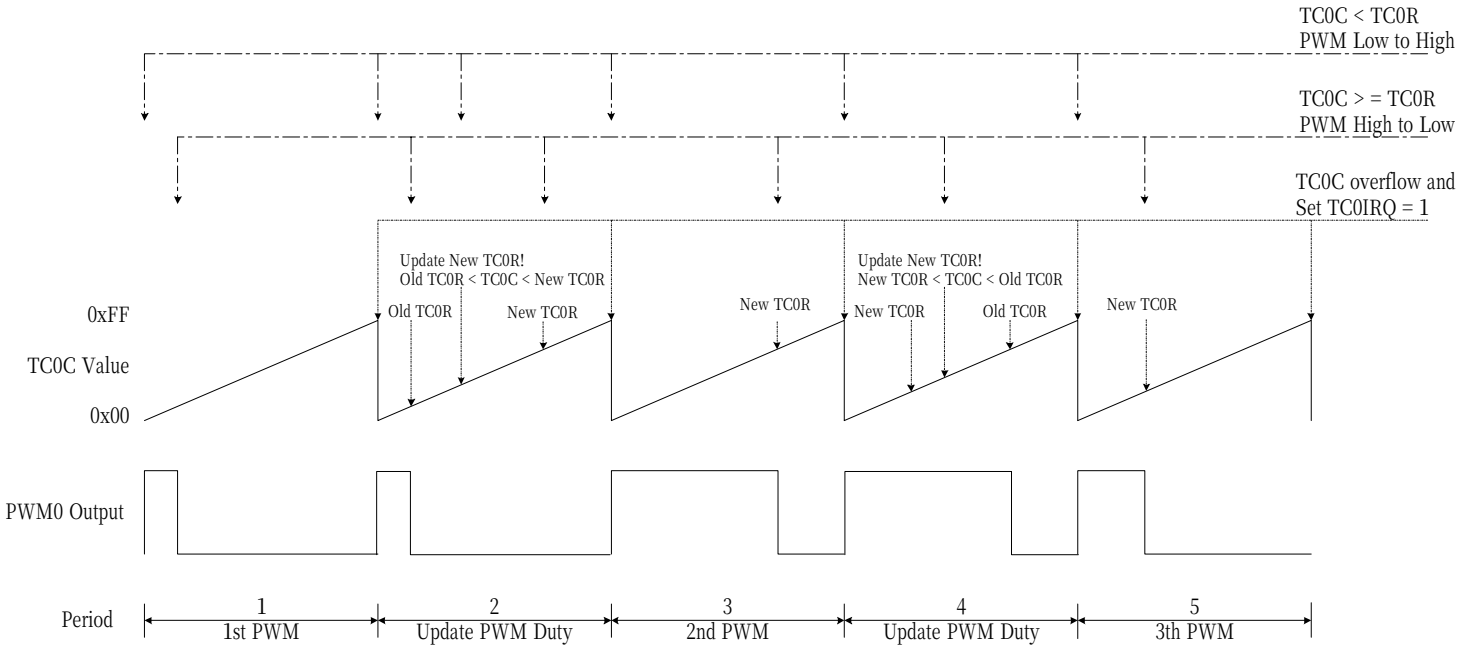
- ***Note3: That is better to set the TC0C and TC0R value together when PWM0 duty modified. It protects the PWM0 signal no glitch as PWM0 duty changing. That is better to set the TC1C and TC1R value together when PWM1 duty modified. It protects the PWM1 signal no glitch as PWM1 duty changing.***
- ***Note4: The TC0OUT function must be set "0" when PWM0 output enable. The TC1OUT function must be set "0" when PWM1 output enable.***
- ***Note5: The PWM can work with interrupt request.***

PWM Duty with TCxR changing

In PWM mode, the system will compare TCxC and TCxR all the time. When TCxC<TCxR, the PWM will output logic "High", when TCxC≧ TCxR, the PWM will output logic "Low". If TCXC is changed in certain period, the PWM duty will change in next PWM period.
When TCxR is fixed all the time, the PWM waveform is also the same



Above diagram is shown the waveform with fixed TCxR. In every TCxC overflow PWM output "High, when TCxC≧ TCxR PWM output "Low".
When TCxR changing in the program processing, the PWM waveform will became:



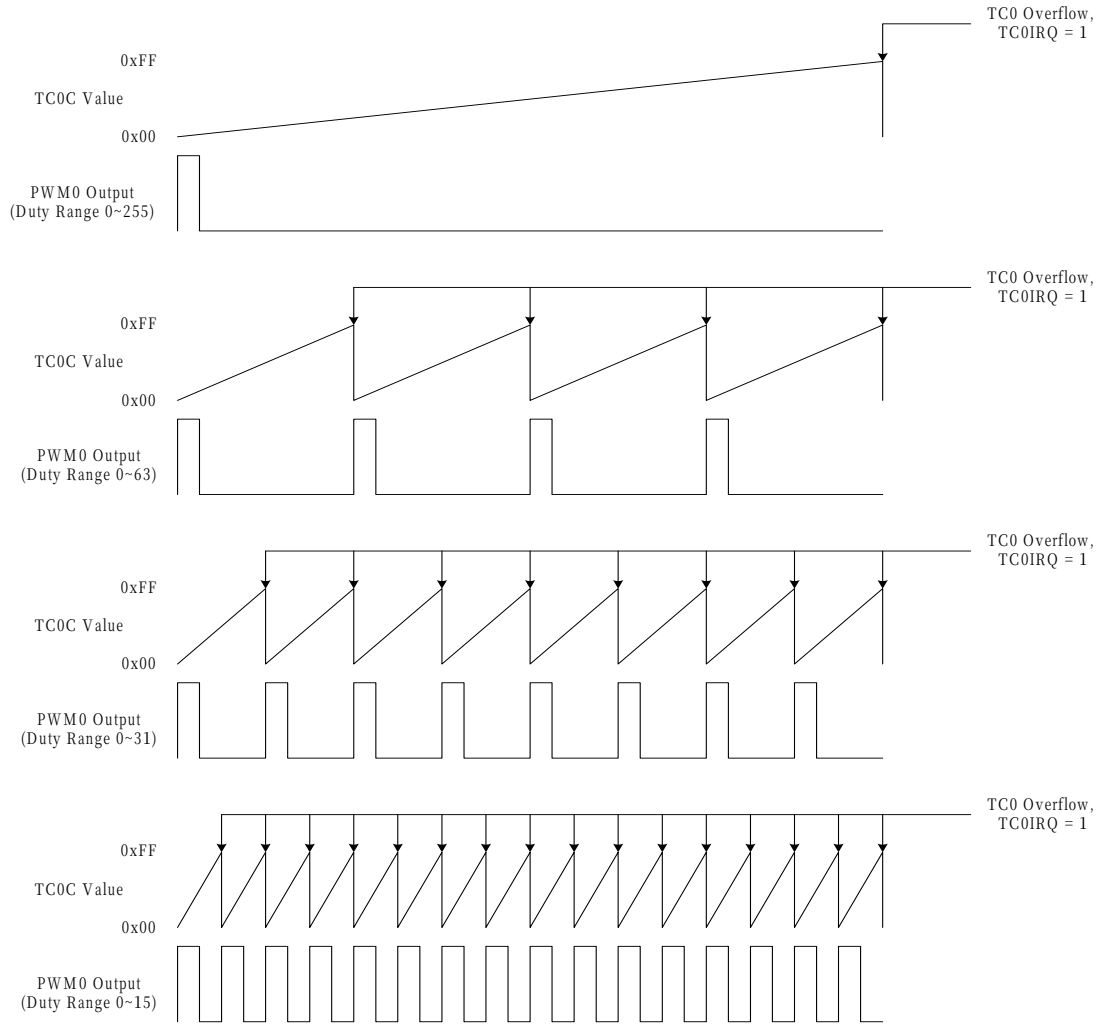
In period 2 and period 4, new Duty (TCxR) is set. However, the PWM still keep the same duty in period 2 and period 4. and the duty changed in next period. By this way, system can avoid the PWM not changing or H/L changing twice in the same cycle and will prevent the unexpected or error operation.

TCxIRQ and PWM Duty

In PWM mode, the frequency of TC0IRQ is depended on PWM duty range.

ALOADx	TCxOUT	TCx Overflow boundary	PWM duty range	TCxIRQ Frequency
0	0	FFh to 00h	0/256 ~ 255/256	TCx clock / 256
0	1	3Fh to 40h	0/64 ~ 63/64	TCx clock / 64
1	0	1Fh to 20h	0/32 ~ 31/32	TCx clock / 32
1	1	0Fh to 10h	0/16 ~ 15/16	TCx clock / 16

From following diagram, the TC0IRQ frequency is related with PWM duty.



9. INTERRUPT

OVERVIEW

The SN8P2710 provides 4 interrupt sources, including two internal interrupts (TC0, TC1) and two external interrupts (INT0, INT1). These external interrupts can wakeup the chip from power down mode to high-speed normal mode. The external clock input pins of INT0/INT1 are shared with P0.0/P0.1 pins. Once interrupt service is executed, the GIE bit in STKP register will clear to "0" for stopping other interrupt request. When interrupt service exits, the GIE bit will set to "1" to accept the next interrupts' request. All of the interrupt request signals are stored in INTRQ register. The user can program the chip to check INTRQ's content for setting executive priority.

➢ ***Note: 1.The GIE bit must enable at first and all interrupt operations work.***

INTEN INTERRUPT ENABLE REGISTER

INTEN is the interrupt request control register including two internal interrupts, two external interrupts. One of the register to be set "1" is to enable the interrupt request function. Once of the interrupt occur, the program jump to ORG 8 to execute interrupt service routines. The program exits the interrupt service routine when the returning interrupt service routine instruction (RETI) is executed.

INTEN initial value = 0000 0000

0C9H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
INTEN	-	TC1IEN	TC0IEN	-	-	-	P01IEN	P00IEN
	-	R/W	R/W	-	-	-	R/W	R/W

Bit 6 **TC1IEN:** Timer interrupt control bit.
0 = Disable TC1 Interrupt
1 = Enable TC1 Interrupt

Bit 5 **TC0IEN:** Timer interrupt control bit.
0 = Disable TC0 Interrupt
1 = Enable TC10Interrupt

Bit 1 **P01IEN:** External P0.1 interrupt control bit.
0 = Disable P01 Interrupt
1 = Enable P01 Interrupt

Bit 0 **P00IEN:** External P0.0 interrupt control bit.
0 = Disable P00 Interrupt
1 = Enable P00 Interrupt

INTRQ INTERRUPT REQUEST REGISTER

INTRQ is the interrupt request flag register. The register includes all interrupt request indication flags. Each one of these interrupt request occurs, the bit of the INTRQ register would be set “1”. The INTRQ value needs to be clear by programming after detecting the flag. In the interrupt vector of program, users know the any interrupt requests occurring by the register and do the routine corresponding of the interrupt request.

INTRQ initial value = x00x xx00

0C8H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
INTRQ	-	TC1IRQ	TC0IRQ	-	-	-	P01IRQ	P00IRQ
	-	R/W	R/W	-	-	-	R/W	R/W

Bit 6 **TC1IRQ:** TC1 timer interrupt request controls bit.
0 = Non request from TC1
1 = Request from TC1

Bit 5 **TC0IRQ:** TC0 timer interrupt request controls bit.
0 = Non request from TC0
1 = Request from TC0

Bit 1 **P01IRQ:** External P0.1 interrupt request bit.
0 = Non-request from P01
1 = Request from P01

Bit 0 **P00IRQ:** External P0.0 interrupt request bit.
0 = Non-request from P00
1 = Request from P00

P0.0 INTERRUPT TRIGGER EDGE CONTROL REGISTER

PEDGE initial value = xxx1 0xxx

0BFH	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
PEDGE	-	-	-	P00G1	P00G0	-	-	-
	-	-	-	R/W	R/W	-	-	-

Bit [4:3] **P00G [1:0]:** P0.0 interrupt trigger edge control register
00 = Reserved
01 = Rising edge
10 = Falling edge **(Reset default setting)**
11 = Falling and rising edge both (level change trigger)

INTERRUPT OPERATION DESCRIPTION

SN8P2710 provides 4 interrupts. The operation of the 4 interrupts is as following.

GIE GLOBAL INTERRUPT OPERATION

GIE is the global interrupt control bit. All interrupts start work after the GIE = 1. It is necessary for interrupt service request. One of the interrupt requests occurs, and the program counter (PC) points to the interrupt vector (ORG 8) and the stack add 1 level.

STKP initial value = 0xxx 1111

0DFH	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
STKP	GIE	-	-	-	-	STKPB2	STKPB1	STKPB0
	R/W	-	-	-	-	R/W	R/W	R/W

Bit 7 **GIE:** Global interrupt control bit.
0 = Disable Interrupt function
1 = Enable Interrupt function

➲ **Example: Set global interrupt control bit (GIE).**

```
B0BSET      FGIE        ; Enable GIE
```

➢ ***Note: The GIE bit must enable and all interrupt operations work.***

INT0 (P0.0) INTERRUPT OPERATION

The INT0 is triggered by falling edge. When the INT0 trigger occurs, the P00IRQ will be set to "1" however the P00IEN is enable or disable. If the P00IEN = 1, the trigger event will make the P00IRQ to be "1" and the system enter interrupt vector. If the P00IEN = 0, the trigger event will make the P00IRQ to be "1" but the system will not enter interrupt vector. Users need to care for the operation under multi-interrupt situation.

➲ **Example: INT0 interrupt request setup.**

```
        B0BSET      FP00IEN         ; Enable INT0 interrupt service
        B0BCLR      FP00IRQ         ; Clear INT0 interrupt request flag
        B0BSET      FGIE            ; Enable GIE
```

➲ **Example: INT0 interrupt service routine.**

```
        ORG         8               ; Interrupt vector
        JMP         INT_SERVICE
INT_SERVICE:

        B0XCH       A, ACCBUF       ; B0XCH doesn't change C, Z flag
        B0MOV       A, PFLAG
        B0MOV       PFLAGBUF, A     ; Save PFLAG register in a buffer

        B0BTS1      FP00IRQ         ; Check P00IRQ
        JMP         EXIT_INT        ; P00IRQ = 0, exit interrupt vector

        B0BCLR      FP00IRQ         ; Reset P00IRQ
        .           .               ; INT0 interrupt service routine
        .           .
EXIT_INT:
        B0MOV       A, PFLAGBUF
        B0MOV       PFLAG, A        ; Restore PFLAG register from buffer
        B0XCH       A, ACCBUF       ; Restore ACC value.

        RETI                        ; Exit interrupt vector
```

INT1 (P0.1) INTERRUPT OPERATION

The INT1 is triggered by falling edge. When the INT1 trigger occurs, the P01IRQ will be set to "1" however the P01IEN is enable or disable. If the P01IEN = 1, the trigger event will make the P01IRQ to be "1" and the system enter interrupt vector. If the P01IEN = 0, the trigger event will make the P01IRQ to be "1" but the system will not enter interrupt vector. Users need to care for the operation under multi-interrupt situation.

➲ **Example: INT1 interrupt request setup.**

```
        B0BSET      FP01IEN         ; Enable INT1 interrupt service
        B0BCLR      FP01IRQ         ; Clear INT1 interrupt request flag
        B0BSET      FGIE            ; Enable GIE
```

➲ **Example: INT1 interrupt service routine.**

```
            ORG     8               ; Interrupt vector
            JMP     INT_SERVICE
INT_SERVICE:

            B0XCH   A, ACCBUF       ; B0XCH doesn't change C, Z flag
            B0MOV   A, PFLAG
            B0MOV   PFLAGBUF, A     ; Save PFLAG register in a buffer

            B0BTS1  FP01IRQ         ; Check P01IRQ
            JMP     EXIT_INT        ; P01IRQ = 0, exit interrupt vector

            B0BCLR  FP01IRQ         ; Reset P01IRQ
            .       .               ; INT1 interrupt service routine
            .       .
EXIT_INT:
            B0MOV   A, PFLAGBUF
            B0MOV   PFLAG, A        ; Restore PFLAG register from buffer
            B0XCH   A, ACCBUF       ; Restore ACC value.

            RETI                    ; Exit interrupt vector
```

TC0 INTERRUPT OPERATION

When the TC0C counter occurs overflow, the TC0IRQ will be set to “1” no matter the TC0IEN is enable or disable. If the TC0IEN = 1, the trigger event will make the TC0IRQ to be “1” and the system enter interrupt vector. If the TC0IEN = 0, the trigger event will make the TC0IRQ to be “1” but the system will not enter interrupt vector. Users need to care for the operation under multi-interrupt situation.

➲ Example: TC0 interrupt request setup.

```
        B0BCLR      FTC0IEN         ; Disable TC0 interrupt service
        B0BCLR      FTC0ENB         ; Disable TC0 timer
        MOV         A, #20H         ;
        B0MOV       TC0M, A         ; Set TC0 clock = Fcpu / 64
        MOV         A, #74H         ; Set TC0C initial value = 74H
        B0MOV       TC0C, A         ; Set TC0 interval = 10 ms

        B0BSET      FTC0IEN         ; Enable TC0 interrupt service
        B0BCLR      FTC0IRQ         ; Clear TC0 interrupt request flag
        B0BSET      FTC0ENB         ; Enable TC0 timer

        B0BSET      FGIE            ; Enable GIE
```

➲ Example: TC0 interrupt service routine.

```
        ORG         8               ; Interrupt vector
        JMP         INT_SERVICE
INT_SERVICE:

        B0XCH       A, ACCBUF       ; B0XCH doesn't change C, Z flag
        B0MOV       A, PFLAG
        B0MOV       PFLAGBUF, A     ; Save PFLAG register in a buffer

        B0BTS1      FTC0IRQ         ; Check TC0IRQ
        JMP         EXIT_INT        ; TC0IRQ = 0, exit interrupt vector

        B0BCLR      FTC0IRQ         ; Reset TC0IRQ
        MOV         A, #74H
        B0MOV       TC0C, A         ; Reset TC0C.
        .               .           ; TC0 interrupt service routine
        .               .
EXIT_INT:
        B0MOV       A, PFLAGBUF
        B0MOV       PFLAG, A        ; Restore PFLAG register from buffer
        B0XCH       A, ACCBUF       ; Restore ACC value.

        RETI                        ; Exit interrupt vector
```

TC1 INTERRUPT OPERATION

When the TC1C counter occurs overflow, the TC1IRQ will be set to “1” no matter the TC1IEN is enable or disable. If the TC1IEN = 1, the trigger event will make the TC1IRQ to be “1” and the system enter interrupt vector. If the TC1IEN = 0, the trigger event will make the TC1IRQ to be “1” but the system will not enter interrupt vector. Users need to care for the operation under multi-interrupt situation.

➲ Example: TC1 interrupt request setup.

```
        B0BCLR      FTC1IEN         ; Disable TC1 interrupt service
        B0BCLR      FT C1ENB        ; Disable TC1 timer
        MOV         A, #20H         ;
        B0MOV       TC1M, A         ; Set TC1 clock = Fcpu / 64
        MOV         A, #74H         ; Set TC1C initial value = 74H
        B0MOV       TC1C, A         ; Set TC1 interval = 10 ms

        B0BSET      FTC1IEN         ; Enable TC1 interrupt service
        B0BCLR      FTC1IRQ         ; Clear TC1 interrupt request flag
        B0BSET      FTC1ENB         ; Enable TC1 timer

        B0BSET      FGIE            ; Enable GIE
```

➲ Example: TC1 interrupt service routine.

```
        ORG         8               ; Interrupt vector
        JMP         INT_SERVICE
INT_SERVICE:

        B0XCH       A, ACCBUF       ; B0XCH doesn’ t change C, Z flag
        B0MOV       A, PFLAG
        B0MOV       PFLAGBUF, A     ; Save PFLAG register in a buffer

        B0BTS1      FTC1IRQ         ; Check TC1IRQ
        JMP         EXIT_INT        ; TC1IRQ = 0, exit interrupt vector

        B0BCLR      FTC1IRQ         ; Reset TC1IRQ
        MOV         A, #74H
        B0MOV       TC1C, A         ; Reset TC1C.
        .               .           ; TC1 interrupt service routine
        .               .
EXIT_INT:
        B0MOV       A, PFLAGBUF
        B0MOV       PFLAG, A        ; Restore PFLAG register from buffer
        B0XCH       A, ACCBUF       ; Restore ACC value.

        RETI                        ; Exit interrupt vector
```

MULTI-INTERRUPT OPERATION

In most conditions, the software designer uses more than one interrupt request. Processing multi-interrupt request needs to set the priority of these interrupt requests. The IRQ flags of the 4 interrupt are controlled by the interrupt event occurring. But the IRQ flag set doesn't mean the system to execute the interrupt vector. The IRQ flags can be triggered by the events without interrupt enable. Just only any the event occurs and the IRQ will be logic "1". The IRQ and its trigger event relationship is as the below table.

Interrupt Name	*Trigger Event Description*
P00IRQ	P0.0 trigger. Falling edge.
P01IRQ	P0.1 trigger. Falling edge.
TC0IRQ	TC0C overflow.
TC1IRQ	TC1C overflow.

There are two things need to do for multi-interrupt. One is to make a good priority for these interrupt requests. Two is using IEN and IRQ flags to decide executing interrupt service routine or not. Users have to check interrupt control bit and interrupt request flag in interrupt vector. There is a simple routine as following.

➲ **Example: How does users check the interrupt request in multi-interrupt situation?**

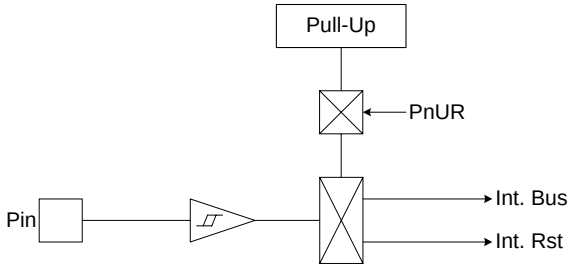
```
        ORG     8               ; Interrupt vector
        NOP
        B0XCH   A, ACCBUF       ; B0XCH doesn't change C, Z flag
        B0MOV   A, PFLAG
        B0MOV   PFLAGBUF, A     ; Save PFLAG register in a buffer
INTP00CHK:                      ; Check INT0 interrupt request
        B0BTS1  FP00IEN         ; Check P00IEN
        JMP     INTP01CHK       ; Jump check to next interrupt
        B0BTS0  FP00IRQ         ; Check P00IRQ
        JMP     INTP00          ; Jump to INT0 interrupt service routine
INTP01CHK:                      ; Check INT1 interrupt request
        B0BTS1  FP01IEN         ; Check P01IEN
        JMP     INTTC0CHK       ; Jump check to next interrupt
        B0BTS0  FP01IRQ         ; Check P01IRQ
        JMP     INTP01          ; Jump to INT1 interrupt service routine
INTTC0CHK:                      ; Check TC0 interrupt request
        B0BTS1  FTC0IEN         ; Check TC0IEN
        JMP     INTTC1CHK       ; Jump check to next interrupt
        B0BTS0  FTC0IRQ         ; Check TC0IRQ
        JMP     INTTC0          ; Jump to TC0 interrupt service routine
INTTC1HK:                       ; Check TC1 interrupt request
        B0BTS1  FTC1IEN         ; Check TC1IEN
        JMP     INT_EXIT        ; Jump check to next interrupt
        B0BTS0  FTC1IRQ         ; Check TC1IRQ
        JMP     INTTC1          ; Jump to TC1 interrupt service routine
INT_EXIT:
        B0MOV   A, PFLAGBUF
        B0MOV   PFLAG, A        ; Restore PFLAG register from buffer
        B0XCH   A, ACCBUF       ; Restore ACC value.

        RETI                    ; Exit interrupt vector
```

10. I/O PORT

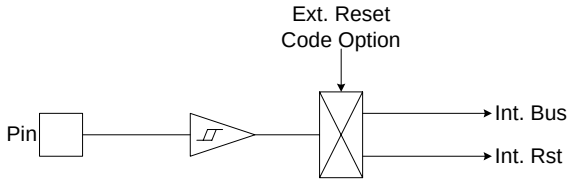
OVERVIEW

The SN8P2710 provides up to four ports for users' application, consisting of one input only port (P0), three I/O ports (P2, P4, P5). The direction of I/O port is selected by PnM register and register PnUR is defined for user setting pull-up register. After the system resets, all ports work as input function without pull-up resistors.

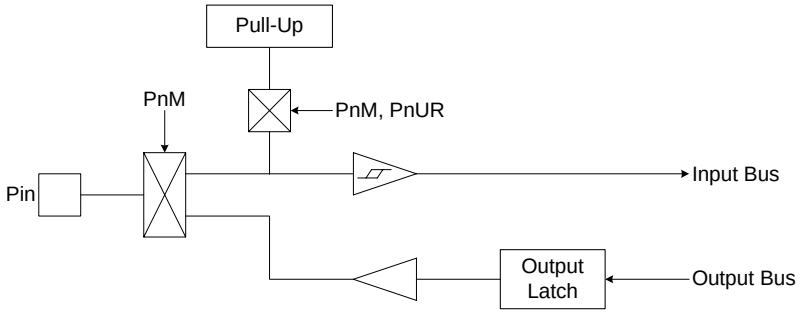
Port 0.1 and P0.2 structure:



Port 0.3 structure:



Port 2, 5 structure:



Port 4 structure:

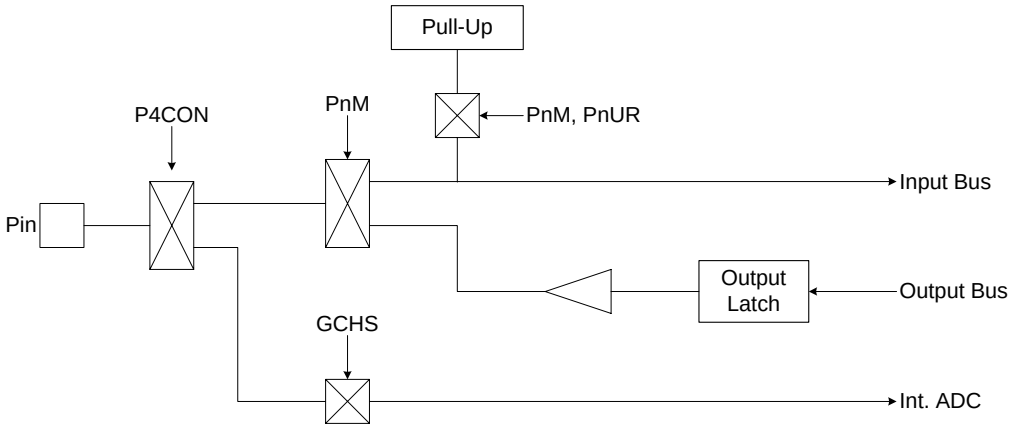


Figure 10-1. The I/O pin circuit Diagram

I/O PORT FUNCTION TABLE

Port/Pin	I/O	Function Description	Remark
P0.0~P0.1	I	General-purpose input function	P0.0: TC0 external clock input pin P0.1: TC1 external clock input pin
		External interrupt (INT0~INT1)	
		Wakeup for power down mode	
P0.2	I	General-purpose input function	
		Wakeup for power down mode	
P0.3	I	General-purpose input function	**No pull-up, No wakeup function**
		Share with Reset pin	
P2.0~P2.7	I/O	General-purpose input/output function	
P4.0~P4.7	I/O	General-purpose input/output function	
		ADC analog signal input	
P5.0~P5.6	I/O	General-purpose input/output function	

Table 10-1. I/O Function Table

PULL-UP RESISTERS

Pull-up register can set Pull-up register by Port. The typical pull-up register value is 200K@3V and 100K@5V.

➢ ***Port0***

0E0H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
P0UR	-	-	-	-	-	P02R	P01R	P00R
Read/Write	-	-	-	-	-	W	W	W
After reset	-	-	-	-	-	0	0	0

➢ ***Port2***

0E2H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
P2UR	P27r	P26R	P25R	P24R	P23R	P22R	P21R	P20R
Read/Write	W	W	W	W	W	W	W	W
After reset	0	0	0	0	0	0	0	0

➢ ***Port4***

0E4H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
P4UR	P47R	P46R	P45R	P44R	P43R	P42R	P41R	P40R
Read/Write	W	W	W	W	W	W	W	W
After reset	0	0	0	0	0	0	0	0

➢ ***Port5***

0E5H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
P5UR		P56R	P55R	P54R	P53R	P52R	P51R	P50R
Read/Write		W	W	W	W	W	W	W
After reset		0	0	0	0	0	0	0

➲ **Example: I/O Pull up Register**

```
CLR     P0UR        ; Disable Port0 Pull-up register.


MOV     A, #01H     ; ,
B0MOV   P0UR, A     ; Enable Port0.0  Pull-up Register
```

I/O PORT MODE

The port direction is programmed by PnM register. Port 0 is always input mode. Port 1,2,3,4 and 5 can select input or output direction.

P2M initial value = 0000 0000

0C2H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
P2M	P27M	P26M	P25M	P24M	P23M	P22M	P21M	P20M
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Bit [7:0] **P2 [7:0] M:** P2.0~P2.7 I/O direction control bit.
0 = Set P2 as input mode
1 = Set P2 as output mode

P4M initial value = 0000 0000

0C4H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
P4M	P47M	P46M	P45M	P44M	P43M	P42M	P41M	P40M
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Bit [7:0] **P4 [7:0] M:** P4.0~P4.7 I/O direction control bit.
0 = Set P4 as input mode
1 = Set P4 as output mode

P5M initial value = x000 0000

0C5H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
P5M		P56M	P55M	P54M	P53M	P52M	P51M	P50M
		R/W	R/W	R/W	R/W	R/W	R/W	R/W

Bit [6:0] **P5 [6:0] M:** P5.0~P5.6 I/O direction control bit.
0 = Set P5 as input mode
1 = Set P5 as output mode

➲ **Example: I/O mode selecting.**

```
CLR     P2M
CLR     P4M
CLR     P5M


MOV     A, #0FFH        ; Set all ports to be output mode.
B0MOV   P2M, A
B0MOV   P4M, A
B0MOV   P5M, A

B0BCLR  P2M.5           ; Set P2.5 to be input mode.

B0BSET  P2M.5           ; Set P2.5 to be output mode.
```

I/O PORT DATA REGISTER

P0 initial value = xxxx xxxx

0D0H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
P0	-	-	-	-	P03	P02	P01	P00
	-	-	-	-	R	R	R	R

P2 initial value = xxxx xxxx

0D2H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
P2	P27	P26	P25	P24	P23	P22	P21	P20
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

P4 initial value = xxxx xxxx

0D4H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
P4	P47	P46	P45	P44	P43	P42	P41	P40
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

P5 initial value = xxxx xxxx

0D5H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
P5		P56	P55	P54	P53	P52	P51	P50
		R/W	R/W	R/W	R/W	R/W	R/W	R/W

➲ **Example: Read data from input port.**

```
B0MOV   A, P0       ; Read data from Port 0
B0MOV   A, P2       ; Read data from Port 2
B0MOV   A, P4       ; Read data from Port 4
B0MOV   A, P5       ; Read data from Port 5
```

➲ **Example: Write data to output port.**

```
MOV     A, #55H     ; Write data 55H to Port 1, Port2, Port 4, Port 5
B0MOV   P2, A
B0MOV   P4, A
B0MOV   P5, A
```

➲ **Example: Write one bit data to output port.**

```
B0BSET      P2.3        ; Set P2.3 and P4.0 to be “1”.
B0BSET      P4.0

B0BCLR      P2.3        ; Set P2.3 and P5.5 to be “0”.
B0BCLR      P5.5
```

➲ **Example: Port bit test.**

```
B0BTS1      P0.0        ; Bit test 1 for P0.0
.
B0BTS0      P2.5        ; Bit test 0 for P2.5
```

11. 8-CHANNEL ANALOG TO DIGITAL CONVERTER

OVERVIEW

This analog to digital converter of SN8P2710 has 8-input sources with up to 4096-step resolution to transfer analog signal into 12-bits digital data. The sequence of ADC operation is to select input source (AIN0 ~ AIN7) at first, then set GCHS and ADS bit to "1" to start conversion. When the conversion is complete, the ADC circuit will set EOC bit to "1" and final value output in ADB register.

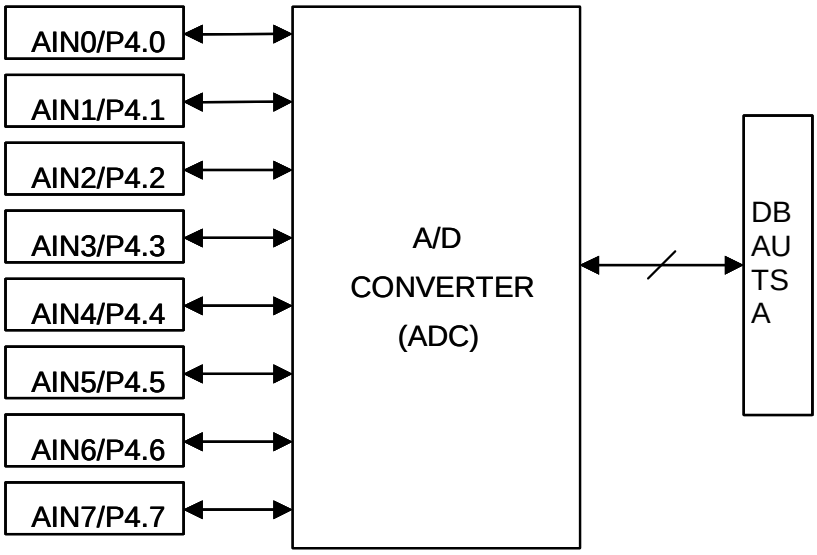


Figure 11-1. AD Converter Function Diagram

- ***Note:For 12-bit resolution the conversion time is 16 steps***
- ***Note: The analog input level must be between the AVREFH and AVREFL.***
- ***Note: The AVREFH level must be between the AVDD and VSS + 2.0V.***
- ***Note: ADC programming notice:***
  - ***Set ADC input pin I/O direction as input mode***
  - ***Disable pull-up resistor of ADC input pin***
  - ***Disable ADC before enter power down (sleep) mode to save power consumption.***
  - ***Set related bit of P4CON register to avoid extra power consumption in power down mode.***
  - ***Delay 100uS after enable ADC (set ADENB = "1") to wait ADC circuit ready for conversion.***
  - ***Disable ADC (set ADENB = "0") before enter sleep mode to save power consumption.***

ADM REGISTER

ADM initial value = 0000 x000

0B1H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
ADM	ADENB	ADS	EOC	GCHS	-	CHS2	CHS1	CHS0
	R/W	R/W	R/W	R/W	-	R/W	R/W	R/W

Bit 7 **ADENB:** ADC control bit.
0 = Disable ADC function
1 = Enable ADC function

Bit 6 **ADS:** ADC start bit.
0 = ADC convert stop
1 = ADC convert starting

Bit 5 **EOC:** ADC status bit.
0 = Progressing
1 = End of converting and reset ADS bit

Bit 4 **GCHS:** Global channel select bit.
0 = Disable AIN channel
1 = Enable AIN channel

Bit [2:0] **CHS [2:0]:** ADC input channels select bit.
000 = AIN0, 001 = AIN1, 010 = AIN2, 011 = AIN3
100 = AIN4, 101 = AIN5, 110 = AIN6, 111 = AIN7

ADR REGISTERS

ADR initial value = x00x 0000

0B3H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
ADR		ADCKS1		ADCKS0	ADB3	ADB2	ADB1	ADB0
		R/W		R/W	R	R	R	R

Bit 6,4 **ADCKS [1:0]:** ADC's clock source select bit.

ADCKS1	ADCKS0	ADC Clock Source
0	0	Fcpu/16
0	1	Fcpu/8
1	0	Fcpu
1	1	Fcpu/2

Bit [3:0] **ADB [3:0]:** ADC data buffer.
ADB11~ADB0 bits for 12-bit ADC

ADB REGISTERS

ADB initial value = xxxx xxxx

0B2H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
ADB	ADB11	ADB10	ADB9	ADB8	ADB7	ADB6	ADB5	ADB4
	R	R	R	R	R	R	R	R

ADB is ADC data buffer to store AD converter result. The ADB is only 8-bit register including bit 4~bit11 ADC data. To combine ADB register and the low-nibble of ADR will get full 12-bit ADC data buffer. The ADC buffer is a read-only register. the ADC data is stored in ADB and ADR registers.

➢ ***Note: ADB [0:11] value is unknown when power on.***

The AIN's input voltage v.s. ADB's output data

AIN n	ADB11	ADB10	ADB9	ADB8	ADB7	ADB6	ADB5	ADB4	ADB3	ADB2	ADB1	ADB0
0/4095*AVREFH	0	0	0	0	0	0	0	0	0	0	0	0
1/4095*AVREFH	0	0	0	0	0	0	0	0	0	0	0	1
.	.	.	.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.	.	.	.
4094/4095*AVREFH	1	1	1	1	1	1	1	1	1	1	1	0
4095/4095*AVREFH	1	1	1	1	1	1	1	1	1	1	1	1

For different applications, users maybe need more than 8-bit resolution but less than 12-bit ADC converter. To process the ADB and ADR data can make the job well. First, the AD resolution must be set 12-bit mode and then to execute ADC converter routine. Then delete the LSB of ADC data and get the new resolution result. The table is as following.

ADC Resolution	**ADB**								**ADR**			
	ADB11	**ADB10**	**ADB9**	**ADB8**	**ADB7**	**ADB6**	**ADB5**	**ADB4**	**ADB3**	**ADB2**	**ADB1**	**ADB0**
8-bit	O	O	O	O	O	O	O	O	x	x	x	x
9-bit	O	O	O	O	O	O	O	O	O	x	x	x
10-bit	O	O	O	O	O	O	O	O	O	O	x	x
11-bit	O	O	O	O	O	O	O	O	O	O	O	x
12-bit	O	O	O	O	O	O	O	O	O	O	O	O
O = Selected, x = Delete												

P4CON REGISTERS

ADB initial value = 0000 0000

0AEH	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
P4CON	P4CON7	P4CON6	P4CON5	P4CON4	P4CON3	P4CON2	P4CON1	P4CON0
	W	W	W	W	W	W	W	W

The Port 4 is shared with ADC input function. Only one pin of port 4 can be configured as ADC input in the same time by ADM register. The other pins of port 4 are digital I/O pins. Connect an analog signal to COMS digital input pin, especially the analog signal level is about 1/2 VDD will cause extra current leakage. In the power down mode, the leakage current will be a big issue. Unfortunately, if users connect more than one analog input signal to port 4 will encounter above current leakage situation. P4CON is Port4 Configuration register. Write "1" into P4CON [7:0] will configure related port 4 pin as pure analog input pin to avoid current leakage.

Bit[7:0] **P4CON [7:0]** Port4 Configuration register.
0 = P4.X can be an analog input (ADC input) or digital I/O pins.
1 = P4.X is pure analog input, can't be a digital I/O pin.
Note: When Port4 [7:0] is general I/O port not ADC channel, P4CON [7:0] must set to "0" or the Port4 input signal would be isolated

ADC CONVERTING TIME

12-bit ADC conversion time = 1/(ADC clock)*16 sec

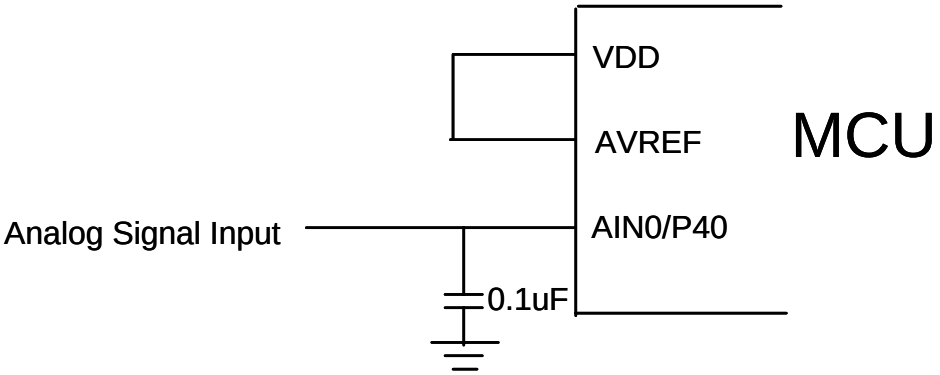
High Clock (Fosc) = 4MHz

Fcpu		ADCKS1	ADCKS0	ADC Clock	ADC Converting Time
Fosc/	1	0	0	Fcpu/16	1/((4MHz / 1) / 16 /4) x16= 256 us
		0	1	Fcpu/8	1/((4MHz / 1) / 8 /4) x16= 128 us
		1	0	Fcpu	1/((4MHz / 1) / 1 /4) x16= 16 us
		1	1	Fcpu/2	1/((4MHz / 1) / 2 /4) x16= 32 us
Fosc/	2	0	0	Fcpu/16	1/((4MHz / 2) / 16 /4) x16= 512 us
		0	1	Fcpu/8	1/((4MHz / 2) / 8 /4) x16= 256 us
		1	0	Fcpu	1/((4MHz / 2) / 1 /4) x16= 32 us
		1	1	Fcpu/2	1/((4MHz / 2) / 2 /4) x16= 64 us
Fosc/	4	0	0	Fcpu/16	1/((4MHz / 4) / 16 /4) x16= 1024 us
		0	1	Fcpu/8	1/((4MHz / 4) / 8 /4) x16= 512 us
		1	0	Fcpu	1/((4MHz / 4) / 1 /4) x16= 64 us
		1	1	Fcpu/2	1/((4MHz / 4) / 2 /4) x16= 128 us
Fosc/	8	0	0	Fcpu/16	1/((4MHz / 8) / 16 /4) x16= 2048 us
		0	1	Fcpu/8	1/((4MHz / 8) / 8 /4) x16= 1024 us
		1	0	Fcpu	1/((4MHz / 8) / 1 /4) x16= 128 us
		1	1	Fcpu/2	1/((4MHz / 8) / 2 /4) x16= 256 us

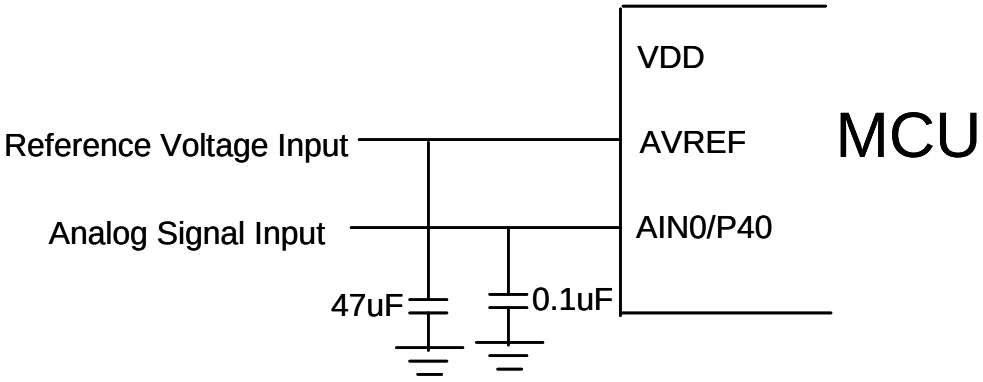
➲ Example : Configure AIN0 as 12-bit ADC input and start ADC conversion then enter power down mode.

```
ADC0:
        B0BSET      FADENB          ; Enable ADC circuit
        CALL        Delay100uS      ; Delay 100uS to wait ADC circuit ready for conversion
        MOV         A, #0FEh
        B0MOV       P4UR, A         ; Disable P4.0 pull-up resistor
        B0BCLR      FP40M           ; Set P4.0 as input pin
        MOV         A, #01h
        B0MOV       P4CON, A        ; Set P4.0 as pure analog input
        MOV         A, #40H
        B0MOV       ADR, A          ; To set 12-bit ADC and ADC clock = Fosc.
        MOV         A,#90H
        B0MOV       ADM,A           ; To enable ADC and set AIN0 input
        B0BSET      FADS            ; To start conversion
WADC0:
        B0BTS1      FEOC            ; To skip, if end of converting =1
        JMP         WADC0           ; else, jump to WADC0
        B0MOV       A,ADB           ; To get AIN0 input data bit11 ~ bit4
        B0MOV       Adc_Buf_Hi, A
        B0MOV       A,ADR           ; To get AIN0 input data bit3 ~ bit0
        AND         A, 0Fh
        B0MOV       Adc_Buf_Low, A
Power_Down  .       .
        B0BCLR      FADENB          ; Disable ADC circuit
        B0BSET      FCPUM0          ; Enter sleep mode
```

ADC CIRCUIT



AVREFH is connected to VDD.



AVREFH is connected to external AD reference voltage.

Figure 11-2. The AINx and AVREFH Circuit of AD Converter

- ***Note: The capacitor between AIN and GND is a bypass capacitor. It is helpful to stable the analog signal. Users can omit it.***

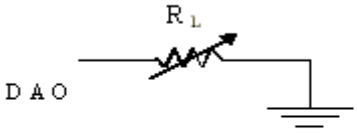
12. 7-BIT DIGITAL TO ANALOG CONVERTER

OVERVIEW

The D/A converter uses 7-bit structure to synthesize 128 steps' analog signal with current source output. After DAENB bit is set to “1”, DAC circuit will turn to be enabled and the DAM register, from bit0 to bit6, will send digital signal to ladder resistors in order to generate analog signal on DAO pin.

Figure 12-1. The DA converter Block Diagram

In order to get a proper linear output, a Loading Resistor R_L is usually added between DAO and Ground. The example shows the result of Vdd = 5V, R_L =150ohm and Vdd = 3V, R_L =150ohm.



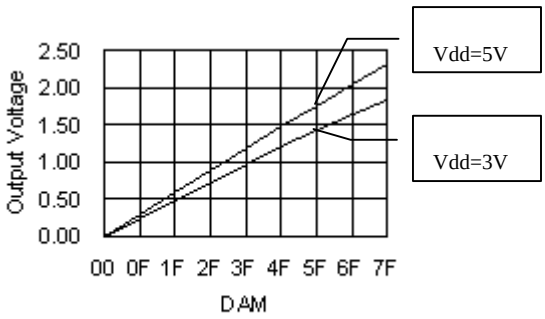


Figure 12-2 DAO Circuit with R_L

Figure 12-3. DAC Output Voltage in Vdd=5V and 3V

The D/A converter is not designed for a precise DC voltage output and is suitable for a simple audio application e.g. Tone or Melody generation.

DAM REGISTER

DAM initial value = 0000 0000

0B0H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
DAM	DAENB	DAB6	DAB5	DAB4	DAB3	DAB2	DAB1	DAB0
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Bit 7 **DAENB:** Digital to Analog converter control bit.
0 = Disable DAC function
1 = Enable DAC function

Bit [6:0] **DAB [6:0]:** Digital input data.

D/A CONVERTER OPERATION

When the DAENB = 0, the DAO pin is output floating status. After setting DAENB to “1”, the DAO output value is controlled by DAB bits.

➲ **Example: Output 1/2 VDD from DAO pin.**

```
MOV     A, #00111111B
B0MOV   DAM, A          ; Set DAB to a half of the full scale.

B0BSET  FDAENB          ; Enable D/A function.
```

The DAB’s data v.s. DAO’s output voltage as following:

DAB6	DAB5	DAB4	DAB3	DAB2	DAB1	DAB0	DAO
0	0	0	0	0	0	0	VSS
0	0	0	0	0	0	1	Idac
0	0	0	0	0	1	0	2 * Idac
0	0	0	0	0	1	1	3 * Idac
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
1	1	1	1	1	1	0	126 * Idac
1	1	1	1	1	1	1	127 * Idac

Table 12-1. DAB and DAO Relative Table

➢ ***Note: Idac = I_{FSO} / (2^7-1) (I_{FSO}: Full-scale Output Current)***

13. CODING ISSUE

TEMPLATE CODE

```
;*************************************************************************
; FILENAME  : TEMPLATE.ASM
; AUTHOR    : SONiX
; PURPOSE   : Template Code for SN8P2710
; REVISION  : V1.0  First issue
;*************************************************************************
;*  (c) Copyright 2004, SONiX TECHNOLOGY CO., LTD.
;*************************************************************************
CHIP    SN8P2715                    ; Select the CHIP

;-------------------------------------------------------------------------
;                           Include Files
;-------------------------------------------------------------------------
.nolist                             ; do not list the macro file
    INCLUDESTD  MACRO1.H
    INCLUDESTD  MACRO2.H
    INCLUDESTD  MACRO3.H

.list                               ; Enable the listing function

;-------------------------------------------------------------------------
;                           Constants Definition
;-------------------------------------------------------------------------
;   ONE         EQU     1
;-------------------------------------------------------------------------
;                           Variables Definition
;-------------------------------------------------------------------------
.DATA
                org     0h          ;Bank 0 data section start from RAM address 0x000
    Wk00B0      DS      1           ;Temporary buffer for main loop
    Iwk00B0     DS      1           ;Temporary buffer for ISR
    AccBuf      DS      1           ;Accumulater buffer
    PflagBuf    DS      1           ;PFLAG buffer

                org     100h        ;Bank 1 data section start from RAM address 0x100
    BufB1       DS      20          ;Temporary buffer in bank 1

;-------------------------------------------------------------------------
;                           Bit Flag Definition
;-------------------------------------------------------------------------
    Wk00B0_0    EQU     Wk00B0.0    ;Bit 0 of Wk00B0
    Iwk00B0_1   EQU     Iwk00B0.1   ;Bit 1 of Iwk00

;-------------------------------------------------------------------------
;                           Code section
;-------------------------------------------------------------------------
.CODE

    ORG         0                   ;Code section start
    jmp         Reset               ;Reset vector
                                    ;Address 4 to 7 are reserved
    ORG         8
    jmp         Isr                 ;Interrupt vector
```

```
    ORG         10h
;-------------------------------------------------------------------------------
;   Program reset section
;-------------------------------------------------------------------------------
Reset:
    mov         A,#07Fh             ;Initial stack pointer and
    b0mov       STKP,A              ;disable global interrupt

    clr         PFLAG               ;pflag = x,x,x,x,x,c,dc,z
    mov         A,#00h              ;Initial system mode
    b0mov       OSCM,A

    mov         A, #0x5A
    b0mov       WDTR, A             ;Clear watchdog timer

    call        ClrRAM              ;Clear RAM
    call        SysInit             ;System initial
    b0bset      FGIE                ;Enable global interrupt

;-------------------------------------------------------------------------------
;   Main routine
;-------------------------------------------------------------------------------
Main:
    mov         A, #0x5A            ;Clear watchdog timer
    b0mov       WDTR, A

    call        MnApp

    jmp         Main
;-------------------------------------------------------------------------------
;   Main application
;-------------------------------------------------------------------------------
MnApp:

    ; Put your main program here
    ret

;-----------------------------------
;   Jump table routine
;-----------------------------------
    ORG         0x0100              ;The jump table should start from the head
                                    ;of boundary.
    b0mov       A,Wk00B0
    and         A,#3
    ADD         PCL,A
    jmp         JmpSub0
    jmp         JmpSub1
    jmp         JmpSub2
;-----------------------------------

JmpSub0:
    ; Subroutine 1
    jmp         JmpExit
JmpSub1:

    ; Subroutine 2
    jmp         JmpExit

JmpSub2:
```

```
    ; Subroutine 3
    jmp         JmpExit

JmpExit:
    ret                             ;Return Main

;-------------------------------------------------------------------------------
; Isr (Interrupt Service Routine)
; Arguments :
; Returns   :
; Reg Change:
;-------------------------------------------------------------------------------
Isr:
;-----------------------------------
;   Save ACC and system registers
;-----------------------------------
    b0xch       A,AccBuf            ;B0xch instruction do not change C,Z flag
    B0MOV       A, PFLAG
    B0MOV       PFLAGBUF, A

;-----------------------------------
;  Check which interrupt happen
;-----------------------------------

IntP00Chk:
    b0bts1      FP00IEN
    jmp         IntTc0Chk           ;Modify this line for another interrupt
    b0bts0      FP00IRQ
    jmp         P00isr

    ;If necessary, insert another interrupt checking here

IntTc0Chk:
    b0bts1      FTC0IEN
    jmp         IsrExit             ;Suppose TC0 is the last interrupt which you
    b0bts0      FTC0IRQ             ;want to check
    jmp         TC0isr
;-----------------------------------
; Exit interrupt service routine
;-----------------------------------
IsrExit:
    B0MOV       A, PFLAGBUF                     ;
    B0MOV       PFLAG, A
    b0xch       A,AccBuf            ;B0xch instruction do not change C,Z flag

    reti                            ;Exit the interrupt routine

;-------------------------------------------------------------------------------
;   INT0 interrupt service routine
;-------------------------------------------------------------------------------
P00isr:
    b0bclr      FP00IRQ

    ;Process P0.0 external interrupt here

    jmp         IsrExit

;-------------------------------------------------------------------------------
;   TC0 interrupt service routine
;-------------------------------------------------------------------------------
```

```
TC0isr:
    b0bclr      FTC0IRQ

    ;Process TC0 timer interrupt here

    jmp         IsrExit

;-------------------------------------------------------------------------------
;   SysInit
;   Initialize I/O, Timer, Interrupt, etc.
;-------------------------------------------------------------------------------
SysInit:
    ret

;-------------------------------------------------------------------------------
; ClrRAM
; Use index @YZ to clear RAM  (00h~7Fh)
;-------------------------------------------------------------------------------

ClrRAM:

; RAM Bank 0
    clr         Y                   ;Select bank 0
    b0mov       Z,#0x7f             ;Set @YZ address from 7fh

ClrRAM10:
    clr         @YZ                 ;Clear @YZ content
    decms       Z                   ;z = z – 1 , skip next if z=0
    jmp         ClrRAM10
    clr         @YZ                 ;Clear address 0x00
    ret

;-------------------------------------------------------------------------------
    ENDP
```

PROGRAM CHECK LIST

Item	Description
Undefined Bits	All bits those are marked as “0” (undefined bits) in system registers should be set “0” to avoid unpredicted system errors.
ADC	1. Set ADC input pin I/O direction as input mode 2. Disable pull-up resister of ADC input pin 3. Disable ADC before enter power down (sleep) mode to save power consumption. 4. Set related bit of P4CON to avoid extra power consumption in power down mode 5. Delay 100uS after enable ADC (set ADENB = “1”) to wait ADC circuit ready for conversion 6. Disable ADC (set ADENB = “0”) before enter sleep mode to save power consumption.
Interrupt	Do not enable interrupt before initializing RAM.
Non-Used I/O	Non-used I/O ports should be pull-up or pull-down in input mode, or be set as low in output mode to save current consumption.
Sleep Mode	Enable on-chip pull-up resistors of port 0 to avoid unpredicted wakeup.
Stack Buffer	Be careful of function call and interrupt service routine operation. Don’t let stack buffer overflow or underflow.
System Initial	1. Write 0x7F into STKP register to initial stack pointer and disable global interrupt 2. Clear all RAM. 3. Initialize all system register even unused registers. 4. Initialize all I/O pin direction.
Noisy Immunity	1. Set the watchdog option as “Always On” to protect system crash. 2. Enable “Noise Filter” code option 3. Non-used I/O ports should be set as output low mode 4. Constantly refresh important system registers and variables in RAM to avoid system crash by a high electrical fast transient noise.

14. INSTRUCTION SET TABLE

Field	Mnemonic		Description	C	DC	Z	Cycle
	MOV	A,M	A ← M	-	-	√	1
M	MOV	M,A	M ← A	-	-	-	1
O	B0MOV	A,M	A ← M (bnak 0)	-	-	√	1
V	B0MOV	M,A	M (bank 0) ← A	-	-	-	1
E	MOV	A,I	A ← I	-	-	-	1
	B0MOV	M,I	M ← I, "M" only supports 0x80~0x87 registers (e.g. PFLAG,R,Y,Z…),	-	-	-	1
	XCH	A,M	A ←→M	-	-	-	**1+N**
	B0XCH	A,M	A ←→M (bank 0),	-	-	-	**1+N**
	MOVC		R, A ← ROM [Y,Z]	-	-	-	2
	ADC	A,M	A ← A + M + C, if occur carry, then C=1, else C=0	√	√	√	1
A	ADC	M,A	M ← A + M + C, if occur carry, then C=1, else C=0	√	√	√	**1+N**
R	ADD	A,M	A ← A + M, if occur carry, then C=1, else C=0	√	√	√	1
I	ADD	M,A	M ← M + A, if occur carry, then C=1, else C=0	√	√	√	**1+N**
T	B0ADD	M,A	M (bank 0) ← M (bank 0) + A, if occur carry, then C=1, else C=0	√	√	√	**1+N**
H	ADD	A,I	A ← A + I, if occur carry, then C=1, else C=0	√	√	√	1
M	SBC	A,M	A ← A - M - /C, if occur borrow, then C=0, else C=1	√	√	√	1
E	SBC	M,A	M ← A - M - /C, if occur borrow, then C=0, else C=1	√	√	√	**1+N**
T	SUB	A,M	A ← A - M, if occur borrow, then C=0, else C=1	√	√	√	1
I	SUB	M,A	M ← A - M, if occur borrow, then C=0, else C=1	√	√	√	**1+N**
C	SUB	A,I	A ← A - I, if occur borrow, then C=0, else C=1	√	√	√	1
	AND	A,M	A ← A and M	-	-	√	1
L	AND	M,A	M ← A and M	-	-	√	**1+N**
O	AND	A,I	A ← A and I	-	-	√	1
G	OR	A,M	A ← A or M	-	-	√	1
I	OR	M,A	M ← A or M	-	-	√	**1+N**
C	OR	A,I	A ← A or I	-	-	√	1
	XOR	A,M	A ← A xor M	-	-	√	1
	XOR	M,A	M ← A xor M	-	-	√	**1+N**
	XOR	A,I	A ← A xor I	-	-	√	1
	SWAP	M	A (b3~b0, b7~b4) ←M(b7~b4, b3~b0)	-	-	-	1
P	SWAPM	M	M(b3~b0, b7~b4) ← M(b7~b4, b3~b0)	-	-	-	**1+N**
R	RRC	M	A ← RRC M	√	-	-	1
O	RRCM	M	M ← RRC M	√	-	-	**1+N**
C	RLC	M	A ← RLC M	√	-	-	1
E	RLCM	M	M ← RLC M	√	-	-	**1+N**
S	CLR	M	M ← 0	-	-	-	1
S	BCLR	M.b	M.b ← 0	-	-	-	**1+N**
	BSET	M.b	M.b ← 1	-	-	-	**1+N**
	B0BCLR	M.b	M(bank 0).b ← 0	-	-	-	**1+N**
	B0BSET	M.b	M(bank 0).b ← 1	-	-	-	**1+N**
	CMPRS	A,I	ZF,C ← A - I, If A = I, then skip next instruction	√	-	√	1 + S
B	CMPRS	A,M	ZF,C ← A - M, If A = M, then skip next instruction	√	-	√	1 + S
R	INCS	M	A ← M + 1, If A = 0, then skip next instruction	-	-	-	1 + S
A	INCMS	M	M ← M + 1, If M = 0, then skip next instruction	-	-	-	**1+N+S**
N	DECS	M	A ← M - 1, If A = 0, then skip next instruction	-	-	-	1 + S
C	DECMS	M	M ← M - 1, If M = 0, then skip next instruction	-	-	-	**1+N+S**
H	B0BTS0	M.b	If M(bank 0).b = 0, then skip next instruction	-	-	-	1 + S
	B0BTS1	M.b	If M(bank 0).b = 1, then skip next instruction	-	-	-	1 + S
	JMP	D	PC15/14 ← RomPages1/0, PC13~PC0 ← d	-	-	-	2
	CALL	D	Stack ← PC15~PC0, PC15/14 ← RomPages1/0, PC13~PC0 ← d	-	-	-	2
M	RET		PC ← Stack	-	-	-	2
I	RETI		PC ← Stack, and to enable global interrupt	-	-	-	2
S							
	NOP		No operation	-	-	-	1
	RETLW	I	PC← Stack, A← I	-	-	-	2

➢ ***Note:***

1. ***The "M" is memory including system registers and user defined memory.***
2. ***If branch condition is true then "S = 0", otherwise "S = 1".***
3. ***If "M" is system registers (80h ~ FFh of bank 0) then "N" = 0, otherwise "N" = 1***

15. ELECTRICAL CHARACTERISTIC

ABSOLUTE MAXIMUM RATING

(All of the voltages referenced to Vss)

Supply voltage (Vdd)..... - 0.3V ~ 6.0V
Input in voltage (Vin).....Vss - 0.2V ~ Vdd + 0.2V
Operating ambient temperature (Topr)
SN8P27142P, SN8P27142S, SN8P27143P, SN8P27143S, SN8P27143X, SN8P2714K, SN8P2714S, SN8P2715P, SN8P2715S.....–20°C ~ + 70°C
SN8P27142APD, SN8P27142ASD, SN8P27143APD, SN8P27143ASD, SN8P27143AXD, SN8P2714AKD, SN8P2714ASD, SN8P2715APD, SN8P2715ASD..... –40°C ~ + 85°C
Storage ambient temperature (Tstor) ..... –30°C ~ + 125°C
Power consumption (Pc).....500 mW

STANDARD ELECTRICAL CHARACTERISTIC

(All of voltages referenced to Vss, Vdd = 5.0V, fosc = 4 MHz, ambient temperature is 25℃ unless otherwise note.)

PARAMETER	SYM.	DESCRIPTION		MIN.	TYP.	MAX.	UNIT
Operating voltage	Vdd	Normal mode, Vpp = Vdd		2.4	5.0	5.5	V
		Programming mode, Vpp = 12.5V			5.0		
RAM Data Retention voltage	Vdr			-	1.5	-	V
Internal POR	Vpor	Vdd rise rate to ensure internal power-on reset		-	0.05	-	V/ms
Input Low Voltage	ViL	Input with Schmitt trigger		Vss	-	0.3Vdd	V
Input High Voltage	ViH	Input with Schmitt trigger		0.7Vdd	-	Vdd	V
Reset pin leakage current	Ilekg	Vin = Vdd		-	-	2	uA
I/O port pull-up resistor	Rup	Vin = Vss , Vdd = 3V		-	200	-	KΩ
		Vin = Vss , Vdd = 5V		-	100	-	KΩ
I/O port input leakage current	Ilekg	Pull-up resistor disable, Vin = Vdd		-	-	2	uA
Port2, Port4, Port 5 output source current	IoH	Vop = Vdd - 0.5V		-	12	-	mA
sink current	IoL	Vop = Vss + 0.5V		-	15	-	mA
INTn trigger pulse width	Tint0	INT0 ~ INT1 interrupt request pulse width		2/fcpu	-	-	Cycle
AVREFH input voltage	Varfh	Vdd = 5.0V		2V	-	Vdd	V
AIN0 ~ AIN7 input voltage	Vani	Vdd = 5.0V		0	-	Varfh	V
ADC enable time	Tast	Ready to start convert after set ADENB = “1”		-	100	-	uS
Supply Current	Idd1	Run Mode Fcpu = Fosc/4	Vdd= 5V 4MHz	-	2.5	5	mA
			Vdd= 3V 4MHz	-	1.5	3	mA
			Vdd= 3V 32768Hz	-	15	30	uA
	Idd2	Slow Mode (Internal RC mode)	Vdd= 5V ~ 32KHz	-	5	10	uA
			Vdd= 3V ~ 16KHz	-	25	50	uA
	Idd3	Sleep mode (LVD = LVD_L)	Vdd= 5V	-	1	2	uA
			Vdd= 3V	-	0.6	1	uA
	Idd4	Sleep mode (LVD = LVD_M)	Vdd= 5V	-	2	4	uA
			Vdd= 3V	-	1	2	uA
LVD Voltage	Vdet0	Low voltage reset level		-	2.0	-	V
	Vdet1	Low voltage reset/indicator level Fcpu=1MHz		-	2.4	-	V
	Vdet2	Low voltage indicator level Fcpu=1MHz		-	3.6	-	V
DAC Full-scale Output Current	I_{FSO}	Vdd=5V, RL =150 ohm		-	12	-	mA

16. Development Tools

Development Tool Version

ICE (In circuit emulation)

- **SN8ICE 2K:** Full function emulates SN8P2714/SN8P2715 series

✷ **SN8ICE2K ICE emulation notice**
  a. **Operation voltage of ICE: 3.0V~5.0V.**
  b. **Recommend maximum emulation speed at 5V: 8 MIPS (e.g. 16MHZ crystal and Fcpu = Fhosc/2).**
  c. **Use SN8P2714 / 2715 EV-KIT to emulation LVD.**

✷ **Note: S8KD-2 ICE doesn't support SN8P2714X and SN8P2715 serial emulation.**

OTP Writer

- **Writer 3.0:** Support SN8P2715/SN8P2714 but no Stand-alone mode.
- **Easy Writer V1.0:** OTP programming is controlled by ICE without firmware upgrade suffers. Please refer easy writer user manual for detailed information.
- **MP-EZ Writer V1.0:** Stand-alone operation to support SN8P2715/SN8P2714 mass production

IDE (Integrated Development Environment)

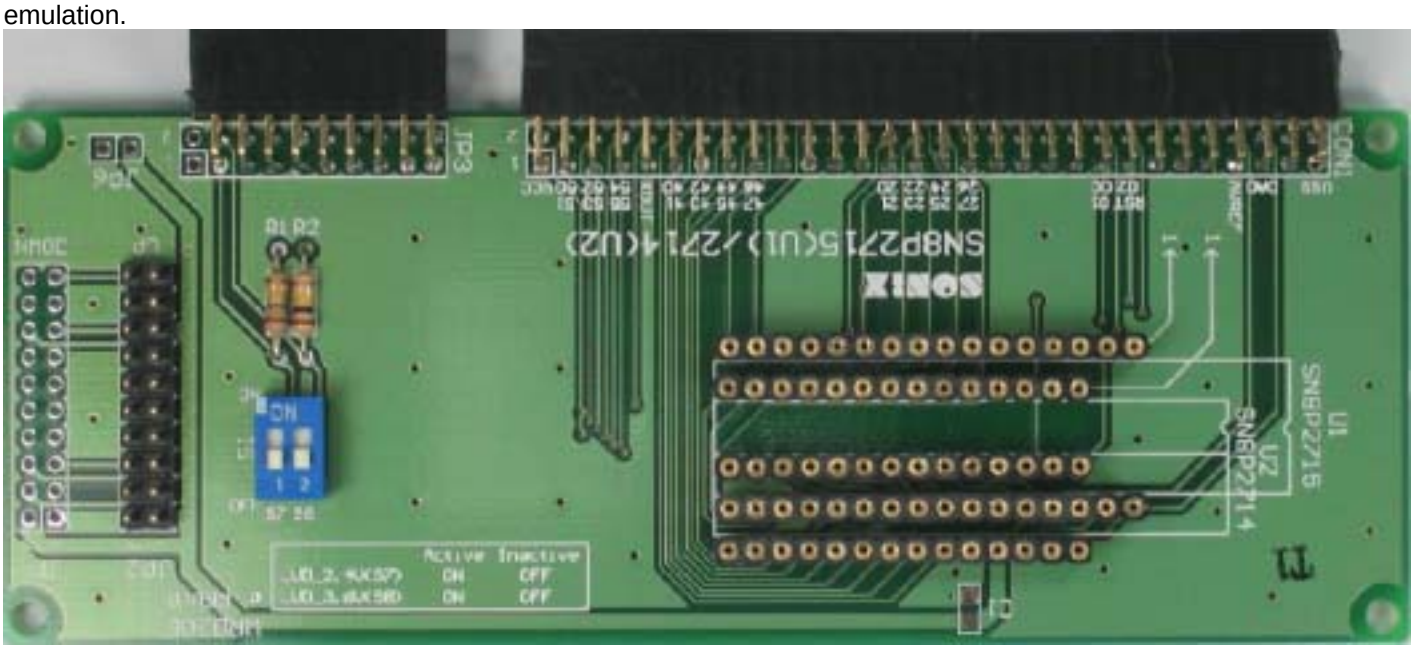
SONiX 8-bit MCU integrated development environment include Assembler, ICE debugger and OTP writer software.

- **For SN8ICE 2K:** M2IDE V1.04 or later
- **For Writer 3.0, Easy Writer and MP-Easy Writer:** M2IDE V1.04 or later
- **SN8IDE V1.99X doesn't support SN8P2714X and SN8P2715.**

SN8P2715/SN8P2714 EV-KIT

PCB DESCRIPTION

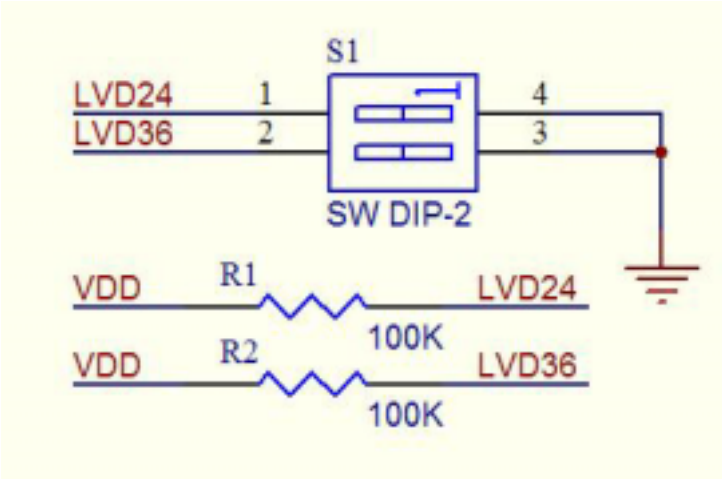
SONiX provides SN8P2715 EV-Kit board for ICE emulation. The EV-Kit provide LVD2.4V/3.6V selection circuit emulation.



CON1 and JP3 :ICE I/O interface.

S1: VD 2.4V and LVD 3.6V trigger control. To emulate LVD 2.4V flag/reset function and LVD 3.6V flag function.

Switch No.	On	Off	Note
S7	LVD 2.4V Active	LVD 2.4V Inactive	Emulate VDD is lower than 2.4V
S8	LVD 3.6V Active	LVD 3.6V Inactive	Emulate VDD is lower than 3.6V



SN8P2715/14 EV-KIT CONNNECT TO SN8ICE 2K

The connection from SN8P2715/14 EV-KIT to SN8ICE 2K is as following.

✷ **SN8ICE2K ICE emulation notice**
a: Operation voltage of ICE: 3.0V~5.0V.
b: Recommend maximum emulation speed at 5V: 8 MIPS(e.g. 16MHZ crystal and Fcpu = Fhosc/2).
c: Use SN8P2715 EV-KIT to emulation LVD.
d: Remove AVREFH/VDD jumper of SN8ICE 2K when Chip declare as SN8P2715/SN8P2714/SN8P27143 .

TRANSITION BOARD FOR OTP PROGTRAMMING

SN8P2715/2715 Rev. B TRANSITION BOARD

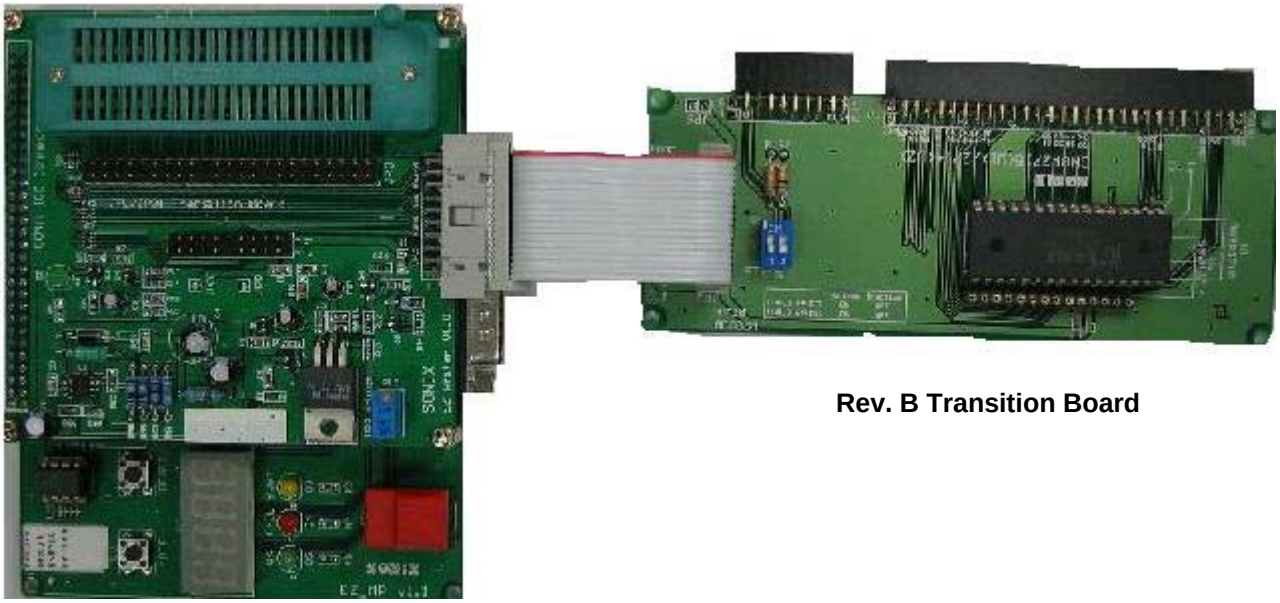
SN8P2715/2714 Rev. B transition board is for SN8P2715/2714 OTP programming including P-DIP 20 pins and P-DIP 18 pins. Rev. B transition board and EV-Kit is the same board.

JP2: Connect to Easy writer or MP-EZ writer through 20 pins cable.

U1: SN8P2715 P-DIP 32 pins socket

U2: SN8P2714 P-DIP 28 pins socket

CONNNECT Rev. B TRANSITION BOARD TO EASY WRITER



Rev. B Transition Board

MP-EZ Writer

OTP Programming Pin to Transition Board Mapping

The pin assignment of Easy and MP EZ Writer transition board socket:

Easy Writer JP1/JP2

VSS	2	1	VDD
CE	4	3	CLK/PGCLK
OE/ShiftDat	6	5	PGM/OTPCLK
D0	8	7	D1
D2	10	9	D3
D4	12	11	D5
D6	14	13	D7
VPP	16	15	VDD
RST	18	17	HLS
ALSB/PDB	20	19	-

JP1 for MP transition board
JP2 for Writer V3.0 transition board

Easy Writer JP3 (Mapping to 48-pin text tool)

DIP1	1	48	DIP48
DIP2	2	47	DIP47
DIP3	3	46	DIP46
DIP4	4	45	DIP45
DIP5	5	44	DIP44
DIP6	6	43	DIP43
DIP7	7	42	DIP42
DIP8	8	41	DIP41
DIP9	9	40	DIP40
DIP10	10	39	DIP39
DIP11	11	38	DIP38
DIP12	12	37	DIP38
DIP13	13	36	DIP36
DIP14	14	35	DIP35
DIP15	15	34	DIP34
DIP16	16	33	DIP33
DIP17	17	32	DIP32
DIP18	18	31	DIP31
DIP19	19	30	DIP30
DIP20	20	29	DIP29
DIP21	21	28	DIP28
DIP22	22	27	DIP27
DIP23	23	26	DIP26
DIP24	24	25	DIP25

JP3 for MP transition board

The pin assignment of Writer V3.0 transition board socket:

GND	**2**	**1**	**VDD**
CE	**4**	**3**	**CLK**
OE	**6**	**5**	**PGM**
D0	**8**	**7**	**D1**
D2	**10**	**9**	**D3**
D4	**12**	**11**	**D5**
D6	**14**	**13**	**D7**
VPP	**16**	**15**	**VDD**
RST	**18**	**17**	**HLS**
	20	**19**	

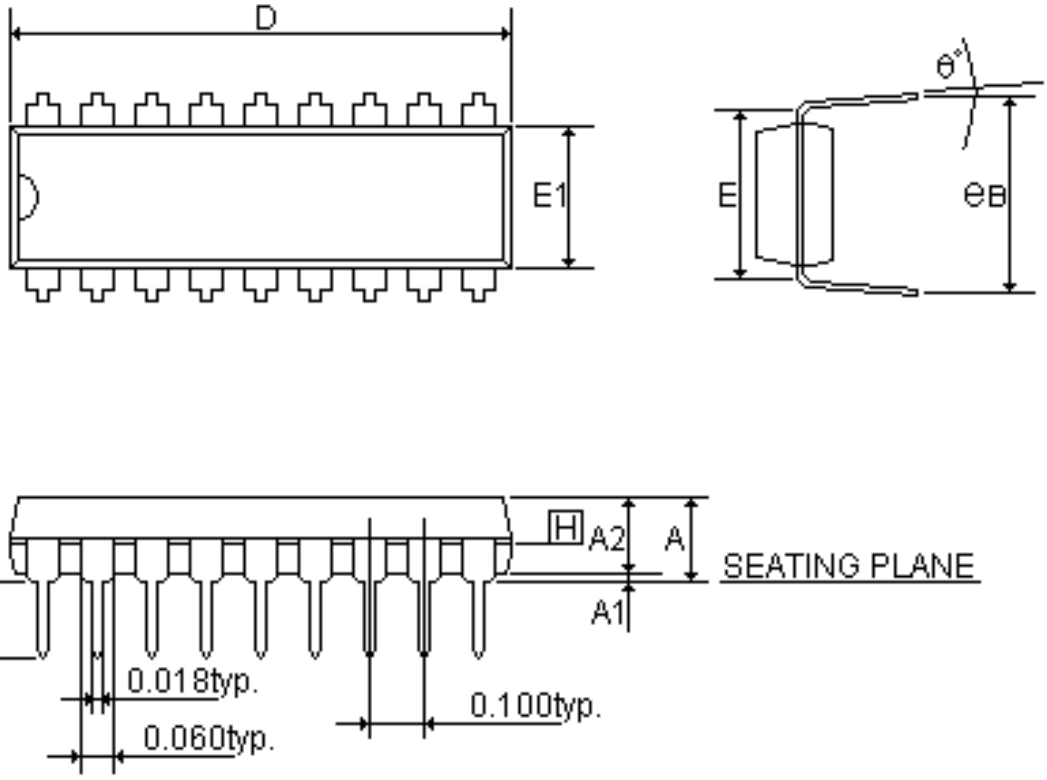
Writer V3.0 JP1 Pin Assignment

SN8P2710 Series Programming Pin Mapping:

OTP Programming Pin of SN8P2710 Series											
Chip Name		SN8P2714		SN8P2715		SN8P27142		SN8P27143			
Easy, MP-EZ Writer And Writer V3.0		OTP IC / JP3 Pin Assignment									
Number	Pin	Number	Pin	Number	Pin	Number	Pin	Number	Pin		
1	VDD	25	VDD	30	VDD	12	VDD	13	VDD		
2	GND	15	VSS	20	VSS	6	VSS	5	VSS		
3	CLK	4	P5.0	6	P5.0	17	P5.0	18	P5.0		
4	CE	-	-	-	-	-	-	-	-		
5	PGM	8	P2.0	10	P2.0	2	P2.0	1	P2.0		
6	OE	3	P5.1	5	P5.1	16	P5.1	17	P5.1		
7	D1	-	-	-	-	-	-	-	-		
8	D0	-	-	-	-	-	-	-	-		
9	D3	-	-	-	-	-	-	-	-		
10	D2	-	-	-	-	-	-	-	-		
11	D5	-	-	-	-	-	-	-	-		
12	D4	-	-	-	-	-	-	-	-		
13	D7	-	-	-	-	-	-	-	-		
14	D6	-	-	-	-	-	-	-	-		
15	VDD	25	VDD	30	VDD	12	VDD	13	VDD		
16	VPP	26	RST	31	RST	13	RST	14	RST		
17	HLS	-	-	-	-	-	-	-	-		
18	RST	-	-	-	-	-	-	-	-		
19	-	-	-	-	-						
20	ALSB/PDB	9	P2.1	11	P2.1						

17. PACKAGE INFORMATION

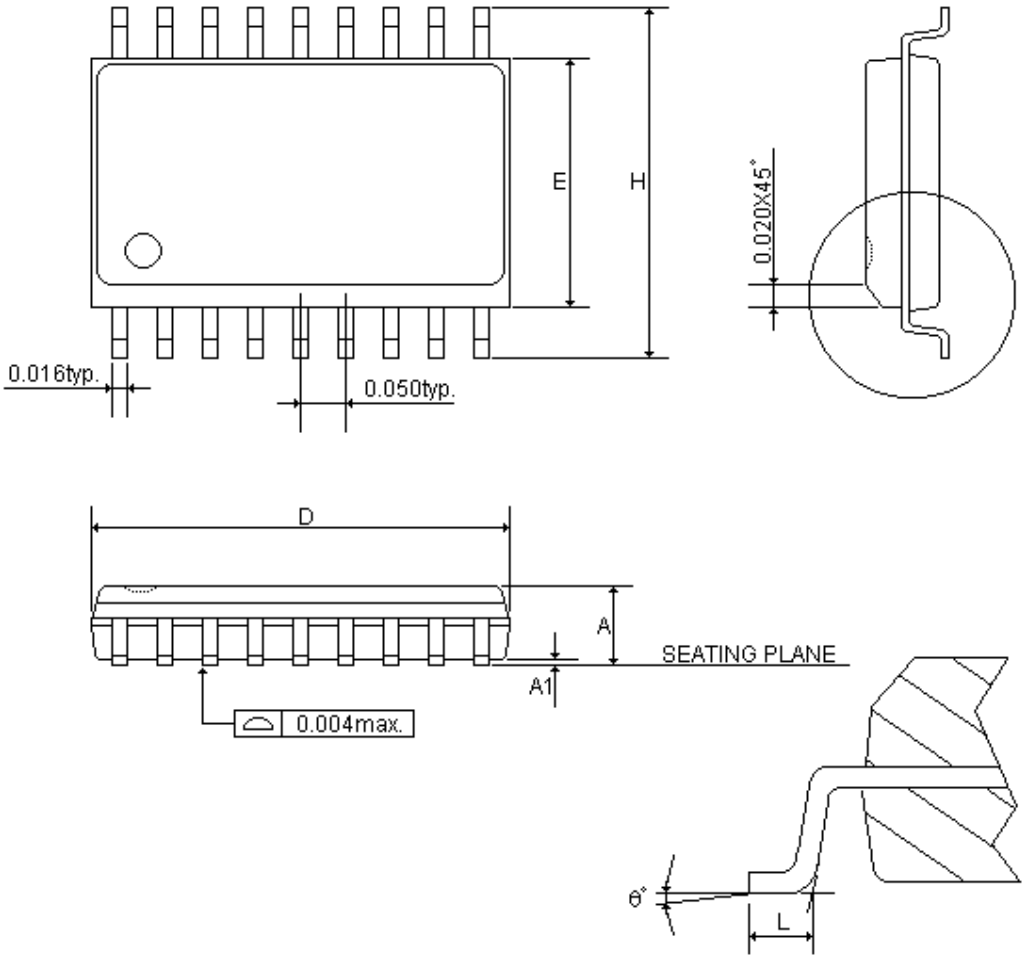
P-DIP18 PIN



Symbols	MIN.	NOR.	MAX.
A	-	-	0.210
A1	0.015	-	-
A2	0.125	0.130	0.135
D	0.880	0.900	0.920
E		0.300BSC.	
E1	0.245	0.250	0.255
L	0.115	0.130	0.150
e B	0.335	0.355	0.375
θ °	0	7	15

UNIT : INCH

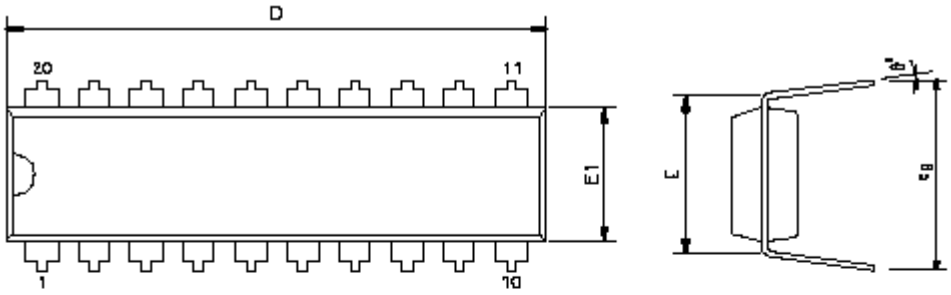
SOP18 PIN

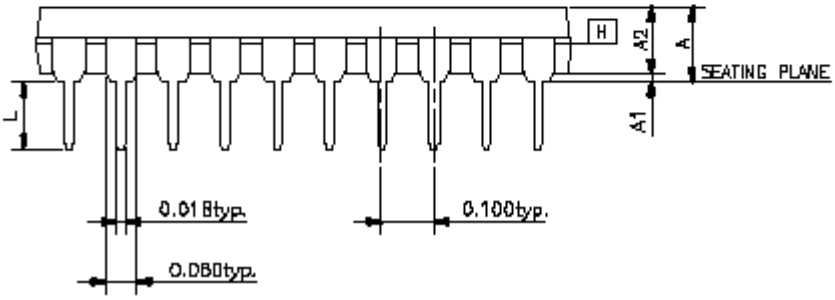


Symbols	MIN.	MAX.
A	0.093	0.104
A1	0.004	0.012
D	0.447	0.463
E	0.291	0.299
H	0.394	0.419
L	0.016	0.050
θ °	0	8

UNIT : INCH

P-DIP 20 PIN





SYMBOLS	MIN.	NOR.	MAX.
A	–	–	0.210
A1	0.015	–	–
A2	0.125	0.130	0.135
D	0.98	1.030	1.060
E		0.300 BSC.	
E1	0.245	0.250	0.255
L	0.115	0.130	0.150
e_B	0.335	0.355	0.375
θ°	0	7	15

UNIT : INCH

NOTES:

1.JEDEC OUTLINE : MS-001 AD

2."D","E1" DIMENSIONS DO NOT INCLUDE MOLD FLASH OR PROTRUSIONS.MOLD FLASH OR PROTRUSIONS SHALL NOT EXCEED .010 INCH.

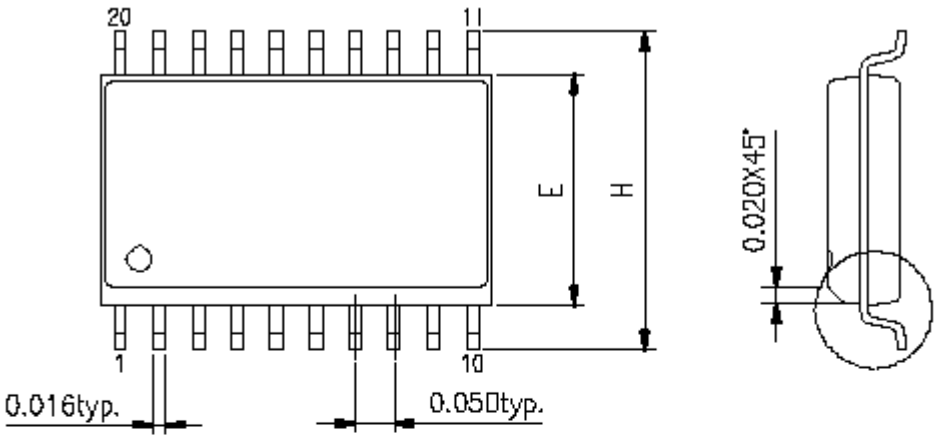
3.eB IS MEASURED AT THE LEAD TIPS WITH THE LEADS UNCONSTRAINED.

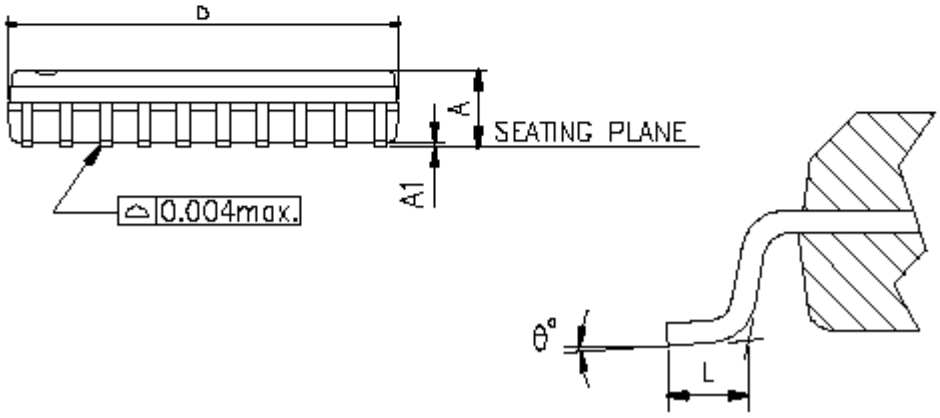
4.POINTED OR ROUNDED LEAD TIPS ARE PREFERRED TO EASE INSERTION.

5.DISTANCE BETWEEN LEADS INCLUDING DAM BAR PROTRUSIONS TO BE .005 INCH MININUM.

6.DATUM PLANE [H] COINCIDENT WITH THE BOTTOM OF LEAD, WHERE LEAD EXITS BODY.

SOP 20 PIN



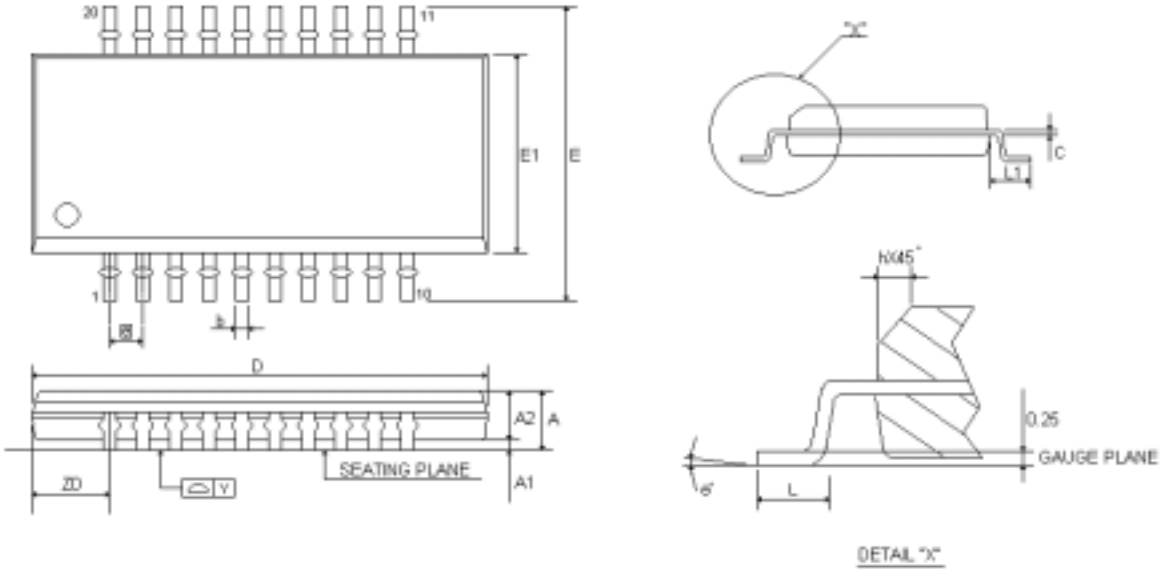


SYMBOLS	MIN.	MAX.
A	0.093	0.104
A1	0.004	0.012
D	0.496	0.508
E	0.291	0.299
H	0.394	0.419
L	0.016	0.050
θ°	0	8

UNIT : INCH

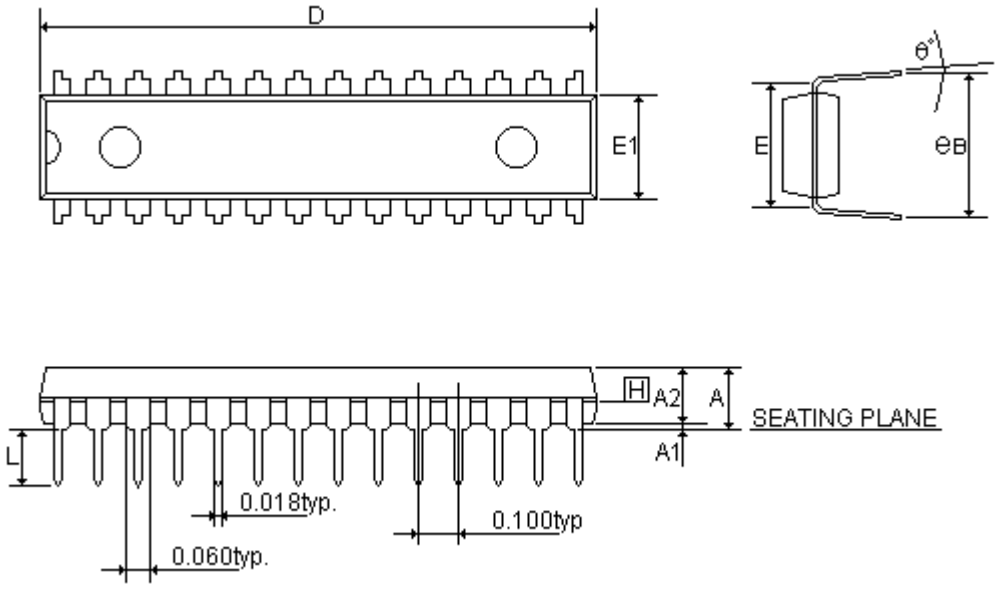
NOTES:
1.JEDEC OUTLINE : MS-013 AC
2.DIMENSIONS "D" DOES NOT INCLUDE MOLD FLASH, PROTRUSIONS OR GATE BURRS.MOLD FLASH, PROTRUSIONS AND GATE BURRS SHALL NOT EXCEED .15mm (.006in) PER SIDE.
3.DIMENSIONS "E" DOES NOT INCLUDE INTER-LEAD FLASH, OR PROTRUSIONS. INTER-LEAD FLASH AND PROTRUSIONS SHALL NOT EXCEED .25mm (.010in) PER SIDE.

SSOP20 PIN



Symbols	DIMENSION (MM)			DIMENSION (MIL)		
	MIN,	NOM.	MAX.	MIN.	NOM.	MAX.
A	1.35	1.60	1.75	53	63	69
A1	0.10	0.15	0.25	4	6	10
A2	-	-	1.50	-	-	59
b	0.20	0.254	0.30	8	10	12
b1	0.20	0.254	0.28	8	11	11
C	0.18	0.203	0.25	7	8	10
C1	0.18	0.203	0.23	7	8	9
D	8.56	8.66	8.74	337	341	344
E	5.80	6.00	6.20	228	236	244
E1	3.80	3.90	4.00	150	154	157
e	0.635 BSC			25 BSC		
h	0.25	0.42	0.50	10	17	20
L	0.40	0.635	1.27	16	25	50
L1	1.00	1.05	1.10	39	41	43
ZD	1.50 REF			58 REF		
Y	-	-	0.10	-	-	4
θ °	0°	-	8°	0°	-	8°

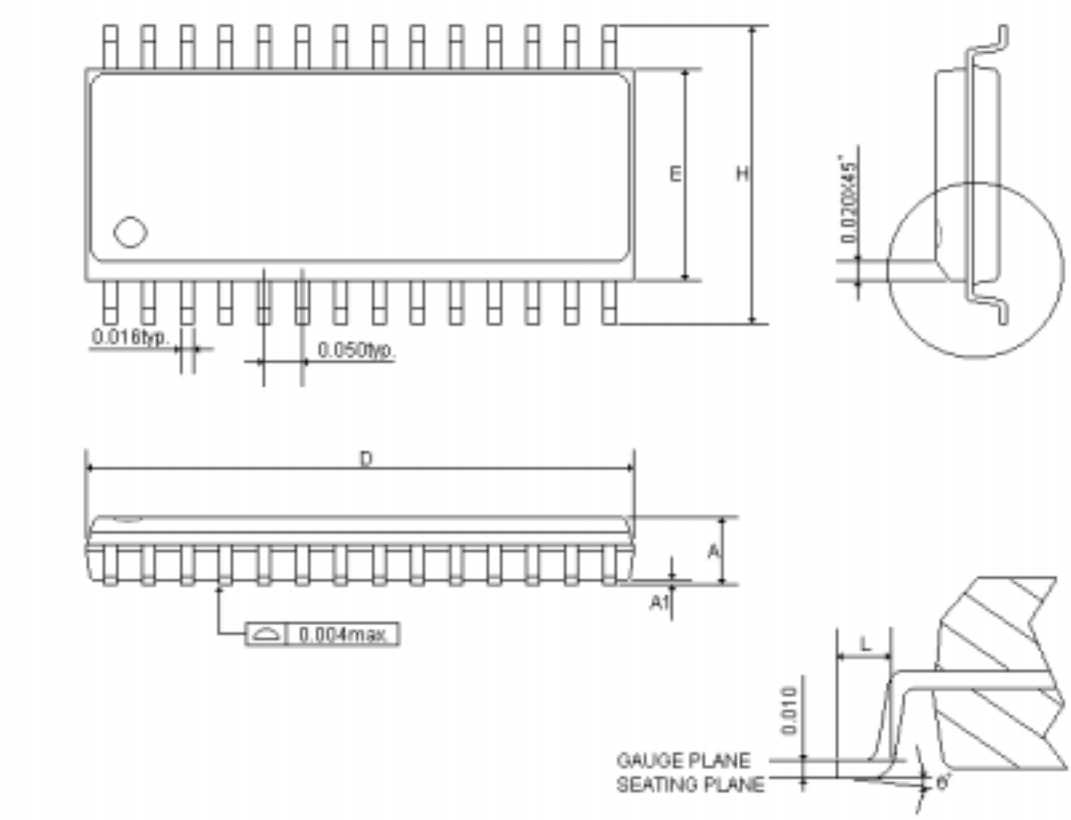
SK-DIP28 PIN



Symbols	MIN.	NOR.	MAX.
A	-	-	0.210
A1	0.015	-	-
A2	0.114	0.130	0.135
D	1.390	1.390	1.400
E		0.310BSC.	
E1	0.283	0.288	0.293
L	0.115	0.130	0.150
e_B	0.330	0.350	0.370
θ °	0	7	15

UNIT : INCH

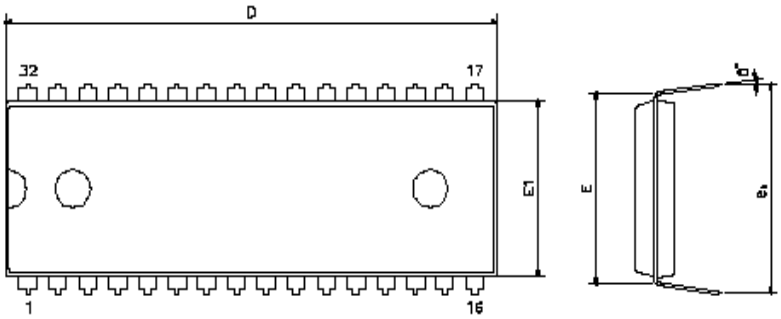
SOP28 PIN



Symbols	MIN.	MAX.
A	0.093	0.104
A1	0.004	0.012
D	0.697	0.713
E	0.291	0.299
H	0.394	0.419
L	0.016	0.050
θ °	0	8

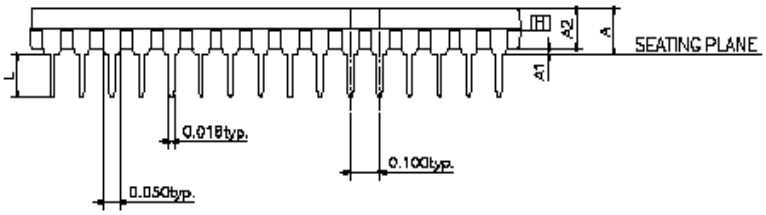
UNIT : INCH

P-DIP 32 PIN



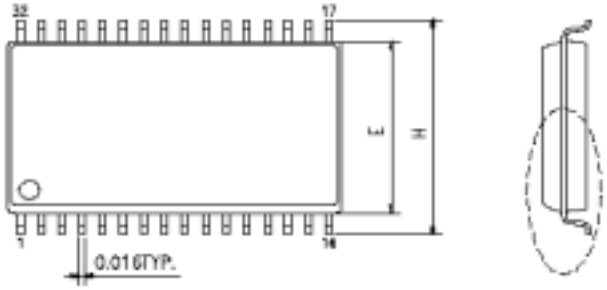
SYMBOLS	MIN.	NOR.	MAX.
A	–	–	0.220
A1	0.015	–	–
A2	0.150	0.155	0.160
D	1.645	1.650	1.660
E		0.600 BSC	
E1	0.540	0.545	0.550
L	0.115	0.130	0.150
e_B	0.630	0.650	0.670
θ°	0	7	15

UNIT : INCH



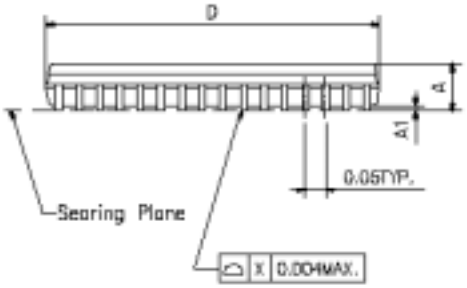
NOTE:
1.JEDEC OUTLINE : MS-011 AC

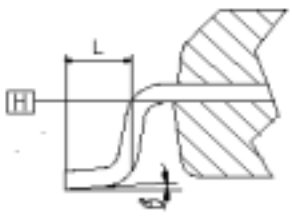
SOP 32 PIN



SYMBOLS	MIN.	MAX.
A	–	0.120
A1	0.004	0.014
D	0.799	0.815
E	0.437	0.450
H	0.530	0.580
L	0.016	0.050
θ°	0°	10°

UNIT : INCH





NOTE:
1.JEDEC OUTLINE:MO-088 AB
2.DATUM PLANE ⊞IS LOCATED AT THE BOTTOM OF THE MOLD PARTING LINE COINCIDENT WITH WHERE THE LEAD EXITS THE BODY.
3.DIMENSIONS E AND D DO NOT INCLUDE MOLD PROTRUSION. ALLOWABLE PROTRUSION IS 10 MIL PER SIDE. DIMENSIONS E AND D DO INCLUDE MOLD MISMATCH AND ARE DETERMINED AT DATUM PLANE ⊞
4.DIMENSION b DOES NOT INCLUDE DAMBAR PROTRUSION.

Main Office:
Address: 9F, NO. 8, Hsien Cheng 5th St, Chupei City, Hsinchu, Taiwan R.O.C.
Tel: 886-3-551 0520
Fax: 886-3-551 0523

Taipei Office:
Address: 15F-2, NO. 171, Song Ted Road, Taipei, Taiwan R.O.C.
Tel: 886-2-2759 1980
Fax: 886-2-2759 8180

Hong Kong Office:
Address: Flat 3 9/F Energy Plaza 92 Granville Road, Tsimshatsui East Kowloon.
Tel: 852-2723 8086
Fax: 852-2723 9179

Technical Support by Email:
Sn8fae@sonix.com.tw