SN8P1600 Series

USER'S MANUAL

SN8P1602
SN8P1603
SN8P1604

No Recommend in New Design

SONiX 8-Bit Micro-Controller

AMENDENT HISTORY

Version	Date	Description
VER 1.90	Sep. 2002	V1.9 first issue
VER 1.91	Sep. 2002	Correct some V1.9 typing errors
VER 1.92	Oct. 2002	1. Correct some template code errors 2. Modify description of code option 3. Modify approval form section 4. Modify description of TC1 timer and add more explanation about PWM function
VER 1.93	Feb. 2003	1. Extend chip operating temperature from “0°C ~ +70°C” to “-20°C ~ +70°C”. 2. Change the description of ADD M,A instruction from “M ← M+A” to “M ← A+M” 3. Change “ACC can’t be access by “B0MOV” instruction” to “ACC can’t be access by “B0MOV” instruction during the instant addressing mode”. 4. Correct the description of STKn. 5. Correct the bit definition of INTEN register. 6. Correct the description of “TC1 CLOCK FREQUENCY OUTPUT” section. 7. Correct an error of template code: “b0bclr FWDRST” → “b0bset FWDRST”. 8. Add a notice about OSCM register access cycle. 9. Correct the table of “STANDARD ELECTRICAL CHARACTERISTICS”.
VER 1.94	Sep. 2003	1. Add new section about checksum calculate must avoid 04H~07H 2. Reserved Last 16 word ROM addresses 3. Remove register bit description 4. Modify TC0M description. 5. Modify TC1M description. 6. Modify PWM description 7. Modify slow mode current. 8. Change code option to chapter2 9. Adjust Electrical characteristic page 10. Remove approval sheet. 11. Remove PCB layout notice section. 12. Modify the description of INTRQ register.

Table of Content

1 PRODUCT OVERVIEW

GENERAL DESCRIPTION

The SN8P1600 series is an 8-bit micro-controller utilized CMOS technology and featured with low power consumption and high performance by its unique electronic structure.

SN8P1602/ SN8P1603 is designed with the excellent IC structure including the program memory up to 1K-word OTP ROM, data memory of 48-bytes RAM, one 8-bit timer (TC0), a watchdog timer, two interrupt sources (TC0, INT0), and 4-level stack buffers.

More expansion functions come with SN8P1604, such like 4K-word OTP ROM, more data memory of 128-byte RAM, 8-bit timer named TC1, and buzzer function for different application. More details listed below.

Besides, user can choose desired oscillator configuration for the controller. There are four external oscillator configurations to select for generating system clock, including High/Low speed crystal, ceramic resonator or cost-saving RC oscillator. SN8P1600 also includes an internal RC oscillator for slow mode controlled by program.

SELECTION TABLE

CHIP	ROM	RAM	I/O	Stack	Timer		LVD	PWM	Wakeup	Package
					TC0	TC1		Buzzer	Pin no.	
SN8P1602	1K*16	48	14	4	V	-	On/Off	-	6	DIP18/SOP18/SSOP20
SN8P1603							On			
SN8P1604	4K*16	128	22		-	V	On/off	1	10	SKDIP28/SOP28

Notice: The SN8P1603 always turn the LVD (low voltage detect) on.

FEATURES

- **Memory configuration**
  OTP ROM size: 1K * 16-bit. (SN8P1602/1603)
  RAM size: 48 * 8-bit. (SN8P1602/1603)
  OTP ROM size: 4K * 16-bit. (SN8P1604)
  RAM size: 128 * 8-bit. (SN8P1604)

- **I/O pin configuration**
  (SN8P1602/1603 14 pins, SN8P1604 22 pins)
  Input only: P0
  Bi-directional: P1, P2, P5
  Wakeup: P0, P1
  Pull-up resisters: P0, P1, P2, P5 (SN8P1604 only)
  External interrupt: P0

- **One 8-bit timer counters.**
  (TC1 for SN8P1604, TC0 for others)
- **On chip watchdog timer.**
- **Four levels stack buffer.**

- **56 powerful instructions**
  Four clocks per instruction cycle
  All of instructions are one word length.
  Most of instructions are one cycle only.
  All ROM area lookup table function (MOVC)

- **Two interrupt sources**
  One internal interrupt: TC0. (SN8P1602/ 1603)
  One internal interrupt: TC1. (SN8P1604)
  One external interrupt: INT0.

- **One channel 8-bits PWM or Buzzer output.**
  (SN8P1604 only)

- **Dual clock system offers three 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

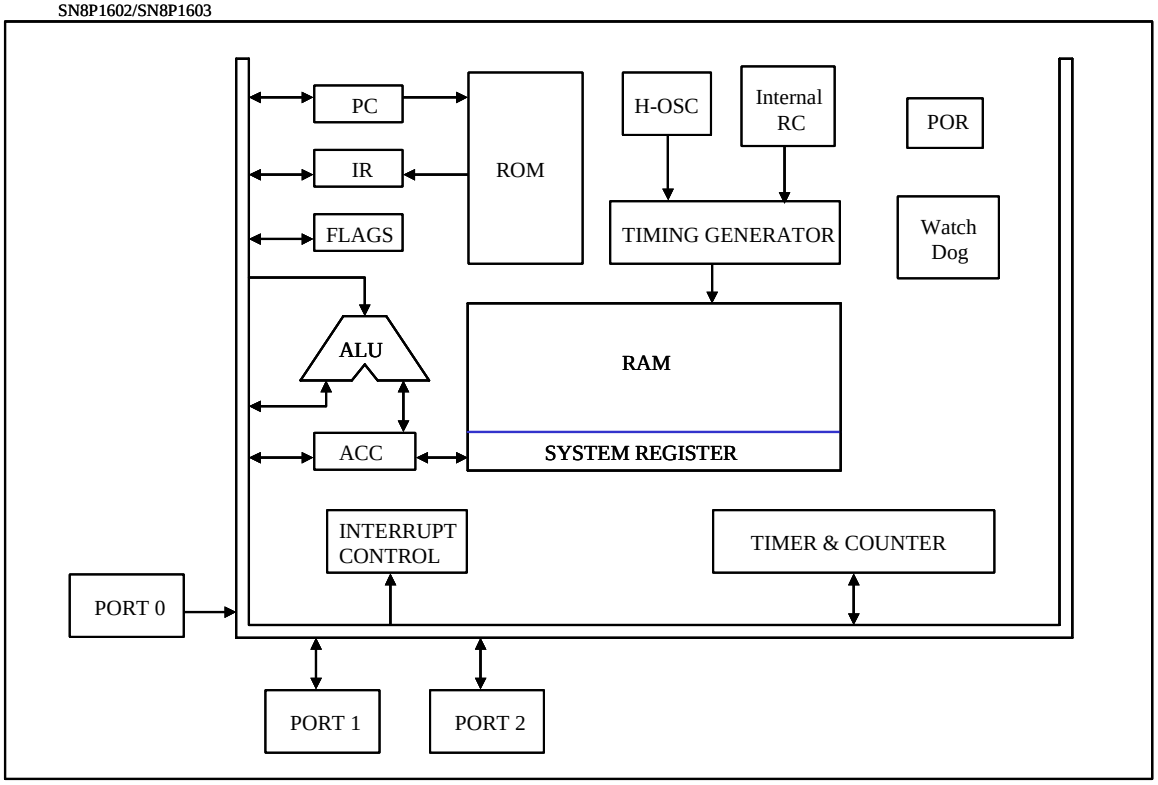
- **Package (Chip form support)**
  P-DIP18 (SN8P1602/ 1603)
  SOP18 (SN8P1602/1603)
  SSOP20 (SN8P1602/1603)

  SKDIP28 (SN8P1604)
  SOP28 (SN8P1604)

SYSTEM BLOCK DIAGRAM

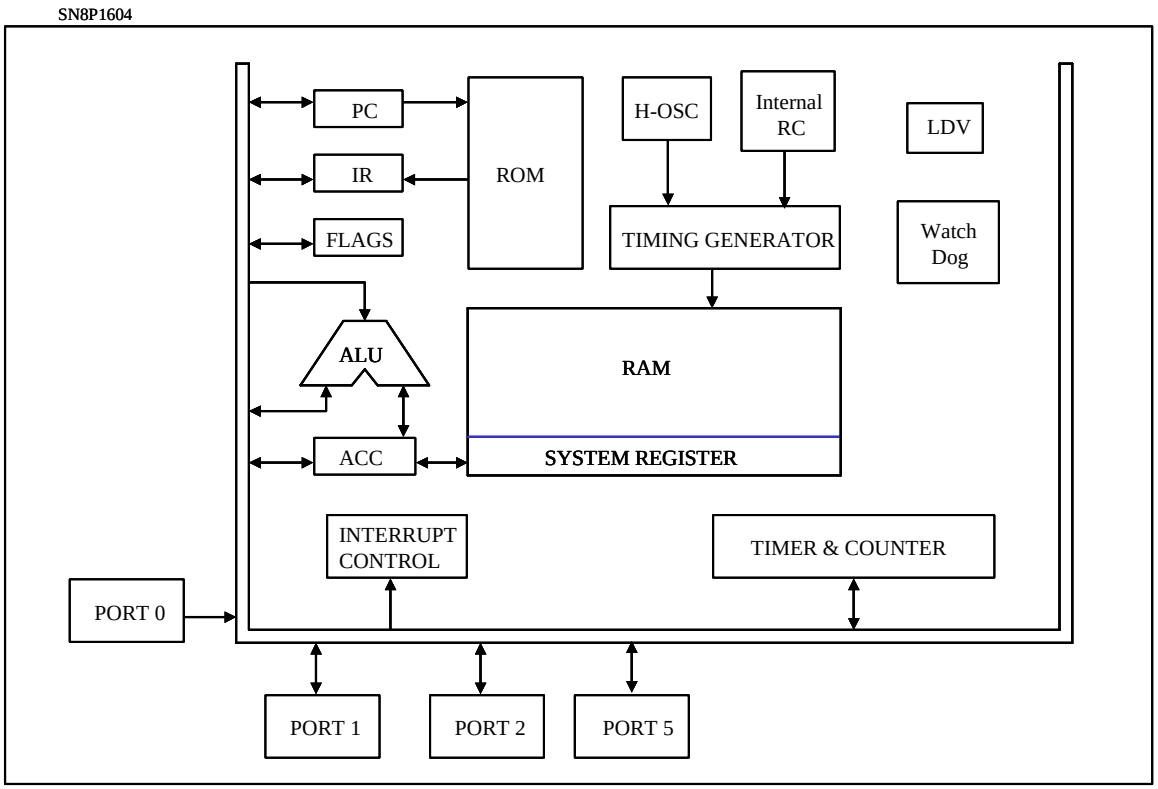
➢ ***SN8P1602/1603***

SN8P1602/SN8P1603



➢ ***SN8P1604***

SN8P1604



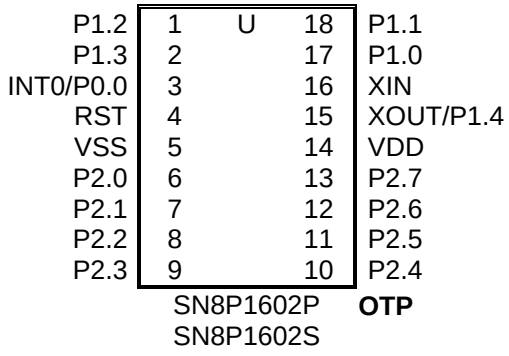
PIN ASSIGNMENT

Part Number Description : **OTP Type: SN8P16XXY, MASK Type: SN8A16XXAY**

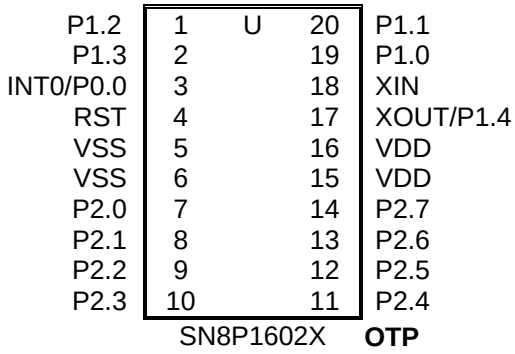
Y = Q: QFP , P: PDIP , K: SKDIP , S: SOP , X: SSOP

OTP Type:

SN8P1602P / SN8P1603P (DIP 18 pins)
SN8P1602S / SN8P1603S (SOP 18 pins)

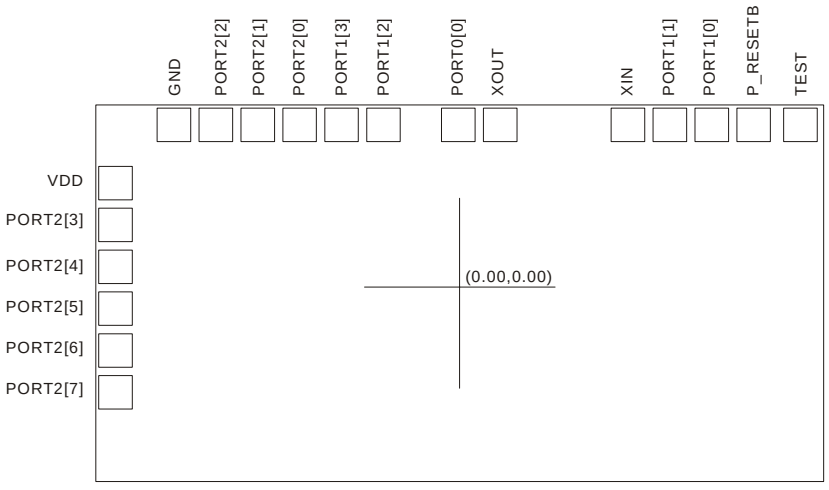


SN8P1602X / SN8P1603X (SSOP 20 pins)



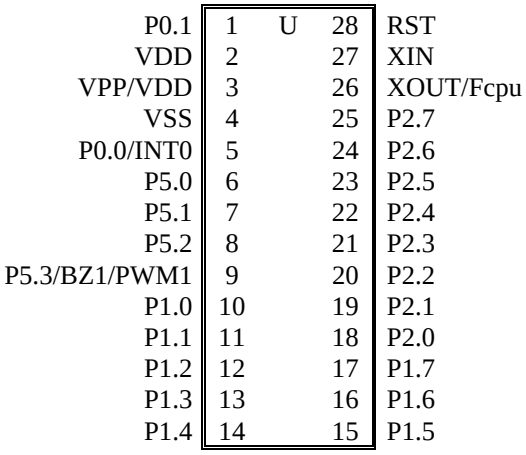
MASK Type:

SN8A1602A: Support dice form only



OTP Type:

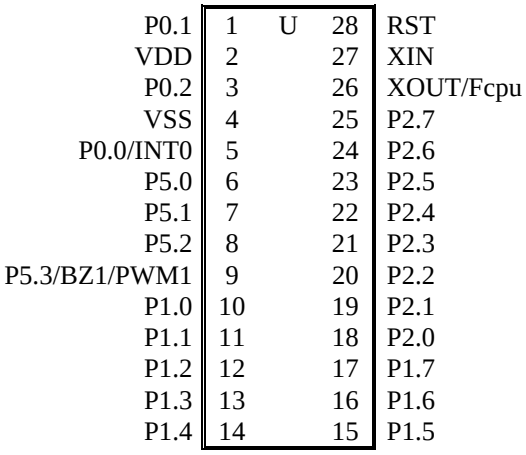
SN8P1604K (SKDIP 28 pins)
SN8P1604S (SOP 28 pins)



SN8P1604K
SN8P1604S

MASK Type:

SN8A1604AK (SKDIP 28 pins)
SN8A1604AS (SOP 28 pins)



SN8A1604AK
SN8A1604AS

Notice: Different pins between MASK and OTP:

- Pin 3 of SN8A1604A is P0.2 but it is VPP in SN8P1604. Pull up P0.2 to VDD if no use to avoid extra power consumption.

PIN DESCRIPTIONS

➢ ***SN8P1602/1603***

PIN NAME	TYPE	DESCRIPTION
VDD, VSS	P	Power supply input pins. Place the 0.1μF bypass capacitor between the VDD and VSS pin.
VPP/RST	I	System reset pin. Schmitt trigger structure, low active, normal stay to “high”.
XIN	I	External oscillator input pin. RC mode input pin.
XOUT/P1.4	I/O	External oscillator output pin. In RC mode is P1.4 I/O.
P0.0 / INT0	I	Input / Interrupt (Schmitt trigger) / Wakeup function.
P1.0 ~ P1.4	I/O	Bi-direction pins with sleep mode wakeup function.
P2.0 ~ P2.7	I/O	Bi-direction pins.

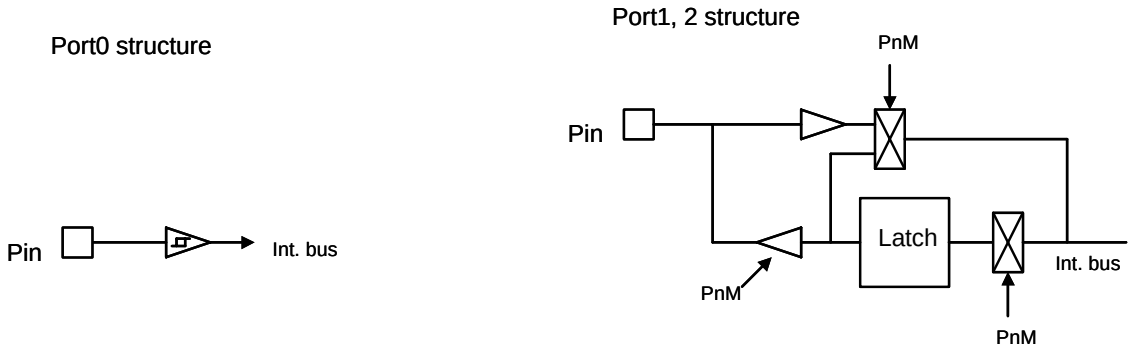
Notice : The SN8P1602/3 do not have the pull-up resistor in the input port. The user must use the external pull-up resistor.

➢ ***SN8P1604***

PIN NAME	TYPE	DESCRIPTION
VDD, VSS	P	Power supply input pins. Place the 0.1μF bypass capacitor between the VDD and VSS pin.
VPP/VDD	P	OTP programming pin. Keep connect to VDD during normal mode.
RST	I	System reset pin. Schmitt trigger structure, low active, normal stay to “high”.
XIN	I	External oscillator input pin. RC mode input pin.
XOUT/Fcpu	I/O	External oscillator output pin. RC Mode as the Fcpu output
P0.0/INT0	I	Input / Interrupt (Schmitt trigger) / Wakeup function / Built-in pull-up resisters.
P0.1	I	Input / Wakeup function. / Built-in pull-up resister.
P1.0 ~ P1.7	I/O	Bi-direction pins with sleep mode Wakeup function. Built-in pull-up resisters.
P2.0 ~ P2.7	I/O	Bi-direction pins / Built-in pull-up resisters.
P5.0 ~ P5.3	I/O	Bi-direction pin, P5.3 as TC1 output for PWM and Buzzer function/Built-in pull-up resisters.

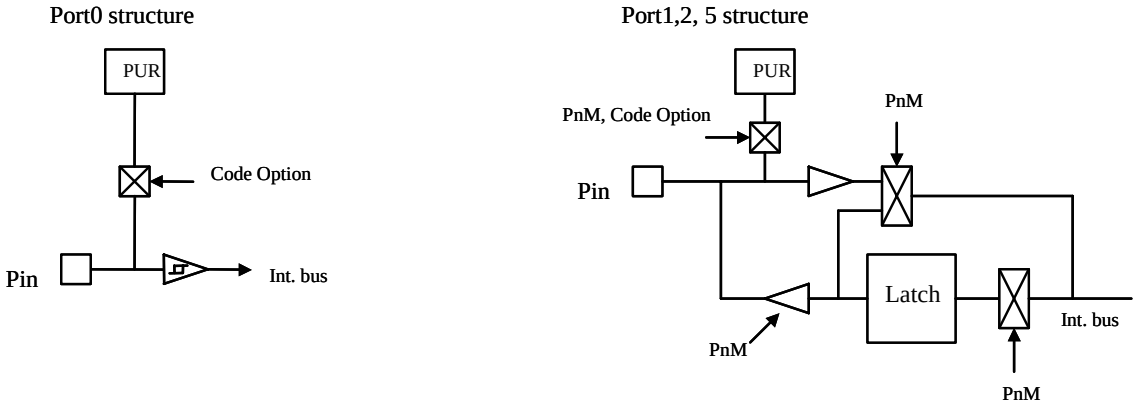
PIN CIRCUIT DIAGRAMS

➢ ***SN8P1602/1603***



➢ ***Note: All of the latch output circuits are push-pull structures.***

➢ ***SN8P1604***



➢ ***Note: The internal pull-up resistor of the SN8P1604 can be enabled by the code option.***

2 CODE OPTION TABLE

SN8P1602

Code Option	Content	Function Description
High_Clk	RC	Low cost RC for external high clock oscillator
	32K X'tal	Low frequency, power saving crystal (e.g. 32.768K) for external high clock oscillator
	12M X'tal	High speed crystal /resonator (e.g. 12M) for external high clock oscillator
	4M X'tal	Standard crystal /resonator (e.g. 3.58M) for external high clock oscillator
High_Clk / 2	Enable	External high clock divided by two, Fosc = high clock / 2
	Disable	Fosc = high clock
OSG	Enable	Enable Oscillator Safe Guard function
	Disable	Disable Oscillator Safe Guard function
Watch_Dog	Enable	Enable Watch Dog function
	Disable	Disable Watch Dog function
LVD	Enable	Enable the low voltage detect
	Disable	Disable the low voltage detect
Security	Enable	Enable ROM code Security function
	Disable	Disable ROM code Security function

SN8P1603----------The LVD always turn on to improve the power on reset and brownout reset performance

Code Option	Content	Function Description
High_Clk	RC	Low cost RC for external high clock oscillator
	32K X'tal	Low frequency, power saving crystal (e.g. 32.768K) for external high clock oscillator
	12M X'tal	High speed crystal /resonator (e.g. 12M) for external high clock oscillator
	4M X'tal	Standard crystal /resonator (e.g. 3.58M) for external high clock oscillator
High_Clk / 2	Enable	External high clock divided by two, Fosc = high clock / 2
	Disable	Fosc = high clock
OSG	Enable	Enable Oscillator Safe Guard function
	Disable	Disable Oscillator Safe Guard function
Watch_Dog	Enable	Enable Watch Dog function
	Disable	Disable Watch Dog function
Security	Enable	Enable ROM code Security function
	Disable	Disable ROM code Security function

3 ADDRESS SPACES

PROGRAM MEMORY (ROM)

OVERVIEW

The SN8P1600 provides the program memory up to 1024 * 16-bit (4096 *16-bit for SN8P1604) to be addressed and is able to fetch instructions through 10-bit wide PC (Program Counter). It can look up ROM data by using ROM code registers (R, Y, Z).

- 1-word reset vector addresses
- 1-word interrupt vector addresses
- 1K words general purpose area (SN8P1602/1603)
- 4K words general purpose area (SN8P1604)
- 5-words reserved area

All of the program memory is partitioned into three coding areas. The 1st area is located from 00H to 03H(The Reset vector area), the 2nd area is a reserved area 04H ~07H, the 3rd area is for the interrupt vector and the user code area from 0008H to 0FFEH. The address 08H is the interrupt enter address point.

SN8P1604	SN8P1602/SN8P1603	ROM	
0000H	0000H	***Reset vector***	User reset vector
0001H	0001H	***General purpose area***	Jump to user start address
0002H	0002H		Jump to user start address
0003H	0003H		Jump to user start address
0004H	0004H	***Reserved***	
0005H	0005H		
0006H	0006H		
0007H	0007H		
0008H	0008H	***Interrupt vector***	User interrupt vector
0009H	0009H	***General purpose area***	User program
.	.		
.	.		
000FH	000FH		
0010H	0010H		
0011H	0011H		
.	.		
.	.		
0FFEH	03FEH		End of user program
0FFFH	03FFH	***Reserved***	

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.**

```
CHIP SN8P1602
                ORG     0           ; 0000H
                JMP     START       ; Jump to user program address.
                .                   ; 0004H ~ 0007H are reserved

                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 executes, the program counter (PC) value is stored in stack buffer and jump to 0008h of program memory to execute the vectored interrupt. Users have to define the interrupt vector. 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.**

```
CHIP SN8P1602
.DATA           PFLAGBUF
.CODE
                ORG     0           ; 0000H
                JMP     START       ; Jump to user program address.
                .                   ; 0004H ~ 0007H are reserved

                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
                .
                .
                B0MOV   A, PFLAGBUF
                B0MOV   PFLAG, A    ; Restore PFLAG register from buffer
                B0XCH   A, ACCBUF   ; B0XCH doesn't change C, Z flag
                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.**

```
CHIP SN8P1602
.DATA       PFLAGBUF
.CODE
            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
            .
            .
            B0MOV   A, PFLAGBUF
            B0MOV   PFLAG, A    ; Restore PFLAG register from buffer
            B0XCH   A, ACCBUF   ; B0XCH doesn't change C, Z flag
            RETI                ; End of interrupt service routine

            ENDP                ; End of program
```

➢ ***Remark: 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. The 0004H~0007H are reserved. Users have to skip 0004H~0007H addresses. It is very important and necessary.

➢ ***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.***

CHECKSUM CALCULATION

The ROM addresses 0004H~0007H and last address are reserved area. User should avoid these addresses (0004H~0007H and last address) when calculate the Checksum value.

➲ ***Example:***
The demo program shows how to avoid 0004H~0007H when calculated Checksum from 00H to the end of user's code

```
                MOV     A,#END_USER_CODE$L
                B0MOV   END_ADDR1,A         ;save low end address to end_addr1
                MOV     A,#END_USER_CODE$
                M
                B0MOV   END_ADDR2,A         ;save middle end address to end_addr2
                CLR     Y                   ;set Y to ooH
                CLR     Z                   ;set Z to 00H
@@:
                CALL    YZ_CHECK            ;call function of check yz value
                MOVC                        ;
                B0BSET  FC                  ;clear C glag
                ADD     DATA1,A             ;add A to Data1
                MOV     A,R
                ADC     DATA2,A             ;add R to Data2
                JMP     END_CHECK           ;check if the YZ address =    the end of code
AAA:
                INCMS   Z                   ;Z=Z+1
                JMP     @B                  ;if Z!= 00H calculate to next address
                JMP     Y_ADD_1             ;if Z=00H increase Y
END_CHECK:
                MOV     A,END_ADDR1
                CMPRS   A,Z                 ;check if Z = low end address
                JMP     AAA                 ;if Not jump to checksum calculate
                MOV     A,END_ADDR2
                CMPRS   A,Y                 ;if Yes, check if Y = middle end address
                JMP     AAA                 ;if Not jump to checksum calculate
                JMP     CHECKSUM_END        ;if Yes checksum calculated is done.
YZ_CHECK:                                   ;check if YZ=0004H
                MOV     A,#04H
                CMPRS   A,Z                 ;check if Z=04H
                RET                         ;if Not return to checksum calculate
                MOV     A,#00H
                CMPRS   A,Y                 ;if Yes, check if Y=00H
                RET                         ;if Not return to checksum calculate
                INCMS   Z                   ;if Yes, increase 4 to Z
                INCMS   Z
                INCMS   Z
                INCMS   Z
                RET                         ;set YZ=0008H then return
Y_ADD_1:
                INCMS   Y                   ;increase Y
                NOP
                JMP     @B                  ;jump to checksum calculate
CHECKSUM_END:
                ……..
                ……..
END_USER_CODE:                              ;Label of program end
```

GENERAL PURPOSE PROGRAM MEMORY AREA

The 1017/4089-word at ROM locations 0009H~03FEH/0FFEH are used as general-purpose memory. The area is stored instruction's op-code and look-up table data. The SN8P1600 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" when the PCL overflows (from 0FFH to 000H).

LOOK-UP TABLE DESCRIPTION

In the ROM's data lookup function, Y register is pointed to the bit 8~bit 15 and Z register to the bit 0~bit 7 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 :**

```
        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 maybe wastes some ROM size.

```
@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.

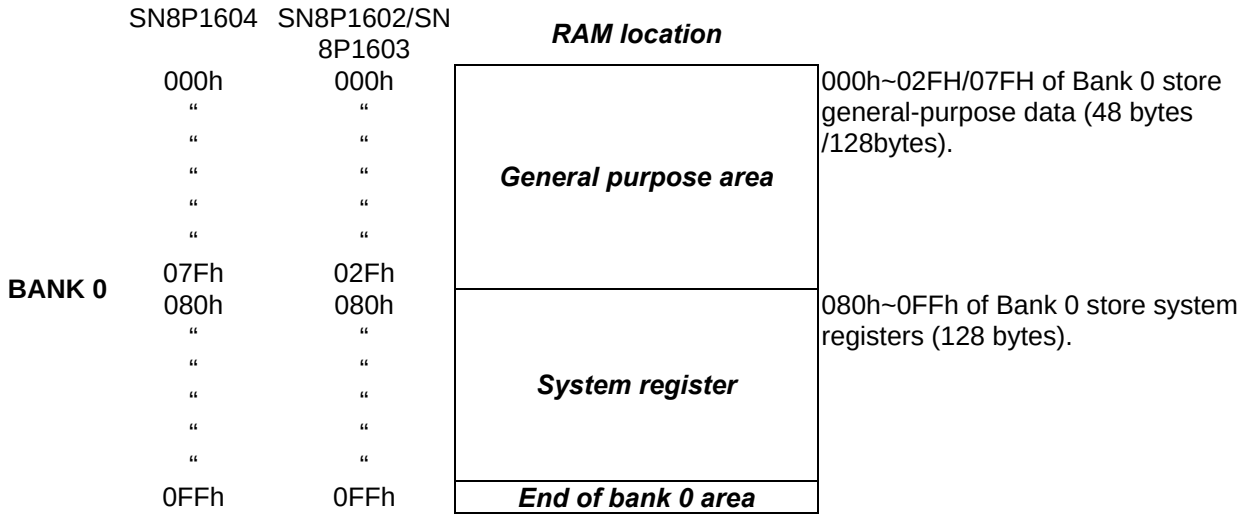
DATA MEMORY (RAM)

OVERVIEW

The SN8P1600 has internally built-in data memory up to 48/128 bytes for storing the general-purpose data.

- 48 * 8-bit general purpose area in bank 0 (SN8P1602/1603)
- 128 * 8-bit general purpose area in bank 0 (SN8P1604)
- 128 * 8-bit system register area

The memory is separated into bank 0. The bank 0 uses the first 48/128 bytes as general-purpose area, and the remaining 128 bytes as system register.



WORKING REGISTERS

The RAM bank0 locations 82H to 84H store the specially defined registers such as register R, Y, Z, respectively shown in the following table. These registers can use as the general-purpose working buffer or access ROM's and RAM's data. For instance, all of the ROM table can be looked-up by R, Y and Z registers. The data of RAM memory can be indirectly accessed with Y and Z registers.

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 address ROM location to look up ROM data.

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
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
After reset	0	0	0	0	0	0	0	0

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
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
After reset	0	0	0	0	0	0	0	0

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

➲ **Example: Following example uses indirectly addressing mode to access data in the RAM address 025H of bank0.**

```
        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.**

```
        B0MOV   Y, #0       ; Y = 0, bank 0
        B0MOV   Z, #07FH    ; 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

R register is an 8-bit buffer. There are two major functions of the register. First, R register can be used as working register. Second, the R register stores high-byte data of look-up ROM data. After MOVC instruction executed, the high-byte data of specified ROM address will store in R register and the low-byte data in ACC.

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
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
After reset	0	0	0	0	0	0	0	0

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

PROGRAM FLAG

The PFLAG includes carry flag (C), 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.

086H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
PFLAG	-	-	-	-	-	C	DC	Z
Read/Write	-	-	-	-	-	R/W	R/W	R/W
After reset	-	-	-	-	-	0	0	0

CARRY FLAG

C = 1: When executed arithmetic addition with overflow or executed arithmetic subtraction without borrow or executed rotation instruction with logic “1” shifting out.
C = 0: When executed arithmetic addition without overflow or executed arithmetic subtraction with borrow or executed rotation instruction with logic “0” shifting out.

DECIMAL CARRY FLAG

DC = 1: If executed arithmetic addition with overflow of low nibble or executed arithmetic subtraction without borrow of low nibble.
DC = 0: If executed arithmetic addition without overflow of low nibble or executed arithmetic subtraction with borrow of low nibble.

ZERO FLAG

Z = 1: When the content of ACC or target memory is zero after executing instructions involving a zero flag.
Z = 0: When the content of ACC or target memory is not zero after executing instructions involving a zero flag.

ACCUMULATOR

The ACC is an 8-bit 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 system doesn't store ACC and PFLAG value when interrupt executed. ACC and PFLAG data must be exchanged to other data memories 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: Protect ACC and working registers.**

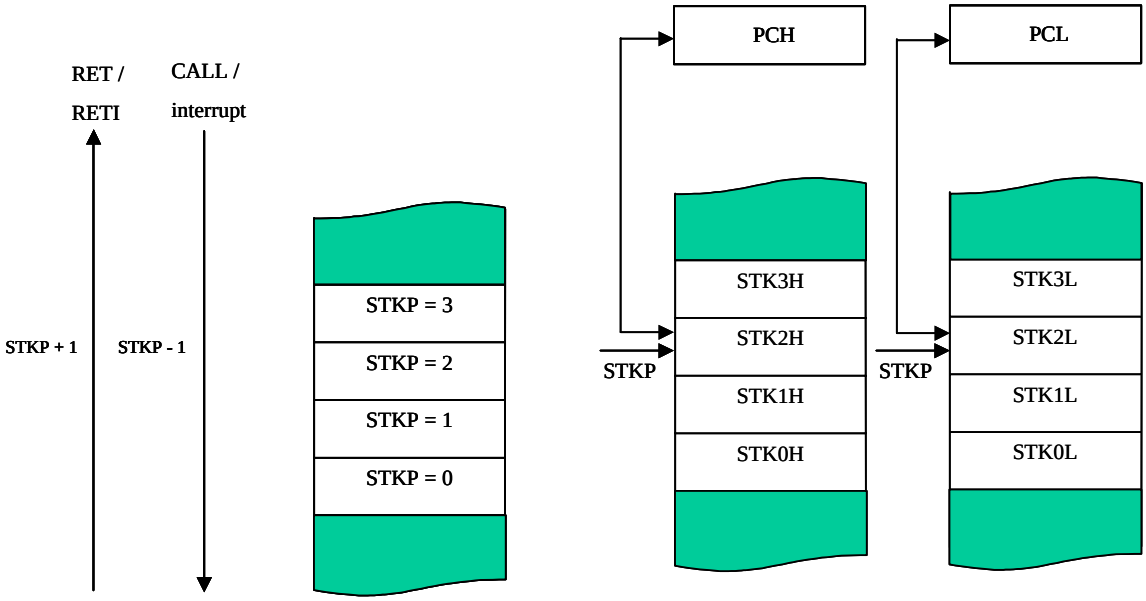
```
ACCBUF      EQU     00H         ; ACCBUF is ACC data buffer.
PFLAGBUF    EQU     01H         ; PFLAGBUF is PFLAG data buffer.

INT_SERVICE:
            B0XCH   A, ACCBUF   ; Store ACC value
            B0MOV   A, PFLAG    ; Store PFLAG value
            B0MOV   PFLAGBUF,A

            .       .
            .
            .

            B0MOV   A, PFLAGBUF ; Re-load PFLAG value
            B0MOV   PFLAG,A
            B0XCH   A, ACCBUF   ; Re-load ACC

            RETI                ; Exit interrupt service vector
```

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

STACK OPERATIONS

OVERVIEW

The stack buffer of SN8P1600 has 4-level high area and each level is 10-bit length. These buffers are designed to push and pop up program counter's (PC) data when interrupt service routine is executed. The STKP register is a pointer designed to point active level in order to push or pop up data from stack buffer of kernel circuit. The STKnH and STKnL are the 10-bit stack buffers to store program counter (PC) data.



STACK REGISTERS

The stack pointer (STKP) is a 3-bit register to store the address used to access the stack buffer, 10-bit 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 and reading from the top of stack. Push operation decrements the STKP and the pop operation increments each time. That makes the STKP always point to the top address of stack buffer and write 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.

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

STKPBn: Stack pointer. (n = 0 ~ 2)
GIE: Global interrupt control bit. 0 = disable, 1 = enable. There is more on this in interrupt chapter.

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

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

STKn = <STKnH , STKnL> (n = 3 ~ 0)

➢ ***SN8P1602/1603***

0F0H~0FFH	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
STKnH	-	-	-	-	-	-	SnPC9	SnPC8
Read/Write	-	-	-	-	-	-	R/W	R/W
After reset	-	-	-	-	-	-	0	0

➢ ***SN8P1604***

0F0H~0FFH	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
STKnH	-	-	-	-	SnPC11	SnPC10	SnPC9	SnPC8
Read/Write	-	-	-	-	R/W	R/W	R/W	R/W
After reset	-	-	-	-	0	0	0	0

➢ ***SN8P1602/1603/1604***

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
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
After reset	0	0	0	0	0	0	0	0

STKnH: Store PCH data as interrupt or call executing. The n expressed 0 ~3.
STKnL: Store PCL data as interrupt or call executing. The n expressed 0 ~3.

STACK OPERATION EXAMPLE

The two kinds of Stack-Save operations refer to the stack pointer (STKP) and write the content of program counter (PC) to the stack buffer are CALL instruction and interrupt service. Under each condition, the STKP decreases 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 the following table.

Stack Level	STKP Register			Stack Buffer		Description
	STKPB2	STKPB1	STKPB0	High Byte	Low Byte	
0	1	1	1	Free	Free	-
1	1	1	0	STK0H	STK0L	-
2	1	0	1	STK1H	STK1L	-
3	1	0	0	STK2H	STK2L	-
4	0	1	1	STK3H	STK3L	-
> 4	0	1	0	-	-	**Stack Over, error**

There are Stack-Restore operations correspond to each push operation to restore the program counter (PC). The RETI instruction uses for interrupt service routine. The RET instruction is for CALL instruction. When a pop 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 the following table.

Stack Level	STKP Register			Stack Buffer		Description
	STKPB2	STKPB1	STKPB0	High Byte	Low Byte	
4	0	1	1	STK3H	STK3L	-
3	1	0	0	STK2H	STK2L	-
2	1	0	1	STK1H	STK1L	-
1	1	1	0	STK0H	STK0L	-
0	1	1	1	Free	Free	-

PROGRAM COUNTER

The program counter (PC) is a 10-bit binary counter separated into the high-byte 2 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 9.

SN8P1602/SN8P1603

	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	-	-	-	-	-	-	PC9	PC8	PC7	PC6	PC5	PC4	PC3	PC2	PC1	PC0
After reset	-	-	-	-	-	-	0	0	0	0	0	0	0	0	0	0
	PCH								PCL							

SN8P1604

	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	-	-	-	-	PC11	PC10	PC9	PC8	PC7	PC6	PC5	PC4	PC3	PC2	PC1	PC0
After reset	-	-	-	-	0	0	0	0	0	0	0	0	0	0	0	0
	PCH								PCL							

ONE ADDRESS SKIPPING

There are nine instructions (CMPRS, INCS, INCMS, DECS, DECMS, BTS0, BTS1, 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          ; To skip, if Carry_flag = 1
            JMP       C0STEP      ; Else jump to C0STEP.
            .
C0STEP:     NOP

            B0MOV     A, BUF0     ; Move BUF0 value to ACC.
            B0BTS0    FZ          ; To skip, 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     ; To skip, if ACC = 12H.
            JMP       C0STEP      ; Else jump to C0STEP.
            .
C0STEP:     NOP
```

If the result after increasing or decreasing by 1 is 0xFF or 0x00, the PC will add 2 steps to skip next instruction.

INCS instruction:

```
            INCS      BUF0
            JMP       C0STEP      ; Jump to C0STEP if ACC is not zero.
            …
C0STEP:     NOP
```

INCMS instruction:

```
            INCMS     BUF0
            JMP       C0STEP      ; Jump to C0STEP if BUF0 is not zero.
            …
C0STEP:     NOP
```

DECS instruction:

```
            DECS      BUF0
            JMP       C0STEP      ; Jump to C0STEP if ACC is not zero.
            …
C0STEP:     NOP
```

DECMS instruction:

```
            DECMS     BUF0
            JMP       C0STEP      ; Jump to C0STEP if BUF0 is not zero.
            …
C0STEP:     NOP
```

MULTI-ADDRESS JUMPING

Users can jump round multi-address by either JMP instruction or ADD M, An instruction (M = PCL) to activate multi-address jumping function. If carry flag occurs after execution of ADD PCL, A, the carry flag 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 SN8P1600 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 table.

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.
```

5 SYSTEM REGISTER

OVERVIEW

The RAM area located in 80H~FFH bank 0 is system register area. The main purpose of system registers is to control peripheral hardware of the chip. Using system registers can control I/O ports, timers and counters by programming. The memory map provides an easy and quick reference source for writing application program. These system registers accessing is controlled by the selected memory bank (RBANK = 0) or the bank 0 read/write instruction (B0MOV, B0BSET, B0BCLR…).

SYSTEM REGISTER ARRANGEMENT (BANK 0)

BYTES of SYSTEM REGISTER

➢ ***SN8P1602/1603***

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
8	-	-	R	Z	Y	-	PFLAG	-	-	-	-	-	-	-	-	-
9	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
A	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
B	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
C	P1W	P1M	P2M	-	-	-	-	-	INTRQ	INTEN	OSCM	-	-	-	PCL	PCH
D	P0	P1	P2	-	-	-	-	-	-	-	TC0M	TC0C	-	-	-	STKP
E	-	-	-	-	-	-	-	@YZ	-	-	-	-	-	-	-	-
F	-	-	-	-	-	-	-	-	STK3L	STK3H	STK2L	STK2H	STK1L	STK1H	STK0L	STK0H

➢ ***SN8P1604***

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
8	-	-	R	Z	Y	-	PFLAG	-	-	-	-	-	-	-	-	-
9	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
A	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
B	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
C	P1W	P1M	P2M	-	-	P5M	-	-	INTRQ	INTEN	OSCM	-	-	-	PCL	PCH
D	P0	P1	P2	-	-	P5	-	-	-	-	-	-	TC1M	TC1C	TC1R	STKP
E	-	-	-	-	-	-	-	@YZ	-	-	-	-	-	-	-	-
F	-	-	-	-	-	-	-	-	STK3L	STK3H	STK2L	STK2H	STK1L	STK1H	STK0L	STK0H

Description

PFLAG = ROM page and special flag register.
P1W = Port 1 Wakeup register.
PnM = Port n input/output mode register.
INTRQ = Interrupt request register.
OSCM = Oscillator mode register.
TCnM = Timer n mode register.
STKP = Stack pointer buffer.
@YZ = RAM YZ indirect addressing index pointer.
R = Working register and ROM look-up data buffer.
Y, Z = Working, @YZ and ROM addressing register.
Pn = Port n data buffer.
INTEN = Interrupt enable register.
PCH, PCL = Program counter.
TCnC = Timer n counting register.
TC1R= TC1 8-bit reload register.
STK0~STK3 = Stack 0 ~ stack 3 buffer.

➢ ***Note:***

a). All register names had been declared in SN8ASM assembler.
b). 1-bit register name had been declared in SN8ASM assembler with "F" prefix code.
c). When using instruction to check empty location, logic "H" will be returned.
d). "b0bset", "b0bclr", "bset", "bclr" instructions only support "R/W" registers.

BITS of SYSTEM REGISTER

➢ ***SN8P1602/1603 system register table***

Address	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	R/W	Remarks
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
086H	-	-	-	-	-	C	DC	Z	R/W	PFLAG
0C0H	0	0	0	P14W	P13W	P12W	P11W	P10W	R/W	P1W wakeup register
0C1H	0	0	0	P14M	P13M	P12M	P11M	P10M	R/W	P1M I/O direction
0C2H	P27M	P26M	P25M	P24M	P23M	P22M	P21M	P20M	R/W	P2M I/O direction
0C8H	0	0	TC0IRQ	0	0	0	0	P00IRQ	R/W	INTRQ
0C9H	0	0	TC0IEN	0	0	0	0	P00IEN	R/W	INTEN
0CAH	0	WDRST	0	0	CPUM0	CLKMD	STPHX	0	R/W	OSCM
0CEH	PC7	PC6	PC5	PC4	PC3	PC2	PC1	PC0	R/W	PCL
0CFH	-	-	-	-	-	-	PC9	PC8	R/W	PCH
0D0H	-	-	-	-	-	-	-	P00	R	P0 data buffer
0D1H	-	-	-	P14	P13	P12	P11	P10	R/W	P1 data buffer
0D2H	P27	P26	P25	P24	P23	P22	P21	P20	R/W	P2 data buffer
0DAH	TC0ENB	TC0rate2	TC0rate1	TC0rate0	0	0	0	0	R/W	TC0M
0DBH	TC0C7	TC0C6	TC0C5	TC0C4	TC0C3	TC0C2	TC0C1	TC0C0	R/W	TC0C
0DFH	GIE	-	-	-	-	STKPB2	STKPB1	STKPB0	R/W	STKP stack pointer
0E7H	@YZ7	@YZ6	@YZ5	@YZ4	@YZ3	@YZ2	@YZ1	@YZ0	R/W	@YZ index pointer
0F8H	S3PC7	S3PC6	S3PC5	S3PC4	S3PC3	S3PC2	S3PC1	S3PC0	R/W	STK3L
0F9H	-	-	-	-	-	-	S3PC9	S3PC8	R/W	STK3H
0FAH	S2PC7	S2PC6	S2PC5	S2PC4	S2PC3	S2PC2	S2PC1	S2PC0	R/W	STK2L
0FBH	-	-	-	-	-	-	S2PC9	S2PC8	R/W	STK2H
0FCH	S1PC7	S1PC6	S1PC5	S1PC4	S1PC3	S1PC2	S1PC1	S1PC0	R/W	STK1L
0FDH	-	-	-	-	-	-	S1PC9	S1PC8	R/W	STK1H
0FEH	S0PC7	S0PC6	S0PC5	S0PC4	S0PC3	S0PC2	S0PC1	S0PC0	R/W	STK0L
0FFH	-	-	-	-	-	-	S0PC9	S0PC8	R/W	STK0H

➢ ***SN8P1604 system register table***

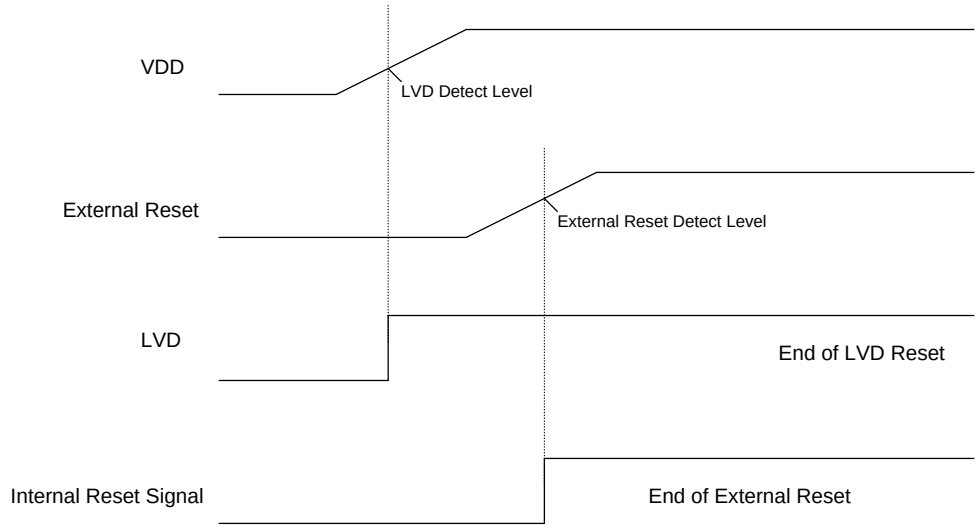
Address	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	R/W	Remarks
082H	RBIT7	RBIT6	RBIT5	RBIT4	RBIT3	RBIT2	RBIT1	RBIT0	R/W	R
083H	ZBIT7	ZBIT6	ZBIT5	ZBIT4	ZBIT3	ZBIT2	ZBIT1	ZBIT0	R/W	Z
084H	YBIT7	YBIT6	YBIT5	YBIT4	YBIT3	YBIT2	YBIT1	YBIT0	R/W	Y
086H	-	-	-	-	-	C	DC	Z	R/W	PFLAG
0C0H	P17W	P16W	P15W	P14W	P13W	P12W	P11W	P10W	R/W	P1W wakeup register
0C1H	P17M	P16M	P15M	P14M	P13M	P12M	P11M	P10M	R/W	P1M I/O direction
0C2H	P27M	P26M	P25M	P24M	P23M	P22M	P21M	P20M	R/W	P2M I/O direction
0C5H	0	0	0	0	P53M	P52M	P51M	P50M	R/W	P5M I/O direction
0C8H	0	TC1IRQ	0	0	0	0	0	P00IRQ	R/W	INTRQ
0C9H	0	TC1IEN	0	0	0	0	0	P00IEN	R/W	INTEN
0CAH	0	WDRST	0	0	CPUM0	CLKMD	STPHX	0	R/W	OSCM
0CEH	PC7	PC6	PC5	PC4	PC3	PC2	PC1	PC0	R/W	PCL
0CFH	-	-	-	-	PC11	PC10	PC9	PC8	R/W	PCH
0D0H	-	-	-	-	-	-	P01	P00	R	P0 data buffer
0D1H	P17	P16	P15	P14	P13	P12	P11	P10	R/W	P1 data buffer
0D2H	P27	P26	P25	P24	P23	P22	P21	P20	R/W	P2 data buffer
0D5H	-	-	-	-	P53	P52	P51	P50	R/W	P5 data buffer
0DCH	TC1ENB	TC1rate2	TC1rate1	TC1rate0	0	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
0E7H	@YZ7	@YZ6	@YZ5	@YZ4	@YZ3	@YZ2	@YZ1	@YZ0	R/W	@YZ index pointer
0F8H	S3PC7	S3PC6	S3PC5	S3PC4	S3PC3	S3PC2	S3PC1	S3PC0	R/W	STK3L
0F9H	-	-	-	-	S3PC11	S3PC10	S3PC9	S3PC8	R/W	STK3H
0FAH	S2PC7	S2PC6	S2PC5	S2PC4	S2PC3	S2PC2	S2PC1	S2PC0	R/W	STK2L
0FBH	-	-	-	-	S2PC11	S2PC10	S2PC9	S2PC8	R/W	STK2H
0FCH	S1PC7	S1PC6	S1PC5	S1PC4	S1PC3	S1PC2	S1PC1	S1PC0	R/W	STK1L
0FDH	-	-	-	-	S1PC11	S1PC10	S1PC9	S1PC8	R/W	STK1H
0FEH	S0PC7	S0PC6	S0PC5	S0PC4	S0PC3	S0PC2	S0PC1	S0PC0	R/W	STK0L
0FFH	-	-	-	-	S0PC11	S0PC10	S0PC9	S0PC8	R/W	STK0H

➢ ***Note: 1)To avoid system error, please be sure to put all the "0" as it indicates in the above table***
2). For detail description please refer file of "System Register Quick Reference Table"

6 POWER ON RESET

OVERVIEW

SN8P1600 provides two system resets. One is external reset and the other is low voltage detector (LVD). The external reset is a simple RC circuit connecting to the reset pin. The low voltage detector (LVD) is built-in internal circuit. When one of the reset devices occurs, the system will reset and the system registers become initial value. The timing diagram is as the following.

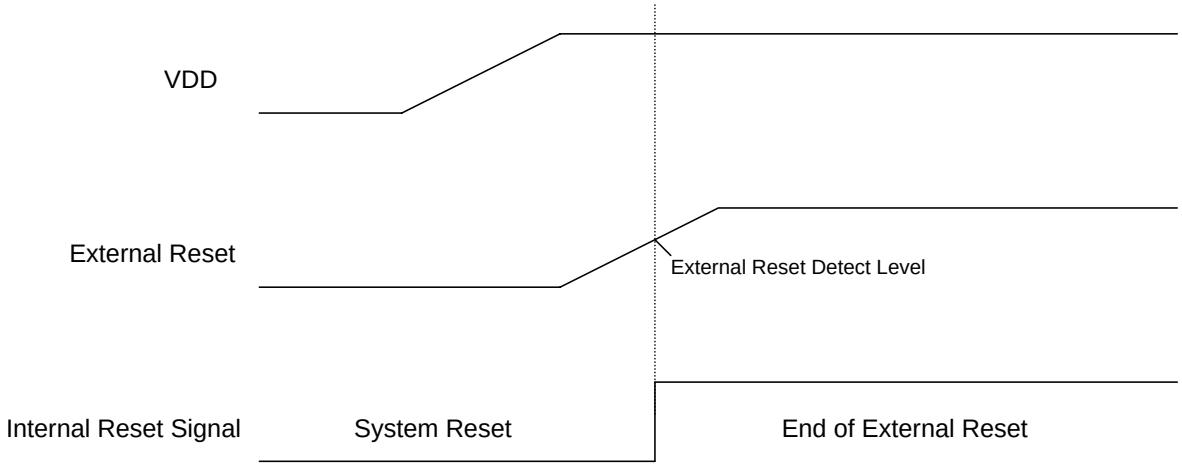


SN8P1600 power on reset timing diagram

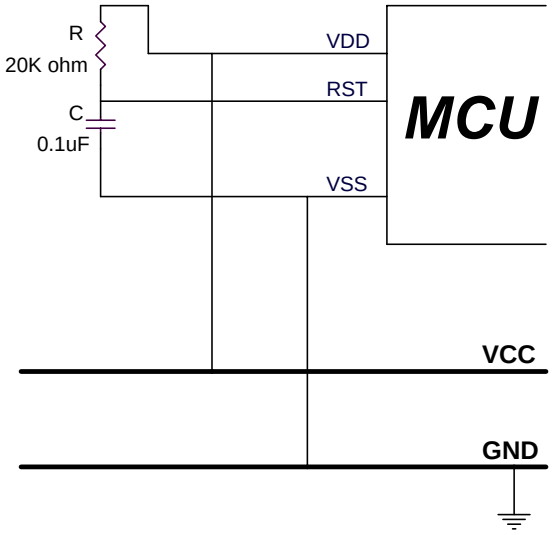
Notice : The working current of the LVD is about 100uA.

EXTERNAL RESET DESCRIPTION

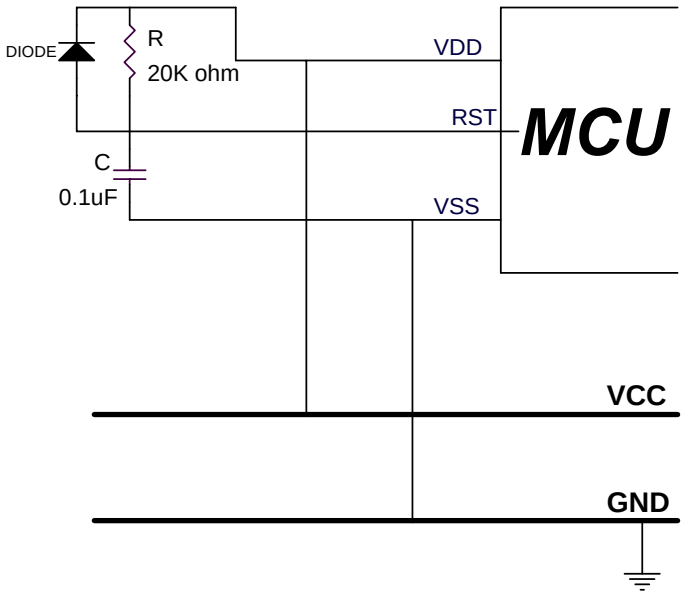
The external reset is a low level active device. The reset pin receives the low voltage and resets the system. When the voltage detects high level, it stops resetting the system. Users can use an external reset circuit to control system operation. It is necessary that the VDD must be stable.



The external reset will fail, if the external reset voltage stabilizes before VDD voltage. Users must make sure the VDD is stable earlier than external reset. The external reset circuit is a simple RC circuit as the following figure.

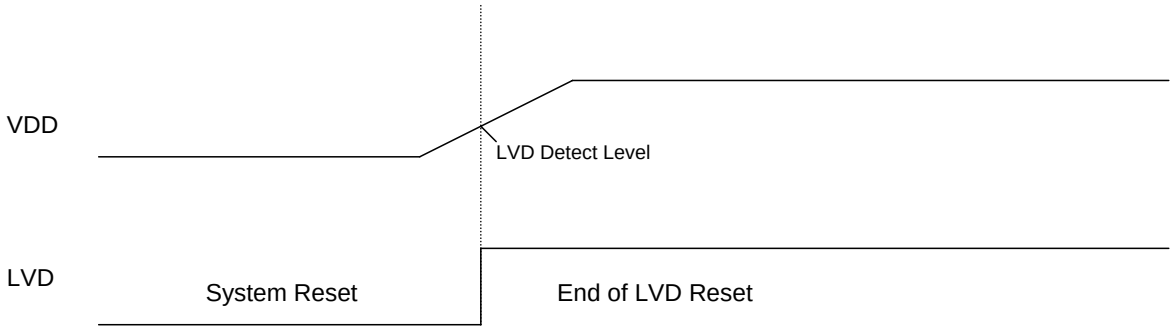


In power-fail condition as Brown-out reset. The reset pin may keep high level but the VDD is low voltage. That makes the system reset fail and chip error. To connect a diode from reset pin to VDD is a good solution. This circuit can force the capacitor to release electrical charge and drop the voltage, and solve the error.



LOW VOLTAGE DETECTOR (LVD) DESCRIPTION

The LVD is a low voltage detector. It detects VDD level and reset the system as the VDD lower than the desired voltage. The detect level is 2.4V. If the VDD lower than 2.4V, the system resets. The LVD function is controlled by code option. Users can turn on it for special application like power-fail condition. LVD work with external reset function. They are OR active.



The LVD can protect system to work well under Brown-out reset, but it is a high consumptive circuit. In 3V condition, the LVD consumes about 100uA. It is a very large consumption for battery system, but supports AC system well.

➢ ***Note: LVD is enabled by code option.***

7 OSCILLATORS

OVERVIEW

The SN8P1600 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.

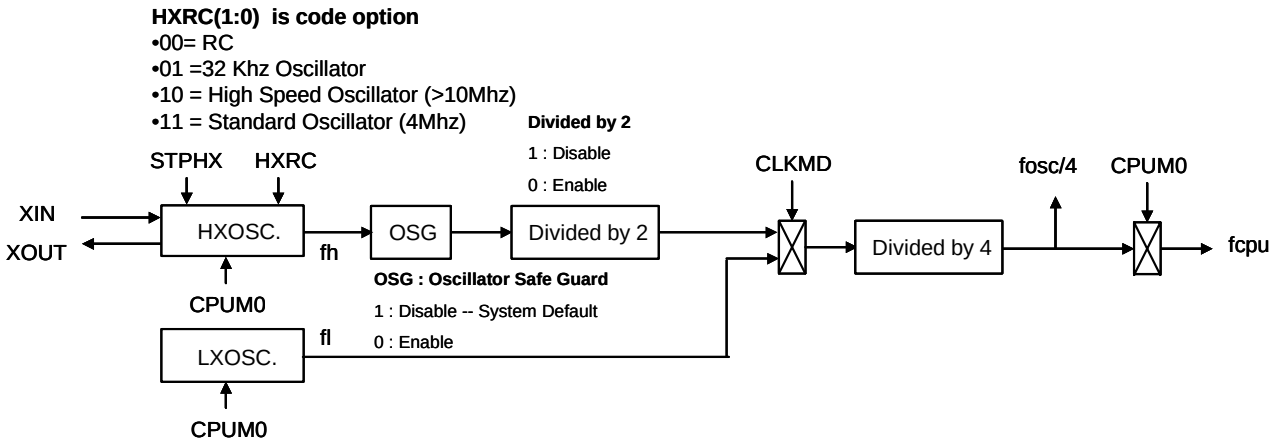
The external high-speed clock and the internal low-speed clock can be system clock (Fosc). The system clock is divided by 4 to be the instruction cycle (Fcpu).

Fcpu = Fosc / 4

The system clock is required by the following peripheral modules:

- ✓ ***Timer 0 (TC0)***
- ✓ ***Watchdog timer***

CLOCK BLOCK DIAGRAM



- ➢ HXOSC: External high-speed clock.
- ➢ LXOSC: Internal low-speed clock.
- ➢ OSG: Oscillator safe guard.

OSCM REGISTER DESCRIPTION

The OSCM register is an oscillator control register. It controls oscillator selection, system mode, watchdog timer clock rate.

0CAH	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
OSCM	0	WDRST	0	0	CPUM0	CLKMD	STPHX	0
Read/Write	-	R/W	-	-	R/W	R/W	R/W	-
After reset	-	0	-	-	0	0	0	-

STPHX: Eternal high-speed oscillator control bit. 0 = free run, 1 = stop. This bit only controls external high-speed oscillator. If STPHX=1, the internal low-speed RC oscillator is still running.
CLKMD: System high/Low speed mode select bit. 0 = normal (dual) mode, 1 = slow mode.
CPUM0: CPU operating mode control bit. 0 = normal, 1 = sleep (power down) mode to turn off both high/low clock.

WDRST is watchdog timer control bits. The detail information is in watchdog timer chapter.

- ***Note: The bit 0, 4, 7 of OSCM register must be "0", or the system will be error.***

EXTERNAL HIGH-SPEED OSCILLATOR

SN8P1600 can be operated in four different oscillator modes. 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 satiable oscillator mode by programming 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).
```

OSCILLATOR MODE CODE OPTION

SN8P1600 has 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. MCU needs more current when operating at High-speed mode than the low-speed mode. For crystals, there are three steps to select. If the oscillator is RC type, to select “RC” and the system will divide the frequency by 2 automatically. User can select oscillator mode from code option table before compiling. Following is the code option table.

Code Option	*Oscillator Mode*	*Remark*
00	RC mode	Output the Fcpu square wave from Xout pin.
01	32K	32768Hz
10	12M	12MHz ~ 16MHz
11	4M	3.58MHz

OSCILLATOR DEVIDE BY 2 CODE OPTION

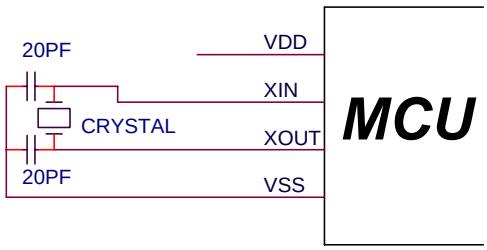
SN8P1600 has a code option to divide external clock by 2,called “High_Clk / 2”. If “High_Clk / 2” is enabled, the external clock frequency is divided by 8 for the Fcpu. Fcpu is equal to Fosc/8. If “High_Clk / 2” is disabled, the external clock frequency is divided by 4 for the Fcpu. The Fcpu is equal to Fosc/4.

➢ ***Note: In RC mode, “High_Clk / 2” is always enabled.***

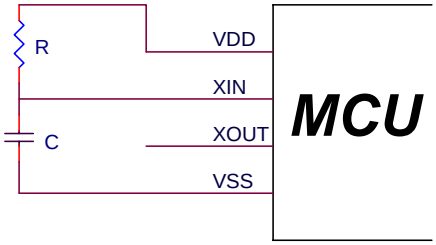
OSCILLATOR SAFE GUARD CODE OPTION

SN8P1600 builds in an oscillator safe guard (OSG) to make oscillator more stable. It is a low-pass filter circuit and stops high frequency noise into system from external oscillator circuit. This function makes system to work better under AC noisy conditions.

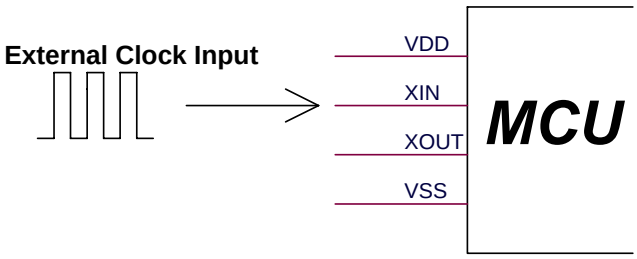
SYSTEM OSCILLATOR CIRCUITS



Crystal/Ceramic Oscillator



RC Oscillator



External clock input

- ***Note1: The VDD and VSS of external oscillator circuit must be from micro-controller. Don't connect them from 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: In RC type oscillator code option situation, the external clock frequency is divided by 2.***

External RC Oscillator Frequency Measurement

There are two ways to get the Fosc frequency of external RC oscillator. One measures the XOUT output waveform. Under external RC oscillator mode, the XOUT outputs the square waveform whose frequency is Fcpu. The other measures the external RC frequency by instruction cycle (Fcpu). The external RC frequency is the Fcpu multiplied by 4. 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    P1M.0     ; Set P1.0 to be output mode for outputting Fcpu toggle
                              signal.

@@:
        B0BSET    P1.0      ; Output Fcpu toggle signal in low-speed clock mode.
        B0BCLR    P1.0      ; Measure the Fcpu frequency by oscilloscope.
        JMP       @B
```

INTERNAL LOW-SPEED OSCILLATOR

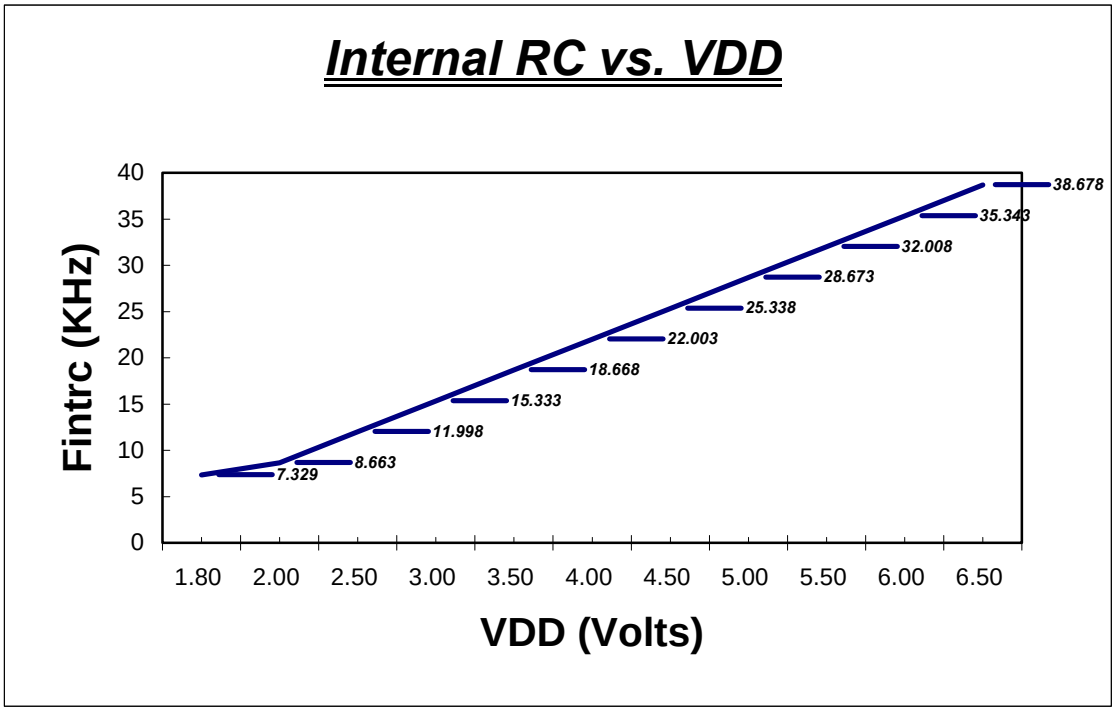
The internal low-speed oscillator is built in the micro-controller. The low-speed clock source is a RC type oscillator circuit. The low-speed clock can supplies clock for system clock and timer,.

➲ **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 the following figure.



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

```
        B0BSET    P1M.0       ; Set P1.0 to be output mode for outputting Fcpu toggle signal.

        B0BSET    FCLKMD      ; Switch the system clock to internal low-speed clock mode.
@@:
        B0BSET    P1.0        ; Output Fcpu toggle signal in low-speed clock mode.
        B0BCLR    P1.0        ; 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.

- High-speed mode
- Low-speed mode
- Power-down mode (Sleep mode)

In actual application, user can adjust the MCU to work in one of these three modes by using OSCM register. At the high-speed mode, the instruction cycle (Fcpu) is Fosc/4. At 3V, low-speed mode, the Fcpu is 16KHz/4.

NORMAL MODE

In normal mode, the system clock source is external high-speed clock. After power on, the system works under normal mode. The instruction cycle is fosc/4. When the external high-speed oscillator is 3.58MHz, the instruction cycle is 3.58MHz/4 = 895KHz. All software and hardware are executed and working. In normal mode, system can get into power down mode and slow mode.

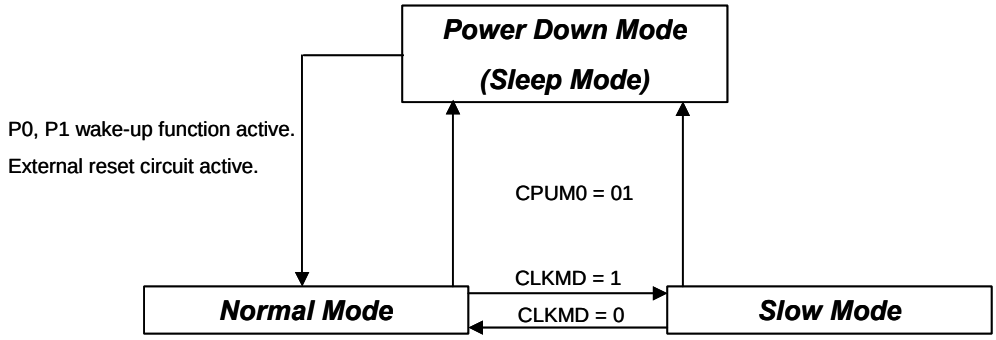
SLOW MODE

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 MODE

The power down mode is also called sleep mode. The MCU stops working as sleeping status. The power consumption is very less as zero. The power down mode is usually applied to power-saving system like battery-powered 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, P1 trigger signal.

SYSTEM MODE CONTROL



SN8P1600 OTP Type.

Operating mode description

MODE	NORMAL	SLOW	POWER DOWN (SLEEP)	REMARK
HX osc.	Running	By STPHX	Stop	
LX osc.	Running	Running	Stop	
CPU instruction	Executing	Executing	Stop	
TC0 timer	*Active	*Active	Inactive	* Active by program
Watchdog timer	Active	Active	Inactive	
Internal interrupt	All active	All active	All inactive	
External interrupt	All active	All active	All inactive	
Wakeup source	-	-	P0, P1, Reset	

SYSTEM MODE SWITCHING

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

```
        B0BSET      FCPUM0          ; set the system into power down mode.
```

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.
```

Switch slow mode to normal mode
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 work stably. The external high-speed oscillator sometimes starts and stops at different situations. The delay time for external high-speed oscillator restart is called Wakeup time.

Following are two conditions need Wakeup time. One is switching power down mode to normal mode. The other is switching slow mode to normal mode. For the first case, SN8P1600 provides 2048 oscillator clocks as the Wakeup time. The second case, users need to calculate the Wakeup time.

HARDWARE WAKEUP

When the system is in power down mode (sleep mode), the external high-speed oscillator stops. When waked up from power down mode, MCU waits for 2048 external high-speed oscillator clocks as the Wakeup time to stable the oscillator circuit. After the Wakeup time, the system goes into the normal mode. The value of the Wakeup time is as the following.

The Wakeup time = 1/Fosc * 2048 (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 or P1 trigger signal. After the Wakeup time, the system goes into normal mode. The Wakeup time of P0, P1 Wakeup function is as the following.**

*The Wakeup time = 1/Fosc * 2048 = 0.57 ms (Fosc = 3.58MHz)*
The total wakeup time = 0.57ms + x'tal settling time

Under power down mode (sleep mode), there are only I/O ports with Wakeup function wake the system up to normal mode. The Port 0 and Port 1 have Wakeup function. Port 0 Wakeup function always enables, but the Port 1 is controlled by the P1W register.

➢ ***SN8P1602/1603***

0C0H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
P1W	0	0	0	P14W	P13W	P12W	P11W	P10W
Read/Write	-	-	-	R/W	R/W	R/W	R/W	R/W
After reset	-	-	-	0	0	0	0	0

➢ ***SN8P1604***

0C0H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
P1W	P17W	P16W	P15W	P14W	P13W	P12W	P11W	P10W
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
After reset	0	0	0	0	0	0	0	0

P10W~P14W: Port 1 Wakeup function control bits. 0 = none Wakeup function, 1 = Enable Wakeup function.

8 TIMERS

WATCHDOG TIMER (WDT)

The watchdog timer (WDT) is a binary up counter designed for monitoring program execution. If the program goes into the unknown status by noise interference, WDT overflow signal raises and resets MCU. The instruction that clear the watchdog timer ("B0BSET FWDRST") should be executed within a certain period. If an instruction that clears the watchdog timer is not executed within the period and the watchdog timer overflows, reset signal is generated and system is restarted. The watchdog timer rate has two rates for high/low speed mode. WDT rate selection is handled by oscillator code option. The watchdog timer disables at power down mode.

0CAH	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
OSCM	0	WDRST	0	0	CPUM0	CLKMD	STPHX	0
Read/Write	-	R/W	-	-	R/W	R/W	R/W	-
After reset	-	0	-	-	0	0	0	-

WDRST: Watchdog timer reset bit. 0 = Non reset, 1 = clear the watchdog timer counter.

➢ ***Note: The bit 0, 4, 5, 7 must be "0", or the system will be error.***

Watchdog timer overflow table.

Code option of High_Clk	Watchdog timer overflow time
4M_X'tal / 12M_X'tal / RC	1 / (fcpu ÷ 2^{14} ÷ 16) = 293 ms, Fosc=3.58MHz
32K_X'tal	1 / (fcpu ÷ 2^{8} ÷ 16) = 500 ms, Fosc=32768Hz

➢ ***Note: The watchdog timer can be enabled or disabled by the code option.***

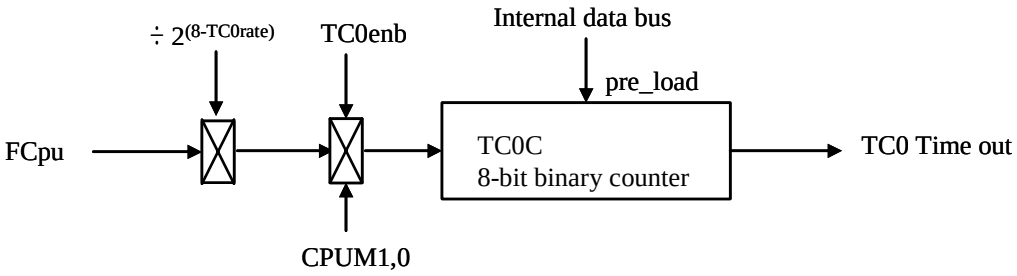
➲ **Example: An operation of watchdog timer is as following. To clear the watchdog timer counter in the top of the main routine of the program.**

```
Main:
        B0BSET      FWDRST      ; Clear the watchdog timer counter.
        .           .
        CALL        SUB1
        CALL        SUB2
        .           .
        .           .
        .           .
        JMP         MAIN
```

TIMER0 (TC0) (SN8P1602/1603 Only)

OVERVIEW

The timer counter 0 (TC0) is used to generate an interrupt request when a specified time interval has elapsed. If the TC0 timer has occur an overflow (from FFH to 00H), it will continue counting and issue a time-out signal to trigger TC0 interrupt to request interrupt service.



The main purposes of the TC0 timer is as following.

- **8-bit programmable timer:** Generates interrupts at specific time intervals based on the selected clock frequency.

TC0M MODE REGISTER

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

Eight rates for TC0 timer can be selected by TC0RATE0 ~ TC0RATE2 bits. 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 clock frequency and the intervals.

0DAH	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
TC0M	TC0ENB	TC0rate2	TC0rate1	TC0rate0	0	0	0	0
Read/Write	R/W	R/W	R/W	R/W	-	-	-	-
After reset	0	0	0	0	0	-	-	-

TC0ENB: TC0 counter enable bit. “0” = disable, “1” = enable.
TC0RATE2~TC0RATE0: TC0 internal clock select bits. 000 = fcpu/256, 001 = fcpu/128, … , 110 = fcpu/4, 111 = fcpu/2.

TC0C COUNTING REGISTER

TC0C is an 8-bit counter register for the timer (TC0). TC0C must be reset whenever the TC0ENB is set to "1" to start the timer. TC0C is incremented each time a clock pulse of the frequency determined by TC0RATE0 ~ TC0RATE2. When TC0C has incremented to "0FFH", it counts to "00H" an overflow generated. Under TC0 interrupt service request (TC0IEN) enable condition, the TC0 interrupt request flag will be set to "1" and the system executes the interrupt service routine. The TC0C has no auto reload function. After TC0C overflow, the TC0C is continuing counting. Users need to reset TC0C value to get an accurate time.

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
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
After reset	0	0	0	0	0	0	0	0

The basic timer table interval time of TC0.

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

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/64)**

$$\begin{aligned} \textit{TC0C initial value} &= 256 - (\textit{TC0 interrupt interval time} * \textit{input clock}) \\ &= 256 - (10\text{ms} * 3.58 * 10^6 / 4 / 64) \\ &= 256 - (10^{-2} * 3.58 * 10^6 / 4 / 64) \\ &= 116 \\ &= 74\text{H} \end{aligned}$$

TC0 TIMER OPERATION SEQUENCE

The TC0 timer's sequence of operation may be as following.

- Set the TC0C initial value to setup the interval time.
- Set the TC0ENB to be "1" to enable TC0 timer.
- TC0C is incremented by one after each clock pulse corresponding to TC0M selection.
- TC0C overflow if 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.**

```
        B0BCLR      FTC0IEN         ; To disable TC0 interrupt service
        B0BCLR      FTC0ENB         ; To disable TC0 timer
        MOV         A,#20H          ;
        B0MOV       TC0M,A          ; To set TC0 clock = fcpu / 64
        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
```

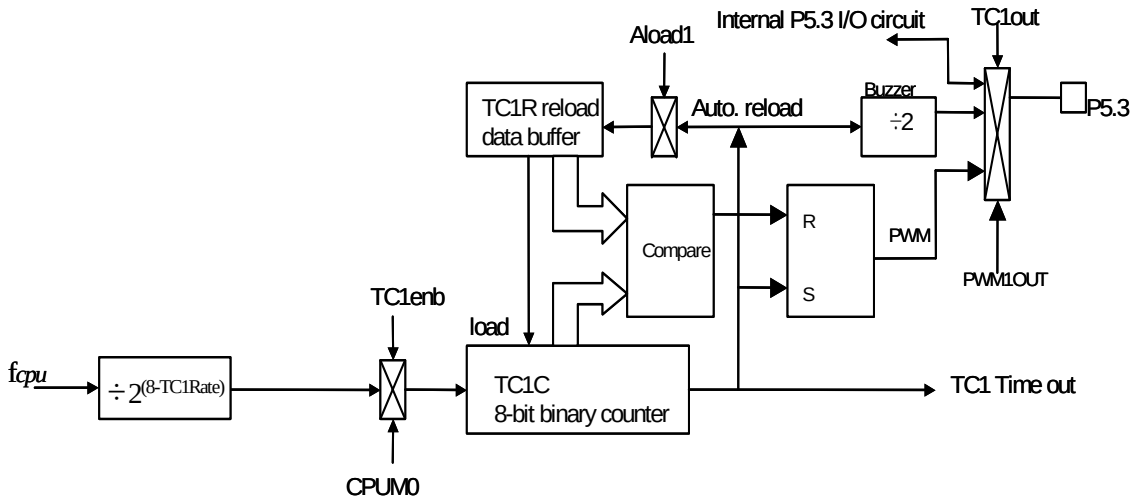
➲ **Example: TC0 interrupt service routine.**

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

        B0XCH       A, ACCBUF       ; B0xch instruction do not change C,Z flag
        B0MOV       A, PFLAG
        B0MOV       PFLAGBUF, A

        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
        B0XCH       A, ACCBUF       ; Restore ACC value.

        RETI                        ; Exit interrupt vector
```

TIMER1 (TC1) (SN8P1604 Only)

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.



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 PWM1 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 frequency dynamically as program executing.

Eight rates for TC1 timer can be selected by TC0RATE1 ~ TC1RATE2 bits. The range is from fcpu/2 to fcpu/256. The TC1M initial value is zero and the rate is fcpu/256. The bit7 of TC1M named 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.

0DCH	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
TC1M	TC1ENB	TC1rate2	TC1rate1	TC1rate0	0	ALOAD1	TC1OUT	PWM1OUT
Read/Write	R/W	R/W	R/W	R/W	-	R/W	R/W	R/W
After reset	0	0	0	0	0	0	0	0

TC1ENB: TC1 counter/BZ1/PWM1OUT enable bit. “0” = disable, “1” = enable.
TC1RATE2~TC1RATE0: TC1 internal clock select bits. 000 = fcpu/256, 001 = fcpu/128, … , 110 = fcpu/4, 111 = fcpu/2.
ALOAD1: TC1 auto-reload control bit, “0” the auto-reload function is disabled. “1 “ is to enable the auto-reload function.
TC1OUT : TC1 Time-out toggle signal output control bit. “0”: No TC1 time-out output signal. “1”: When TC1 time-out occurs, P5.3 output toggles.
PWM1OUT : PWM output control bit. “0”: No PWM output function. “1”: PWM will output waveform through pin P5.3.

TC1C COUNTING REGISTER

TC1C is an 8-bit counter register for the timer (TC1). TC1C must be reset whenever the TC1ENB is set to "1" to start the timer. TC1C is incremented each time a clock pulse of the frequency determined by TC1RATE0 ~ TC1RATE2. When TC1C has incremented to "0FFH", it counts to "00H" an overflow generated. Under TC1 interrupt service request (TC1IEN) enable condition, the TC1 interrupt request flag will be set to "1" and the system executes the interrupt service routine. When TC1C overflows, the TC1C will be restored automatically if ALOAD1 of TC1M register is enabled.

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
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
After reset	-	-	-	-	-	-	-	-

The basic timer table interval time of TC1.

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

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/64)**

$$\begin{aligned} \textit{TC1C initial value} &= 256 - (\textit{TC1 interrupt interval time} * \textit{input clock}) \\ &= 256 - (10ms * 3.58 * 10^6 / 4 / 64) \\ &= 256 - (10^{-2} * 3.58 * 10^6 / 4 / 64) \\ &= 116 \\ &= 74H \end{aligned}$$

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.

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
Read/Write	W	W	W	W	W	W	W	W
After reset	-	-	-	-	-	-	-	-

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
MOV         A,#20H      ;
B0MOV       TC1M,A      ; To set TC1 clock = fcpu / 64
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
MOV         A,#20H      ;
B0MOV       TC1M,A      ; To set TC1 clock = fcpu / 64
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       ; B0XCH doesn't change C, Z flag

                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       ; B0XCH doesn't change C, Z flag

                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.

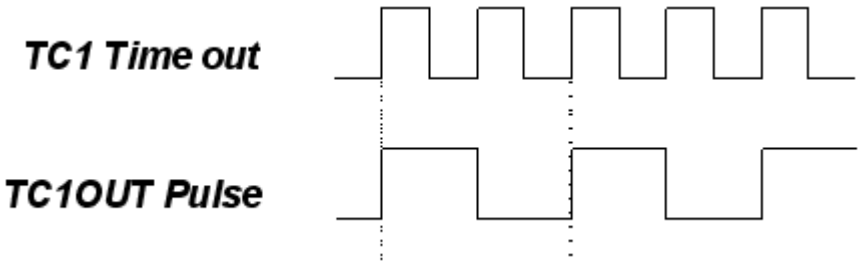


Figure 8-1. The TC1OUT Pulse Frequency

➲ **Example: Setup TC1OUT output from TC1 to TC1OUT (P5.3). The external high-speed clock 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. TC1C = TC1R = 131.**

```
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
```

PWM FUNCTION DESCRIPTION (SN8P1604 Only)

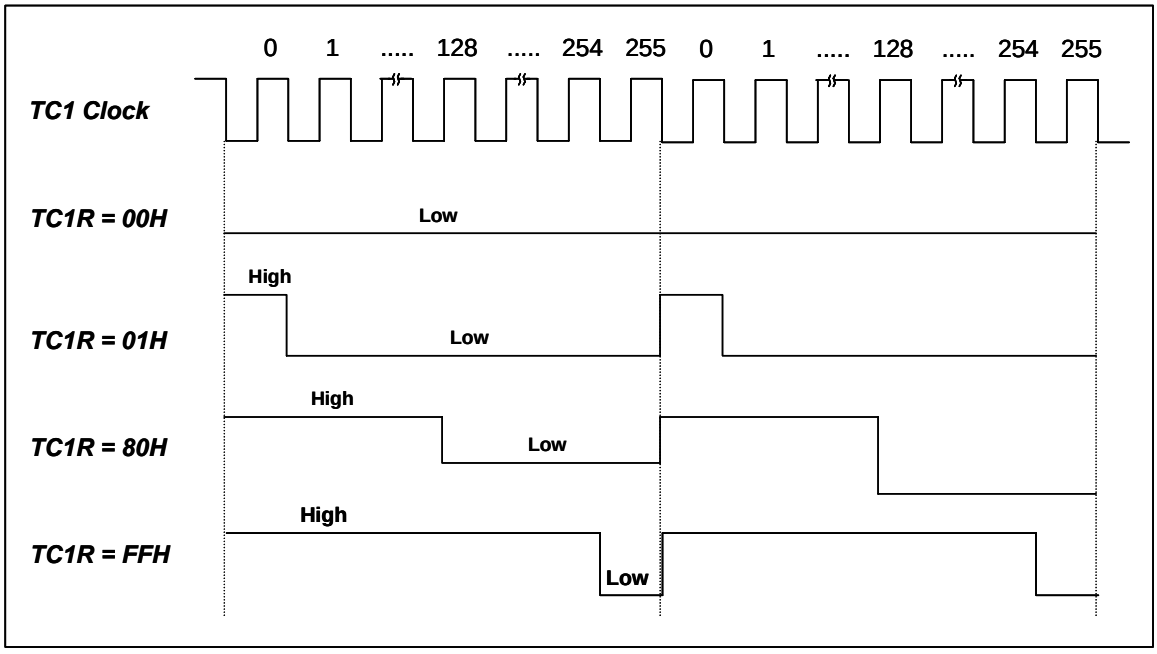
OVERVIEW

PWM function is generated by TC1 timer counter and output the PWM signal to 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 TC1R. When the reference register value TC1R is equal to the counter value TC1C, the PWM output goes low. When the counter reaches zero, the PWM output is forced high. The low-to-high ratio (duty) of the PWM1 output is TC1R/256.

All PWM outputs remain inactive during the first 256 input clock signals. Then, when the counter value (TC1C) changes from FFH back to 00H, the PWM output is forced to high level. The pulse width ratio (duty cycle) is defined by the contents of the reference register (TC1R) and is programmed in increments of 1:256. The 8-bit PWM data register TC1R is a write only register.

PWM output can be held at low level by continuously loading the reference register with 00H. Under PWM operating, to change the PWM's duty cycle is to modify the TC1R.

Reference Register Value (TC1R)	*Duty*
0000 0000	0/256
0000 0001	1/256
0000 0010	2/256
.	.
.	.
1000 0000	128/256
1000 0001	129/256
.	.
.	.
1111 1110	254/256
1111 1111	255/256



PWM PROGRAM DESCRIPTION

➲ **Example: Setup PWM1 output from TC1 to PWM1OUT (P5.3). The external high-speed oscillator clock is 4MHz. The duty of PWM is 30/256. The PWM frequency is about 1KHz. The PWM clock source is from external oscillator clock. TC1 rate is Fcpu/4. The TC1RATE2~TC1RATE1 = 110. TC1C = TC1R = 30.**

```
MOV     A,#01100000B
B0MOV   TC1M,A          ; Set the TC1 rate to Fcpu/4
MOV     A,#0x00         ;First Time Initial TC0
B0MOV   TC1C,A

MOV     A,#30           ; Set the PWM duty to 30/256
B0MOV   TC1R,A

B0BCLR  FTC1OUT         ; Disable TC1OUT function.
B0BSET  FPWM1OUT        ; Enable PWM1 output to P5.3 and disable P5.3 I/O function
B0BSET  FTC1ENB         ; Enable TC1 timer
```

➢ ***Note1: The TC1R is write-only register. Don't process them using INCMS, DECMS instructions.***
➢ ***Note2: Set TC1C at initial is to make first duty-cycle correct. After TC1 is enabled, don't modify TC1R value to avoid duty cycle error of PWM output.***

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

```
MOV     A, #30H         ; Input a number using B0MOV instruction.
B0MOV   TC1R, A

INCMS   BUF1            ; Get the new TC1R value from the BUF1 buffer defined by
B0MOV   A, BUF1         ; programming.
B0MOV   TC1R, A
```

➢ ***Note3: 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 TC1OUT function must be set "0" when PWM1 output enable.***

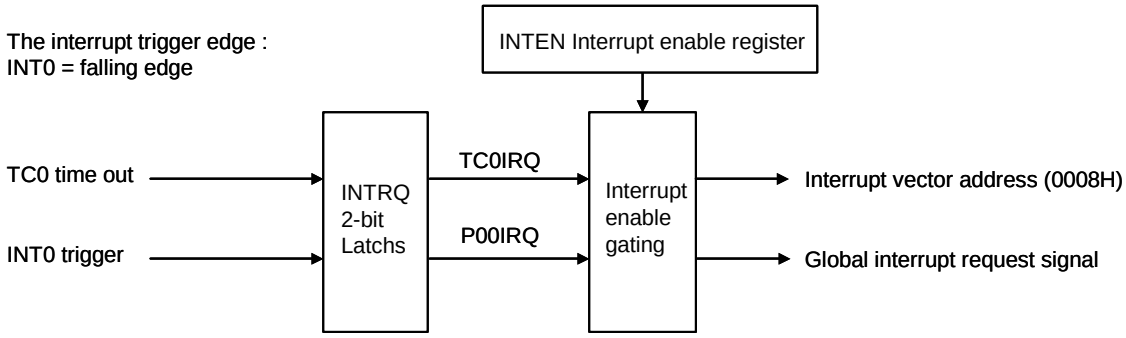
➢ ***Note5: The PWM can work with interrupt request.***

9 INTERRUPT

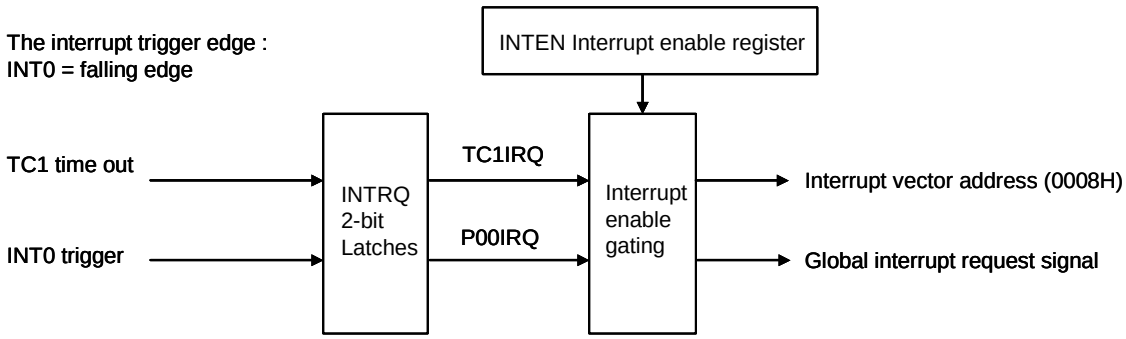
OVERVIEW

The SN8P1600 provides 2 interrupt sources, including one internal interrupts (TC0/TC1) and one external interrupts (INT0). The external interrupt can Wakeup the chip while the system is switched from power down mode to high-speed normal mode. Once interrupt service is executed, the GIE bit in STKP register will clear to “0” for stopping other interrupt request. On the contrast, 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.

➢ ***SN8P1602/1603***



➢ ***SN8P1604***



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

INTEN INTERRUPT ENABLE REGISTER

INTEN is the interrupt request control register including one internal interrupts, one external interrupts enable control bits. One of the register to be set "1" is to enable the interrupt request function. Once of the interrupt occur, the stack is incremented and 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.

➢ ***SN8P1602/1603***

0C9H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
INTEN	0	0	TC0IEN	0	0	0	0	P00IEN
Read/Write	-	-	R/W	-	-	-	-	R/W
After reset	-	-	0	-	-	-	-	0

➢ ***SN8P1604***

0C9H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
INTEN	0	TC1IEN	0	0	0	0	0	P00IEN
Read/Write	-	R/W	-	-	-	-	-	R/W
After reset	-	0	-	-	-	-	-	0

P00IEN : External P0.0 interrupt control bit. 0 = disable, 1 = enable.
TC0IEN : Timer 0 interrupt control bit. 0 = disable, 1 = enable.

INTRQ INTERRUPT REQUEST REGISTER

INTRQ is the interrupt request flag register. The register includes all interrupt request indication flags. Each one of the interrupt requests 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.

➢ ***SN8P1602/1603***

0C8H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
INTRQ	0	0	TC0IRQ	0	0	0	0	P00IRQ
Read/Write	-	-	R/W	-	-	-	-	R/W
After reset	-	-	0	-	-	-	-	0

➢ ***SN8P1604***

0C8H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
INTRQ	0	TC1IRQ	0	0	0	0	0	P00IRQ
Read/Write	-	R/W	-	-	-	-	-	R/W
After reset	-	0	-	-	-	-	-	0

P00IRQ : External P0.0 interrupt request bit. 0 = non-request, 1 = request.
TC0IRQ : TC0 timer interrupt request controls bit. 0 = non request, 1 = request.

When interrupt occurs, the related request bit of INTRQ register will be set to "1" no matter the related enable bit of INTEN register is enabled or disabled. If the related bit of INTEN = 1 and the related bit of INTRQ is also set to be "1". As the result, the system will execute the interrupt vector (ORG 8). If the related bit of INTEN = 0, moreover, the system won't execute interrupt vector even when the related bit of INTRQ is set to be "1". Users need to be cautious with the operation under multi-interrupt situation.

INTERRUPT OPERATION DESCRIPTION

SN8P1600 provides 2 interrupts. Each operation of the 2 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.

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

GIE: Global interrupt control bit. 0 = disable, 1 = enable.

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

```
B0BSET      FGIE        ; Enable GIE
```

➢ ***Note: Set GIE bit as "1' to enable all interrupt.***

INT0 (P0.0) INTERRUPT OPERATION

The INT0 has falling edge interrupt trigger. For SN8P1600, the INT0 just uses the falling edge to trigger the external interrupt 0. 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 sets the P00IRQ to be “1” and the system into interrupt vector (ORG 8). If the P00IEN = 0, the trigger event just only sets the P00IRQ to be “1” but the system doesn’t get into interrupt vector. Users need to care 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       ; Store ACC value.
        B0MOV       A, PFLAG
        B0MOV       PFLAGBUF, A

        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
        B0XCH       A, ACCBUF       ; Restore ACC value.

        RETI                        ; Exit interrupt vector
```

TC0/TC1 INTERRUPT OPERATION

When the TC0C/TC1C counter occurs overflow, the TC0IRQ/TC1IRQ will be set to “1” however the TC0IEN/TC1IEN is enable or disable. If the TC0IEN = 1, the trigger event sets the TC0IRQ/TC1IRQ to be “1” and the system into interrupt 0vector. If the TC0IEN/TC1IEN = 0, the trigger event will make the TC0IRQ/TC1IEN to be “1” but the system not into interrupt vector. Users need to care 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       ; Store ACC value.
        B0MOV       A, PFLAG
        B0MOV       PFLAGBUF, A

        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
        B0XCH       A, ACCBUF       ; Restore ACC value.

        RETI                        ; Exit interrupt vector
```

MULTI-INTERRUPT OPERATION

In almost conditions, the software designer uses more than one interrupt requests. Processing multi-interrupt request needs to set the priority of these interrupt requests. The IRQ flags of interrupts are controlled by the interrupt event. But the IRQ flag "1" doesn't mean the system to execute the interrupt vector. The IRQ flags can be set "1" 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. OTP is falling edge.
TC0IRQ	TC0C overflow. (SN8P1602/1603)
TC1IRQ	TC1C overflow. (SN8P1604)

There are two works need to do for multi-interrupt conditions. 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 do users check the interrupt request in multi-interrupt situation?**

```
        ORG     8               ; Interrupt vector

        B0XCH   A, ACCBUF       ; Store ACC value.
        B0MOV   A, PFLAG        ; Store PFLAG value
        B0MOV   PFLAGBUF,A


INTP00CHK:                      ; Check INT0 interrupt request
        B0BTS1  FP00IEN         ; Check P00IEN
        JMP     INTTC0CHK       ; Jump check to next interrupt
        B0BTS0  FP00IRQ         ; Check P00IRQ
        JMP     INTP00          ; Jump to INT0 interrupt service routine
INTTC0CHK:                      ; Check TC0 interrupt request
        B0BTS1  FTC0IEN         ; Check TC0IEN
        JMP     INT_EXIT        ; Jump to exit of IRQ
        B0BTS0  FTC0IRQ         ; Check TC0IRQ
        JMP     INTTC0          ; Jump to TC0 interrupt service routine
INT_EXIT:

        B0MOV   A, PFLAGBUF     ; Restore PFLAG value
        B0MOV   PFLAG,A
        B0XCH   A, ACCBUF       ; Restore ACC value.

        RETI                    ; Exit interrupt vector
```

10 I/O PORT

OVERVIEW

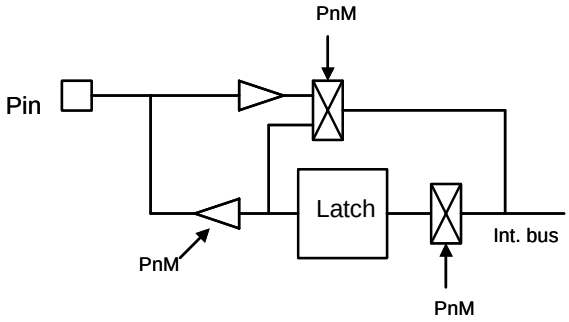
The SN8P1602/1603 provides up to 3 ports for users' application, consisting of one input only port (P0), two I/O ports (P1, P2,). The SN8P1602/1603 is without input pull-up resistors. The direction of I/O port is selected by PnM register. After the system resets, these ports work as input function.

The SN8P1604 provides 4 ports for users' application, consisting of one input port (P0) and three I/O ports (P1,P2,P5). The direction of I/O port is selected by PnM register. After the system resets, these ports work as input port without pull up resistors. If the user want to read-in a signal from I/O pin, it recommends to switch I/O pin as input mode to execute read-in instruction. (B0BTS0 M.b, B0BTS1 M.b or B0MOV A,M). The pull-up resistor can be set up by the code option in the programming phase.

- ***SN8P1602/1603***

Port0 structure

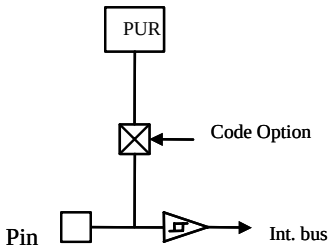
Port1, 2 structure



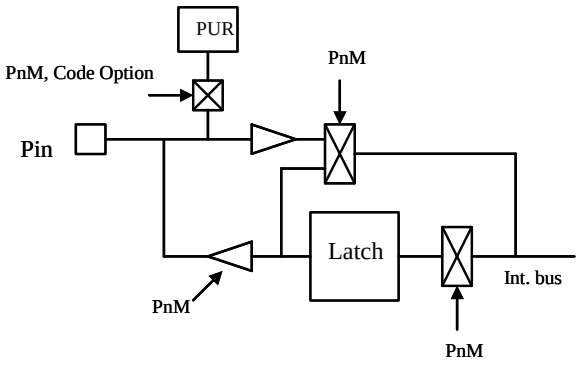
- ***Note: All of the latch output circuits are push-pull structures.***

- ***SN8P1604***

Port0 structure



Port1,2, 5 structure



- ***Note: The pull-up resistor can be set up by the code option in the programming phase.***

I/O PORT FUNCTION TABLE

➢ ***SN8P1602/1603***

Port/Pin	I/O	Function Description	Remark
P0.0	I	General-purpose input function	
		External interrupt (INT0)	Falling edge
		Wakeup for power down mode	Low level
P1.0~P1.4	I/O	General-purpose input/output function	
		Wakeup for power down mode	Low level
P2.0~P2.7	I/O	General-purpose input/output function	

➢ ***Note: The P1.4 enables when the external oscillator is RC type.***

➢ ***SN8P1604***

Port/Pin	I/O	Function Description	Remark
P0.0	I	General-purpose input function	
		External interrupt (INT0)	Falling edge
		Wakeup for power down mode	Low level
P0.1	I	Wakeup for power down mode	Low level
P1.0~P1.7	I/O	General-purpose input/output function	
		Wakeup for power down mode	Low level
P2.0~P2.7	I/O	General-purpose input/output function	
P5.0~P5.3	I/O	General-purpose input/output function	

I/O PORT MODE

The port direction is programmed by PnM register. Port 0 is always input mode. Port 1 and Port 2 can select input or output direction. The each bit of PnM is set to “0”, the I/O pin is input mode. The each bit of PnM is set to “1”, the I/O pin is output mode.

➢ ***SN8P1602/1603***

0C1H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
P1M	0	0	0	P14M	P13M	P12M	P11M	P10M
Read/Write	-	-	-	R/W	R/W	R/W	R/W	R/W
After reset	-	-	-	0	0	0	0	0

➢ ***SN8P1604***

0C1H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
P1M	P17M	P16M	P15M	P14M	P13M	P12M	P11M	P10M
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
After reset	0	0	0	0	0	0	0	0

➢ ***SN8P1602/1603/1604***

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
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
After reset	0	0	0	0	0	0	0	0

➢ ***SN8P1604***

0C5H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
P5M	0	0	0	0	P53M	P52M	P51M	P50M
Read/Write	-	-	-	-	R/W	R/W	R/W	R/W
After reset	-	-	-	-	0	0	0	0

➢ ***The PnM registers are read/write bi-direction registers. Users can program them by bit control instructions (B0BSET, B0BCLR).***

➲ **Example: I/O mode selecting.**

```
CLR     P1M             ; Set all ports to be input mode.
CLR     P2M

MOV     A, #0FFH        ; Set all ports to be output mode.
B0MOV   P1M, A
B0MOV   P2M, A

B0BCLR  P1M.2           ; Set P1.2 to be input mode.

B0BSET  P1M.2           ; Set P1.2 to be output mode.
```

I/O PORT DATA REGISTER

➢ ***SN8P1602/1603***

0D0H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
P0	-	-	-	-	-	-	-	P00
Read/Write	-	-	-	-	-	-	-	R
After reset	-	-	-	-	-	-	-	0

➢ ***SN8P1604***

0D0H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
P0	-	-	-	-	-	-	P01	P00
Read/Write	-	-	-	-	-	-	R	R
After reset	-	-	-	-	-	-	0	0

➢ ***SN8P1602/1603***

0D1H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
P1	-	-	-	P14	P13	P12	P11	P10
Read/Write	-	-	-	R/W	R/W	R/W	R/W	R/W
After reset	-	-	-	0	0	0	0	0

➢ ***SN8P1604***

0D1H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
P1	P17	P16	P15	P14	P13	P12	P11	P10
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
After reset	0	0	0	0	0	0	0	0

➢ ***SN8P1602/1603/1604***

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
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
After reset	0	0	0	0	0	0	0	0

➢ ***SN8P1604***

0D5H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
P5	-	-	-	-	P53	P52	P51	P50
Read/Write	-	-	-	-	R/W	R/W	R/W	R/W
After reset	-	-	-	-	0	0	0	0

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

```
B0MOV   A, P0   ; Read data from Port 0
B0MOV   A, P1   ; Read data from Port 1
B0MOV   A, P2   ; Read data from Port 2
```

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

```
MOV     A, #55H ; Write data 55H to Port 1 and Port 2
B0MOV   P1, A
B0MOV   P2, A
```

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

```
B0BSET  P1.3    ; Set P1.3 and P2.5 to be “1”.
B0BSET  P2.5

B0BCLR  P1.3    ; Set P1.3 and P2.5 to be “0”.
B0BCLR  P2.5
```

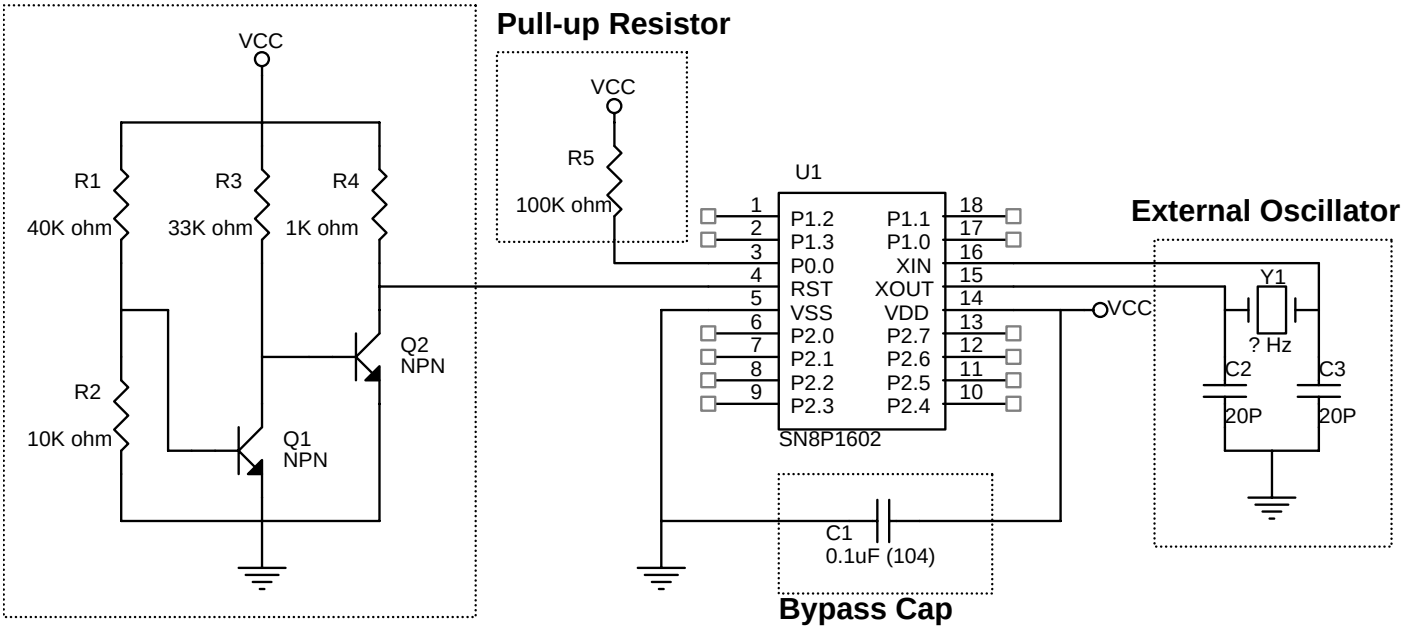
➲ **Example: Port bit test.**

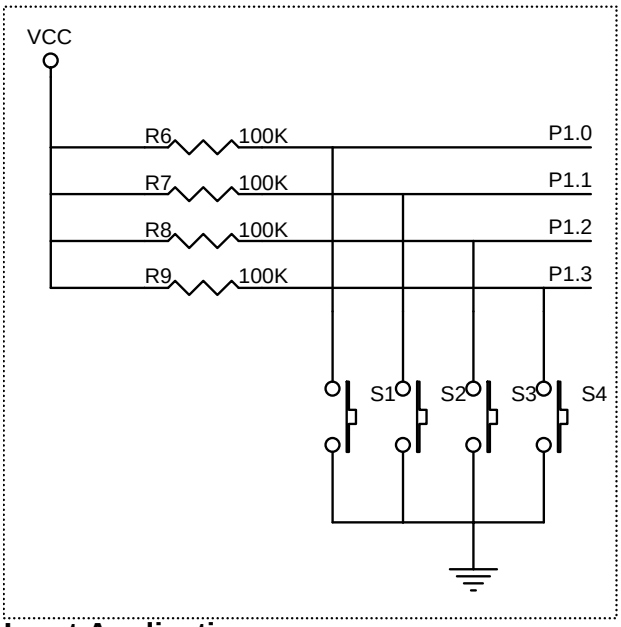
```
B0BTS1  P0.0    ; Bit test 1 for P0.0
.
B0BTS0  P1.2    ; Bit test 0 for P1.2
.
```

11 External Reset Circuit

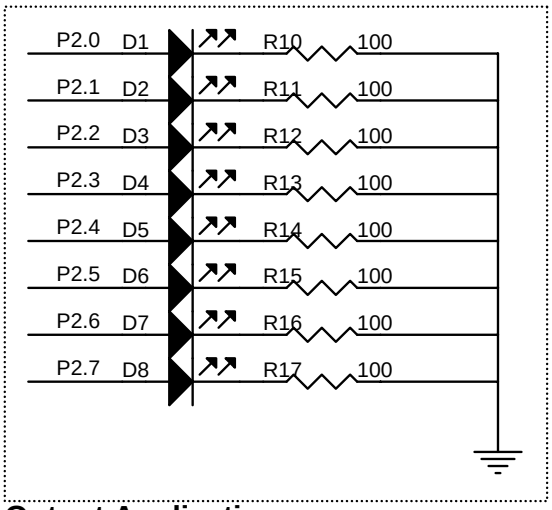
SN8P1602 External Reset Circuit Example

External Reset Circuit





Input Application

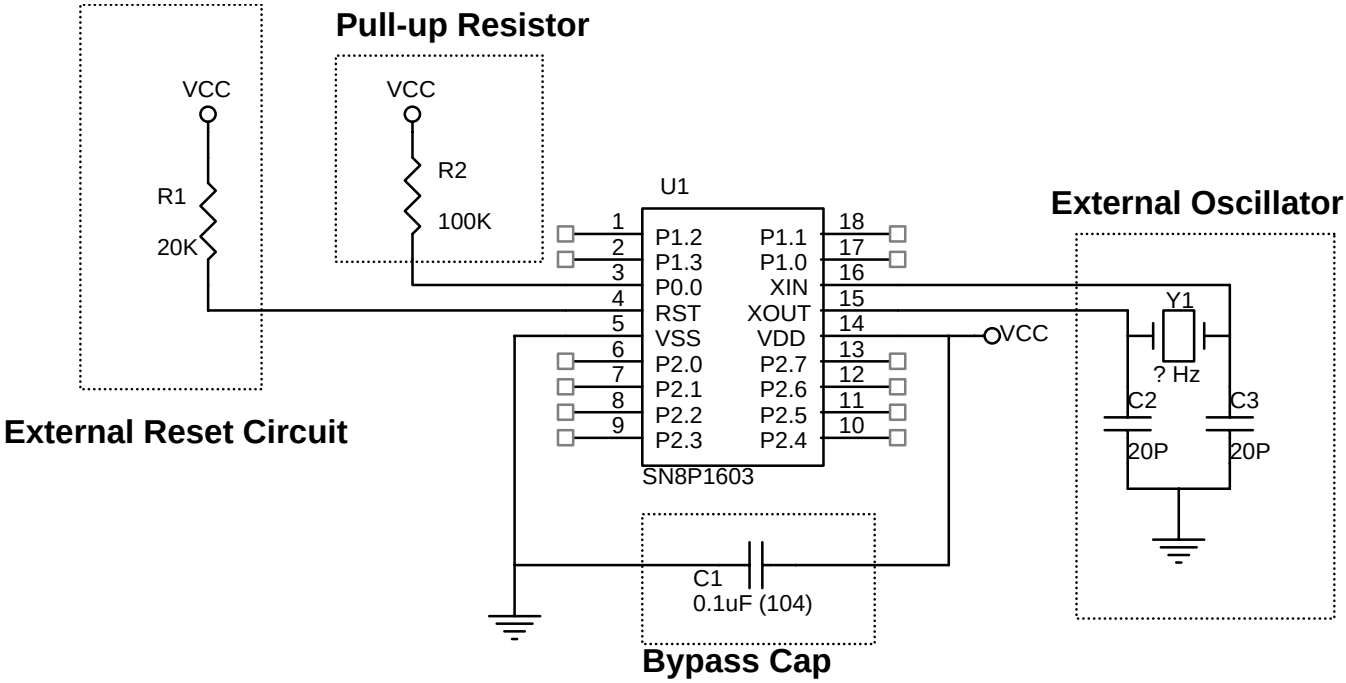


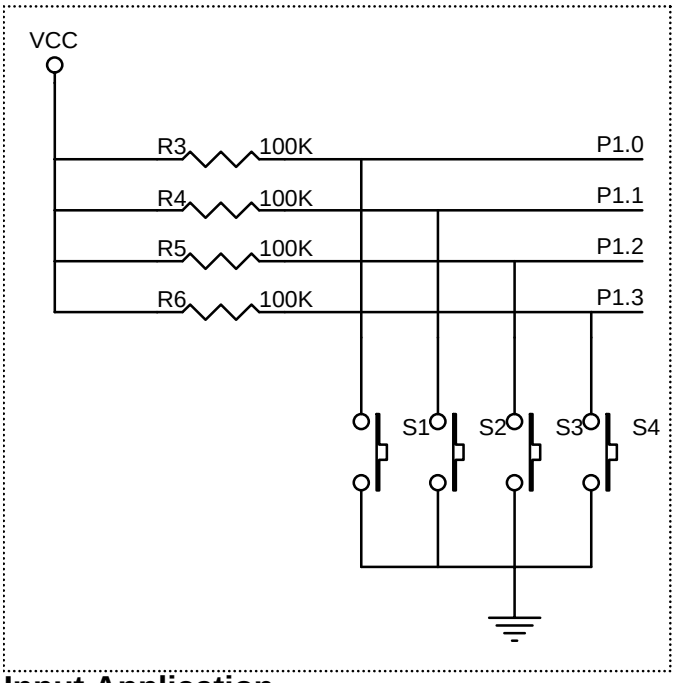
Output Application

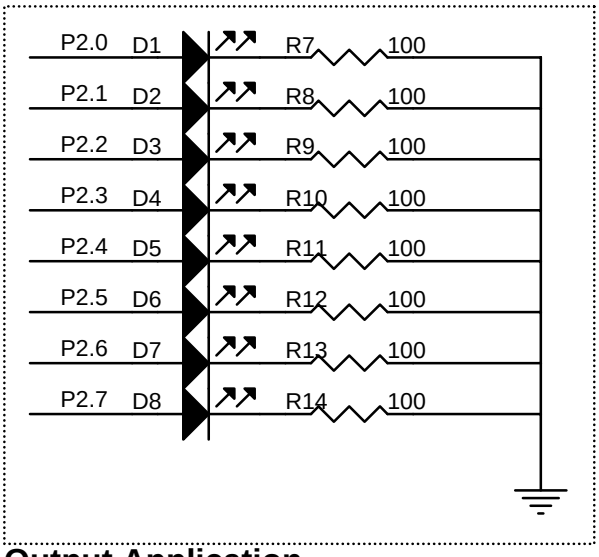
Notice :

The SN8P1602 do not have the internal power on reset circuit. It is very important to connect the external component to implement the reset function. The simple RC circuit does not get a good brownout reset. The External reset circuit in the demo schematic can improve the SN8P1602 power-on/Brown-out reset performance, but it increases the system power consumption (about 200μA).

SN8P1603 External Reset Circuit Example







Notice :

The SN8P1603 have built-in POR(LVD) function. It has a excellent power-on/brown-out reset performance.

SN8P1604

Code Option	Content	Function Description
High_Clk	RC	Low cost RC for external high clock oscillator
	32K X'tal	Low frequency, power saving crystal (e.g. 32.768K) for external high clock oscillator
	12M X'tal	High speed crystal /resonator (e.g. 12M) for external high clock oscillator
	4M X'tal	Standard crystal /resonator (e.g. 3.58M) for external high clock oscillator
High_Clk / 2	Enable	External high clock divided by two, Fosc = high clock / 2
	Disable	Fosc = high clock
OSG	Enable	Enable Oscillator Safe Guard function
	Disable	Disable Oscillator Safe Guard function
Watch_Dog	Enable	Enable Watch Dog function
	Disable	Disable Watch Dog function
LVD	Enable	Enable the low voltage detect function
	Disable	Disable the low voltage detect function
Security	Enable	Enable ROM code Security function
	Disable	Disable ROM code Security function
Pull_Up	Enable	Enable on-chip pull-up resisters
	Disable	Disable on-chip pull-up resisters

Notice:

- ***The LVD function can improve the power on reset and brown-out reset performance. It will increase about extra 100μA current consumption at 5V if LVD is enabled. The minimum working voltage will be affect by the OSG option. It is very important to check this code option.***
- ***Turn on the OSG will improve the EMI performance. But the side effect is to increase the lowest valid working voltage level.***
- ***If users select "32K X'tal" in "High_Clk" option, assembler will force "OSG" to be enabled.***
- ***If users select "RC" in "High_Clk" option, assembler will force "High_Clk / 2" to be enabled.***

12 CODING ISSUE

TEMPLATE CODE

```
;*************************************************************************
; FILENAME  : TEMPLATE.ASM
; AUTHOR    : SONiX
; PURPOSE   : Template Code for SN8X16XX
; REVISION  : 09/01/2002 V1.0   First issue
;*************************************************************************
;*  (c) Copyright 2002, SONiX TECHNOLOGY CO., LTD.
;*************************************************************************

CHIP    SN8P1602                    ; 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          ;Data section start from RAM address 0
    Wk00        DS      1           ;Temporary buffer for main loop
    Iwk00       DS      1           ;Temporary buffer for ISR
    AccBuf      DS      1           ;Accumulater buffer
    PflagBuf    DS      1           ;PFLAG buffer

;-------------------------------------------------------------------------
;                           Bit Variables Definition
;-------------------------------------------------------------------------

    Wk00B0      EQU    Wk00.0       ;Bit 0 of Wk00
    Iwk00B1     EQU    Iwk00.1      ;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
    b0mov       PFLAG,#00h          ;pflag = x,x,x,x,x,c,dc,z
    mov         A,#40h              ;Clear watchdog timer and initial system mode
    b0mov       OSCM,A

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

;-------------------------------------------------------------------------------
; Main routine
;-------------------------------------------------------------------------------
Main:
    b0bset      FWDRST              ;Clear watchdog timer

    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,Wk00
    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
;-----------------------------------

    b0xch       A,AccBuf            ;B0xch instruction do not change C,Z flag

    b0mov       A,PFLAG
    b0mov       PflagBuf,A

;-----------------------------------
; Interrupt service routine
;-----------------------------------

    b0bts0      FP00IRQ
    jmp         INT0isr
    b0bts0      FTC0IRQ
    jmp         TC0isr

;-----------------------------------
; Exit interrupt service routine
;-----------------------------------

IsrExit:

    b0mov       A, PflagBuf
    b0mov       PFLAG, A            ;Restore the PFlag
    b0xch       A,AccBuf            ;Restore the Reg. A
                                    ;B0xch instruction do not change C,Z flag
    reti                            ;Exit the interrupt routine
```

```
;-------------------------------------------------------------------------------
; INT0 interrupt service routine
;-------------------------------------------------------------------------------
INT0isr:
    b0bclr      FP00IRQ

    ;Process P0.0 external interrupt here

    jmp         IsrExit

;-------------------------------------------------------------------------------
; TC0 interrupt service routine
;-------------------------------------------------------------------------------
TC0isr:
    b0bclr      FTC0IRQ

    ;Process TC0 interrupt here


    jmp         IsrExit

;-------------------------------------------------------------------------------
; SysInit
; System initial to define Register, RAM, I/O, Timer......
;-------------------------------------------------------------------------------
SysInit:

    ret

;-------------------------------------------------------------------------------
; ClrRAM
; Use index @YZ to clear RAM (00h~2Fh)
;-------------------------------------------------------------------------------

ClrRAM:

    clr         Y                   ;
    b0mov       Z,#0x2f             ;Set @YZ address from 2fh

ClrRAM10:
    clr         @YZ                 ;Clear @YZ content
    decms       Z                   ;z = z – 1 , skip next if z=0
    jmp         ClrRAM10
    clr         @YZ                 ;Clear address $00

    ret

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

CHIP DECLARATION IN ASSEMBLER

Assembler	OTP Device Part Number	MASK Device Part Number
CHIP SN8P1602	SN8P1602	SN8A1602A
CHIP SN8P1603	SN8P1603	SN8A1602A
CHIP SN8P1604	SN8P1604	SN8A1604A

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.
PWM1	Set PWM1 (P5.3) pin as output mode.
Interrupt	Do not enable interrupt before initializing RAM.
Non-Used I/O	Non-used I/O ports should be set as output low mode or pull-up at input mode to save current consumption.
Sleep Mode	Enable on-chip pull-up resisters of port 0 and port 1 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.
Noisy Immunity	1. Enable OSG and High_Clk / 2 code option together 2. Enable the watchdog option to protect system crash. 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. 5. Enable the LVD option to improve the power on reset or brown-out reset performance

13 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 for Working registers R, Y, Z , RBANK & PFLAG)	-	-	-	1
	XCH	A,M	A ←→M	-	-	-	1
	B0XCH	A,M	A ←→M (bank 0)	-	-	-	1
	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
R	ADD	A,M	A ← A + M, if occur carry, then C=1, else C=0	√	√	√	1
I	ADD	M,A	M ← A + M, if occur carry, then C=1, else C=0	√	√	√	1
T	B0ADD	M,A	M (bank 0) ← M (bank 0) + A, if occur carry, then C=1, else C=0	√	√	√	1
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
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
C	SUB	A,I	A ← A - I, if occur borrow, then C=0, else C=1	√	√	√	1
	DAA		To adjust ACC's data format from HEX to DEC.	√	-	-	1
	AND	A,M	A ← A and M	-	-	√	1
L	AND	M,A	M ← A and M	-	-	√	1
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
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
	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
R	RRC	M	A ← RRC M	√	-	-	1
O	RRCM	M	M ← RRC M	√	-	-	1
C	RLC	M	A ← RLC M	√	-	-	1
E	RLCM	M	M ← RLC M	√	-	-	1
S	CLR	M	M ← 0	-	-	-	1
S	BCLR	M.b	M.b ← 0	-	-	-	1
	BSET	M.b	M.b ← 1	-	-	-	1
	B0BCLR	M.b	M(bank 0).b ← 0	-	-	-	1
	B0BSET	M.b	M(bank 0).b ← 1	-	-	-	1
	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 + 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 + S
H	BTS0	M.b	If M.b = 0, then skip next instruction	-	-	-	1 + S
	BTS1	M.b	If M.b = 1, then skip next instruction	-	-	-	1 + S
	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
C							

Note: Any instruction that read/write from OSCM, will add an extra cycle.

14 ELECTRICAL CHARACTERISTICS

ABSOLUTE MAXIMUM RATINGS

(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) ..... -20°C ~ +70°C
Storage ambient temperature (Tstor) ..... -30°C ~ +125°C
Power consumption (Pc) ..... 500 mW

STANDARD ELECTRICAL CHARACTERISTICS

SN8P1602

(All of voltages referenced to Vss, Vdd = 5.0V, fosc = 3.579545 MHz, ambient temperature is 25℃ unless otherwise notice.)

PARAMETER	SYM.	DESCRIPTION		MIN.	TYP.	MAX.	UNIT
Operating voltage	Vdd	Normal mode, Vpp = Vdd (Check OSG note)		2.2	5.0	5.5	V
		Programming mode, Vpp = 12.5V		4.5	5.0	5.5	
RAM Data Retention voltage	Vdr			-	1.5	-	V
Input Low Voltage	ViL1	All input pins except those specified below		Vss	-	0.3Vdd	V
	ViL2	Input with Schmitt trigger buffer - Port0		Vss	-	0.2Vdd	V
	ViL3	Reset pin ; Xin (in RC mode)		Vss	-	0.3Vdd	V
	ViL4	Xin (in X'tal mode)		Vss	-	0.3Vdd	V
Input High Voltage	ViH1	All input pins except those specified below		0.7Vdd	-	Vdd	V
	ViH2	Input with Schmitt trigger buffer –Port0		0.8Vdd	-	Vdd	V
	ViH3	Reset pin ; Xin (in RC mode)		0.9Vdd	-	Vdd	V
	ViH4	Xin (in X'tal mode)		0.7Vdd	-	Vdd	V
Reset pin leakage current	Ilekg	Vin = Vdd		-	-	1	uA
I/O port input leakage current	Ilekg	Pull-up resistor disable, Vin = Vdd		-	-	2	uA
Port1 output source current	IoH	Vop = Vdd – 0.5V		-	15	-	mA
sink current	IoL	Vop = Vss + 0.5V		-	15	-	
Port2 output source current	IoH	Vop = Vdd – 0.5V		-	15	-	mA
sink current	IoL	Vop = Vss + 0.5V		-	15	-	
INTn trigger pulse width	Tint0	INT0 interrupt request pulse width		2/fcpu	-	-	cycle
Oscillator Frequency	Fosc	Crystal type or ceramic resonator		32768	4M	16M	Hz
		VDD = 3V, RC type for external mode		-	6M	-	
		VDD = 5V, RC type for external mode		-	10M	-	
Supply Current (LVD OFF)	Idd1	Run Mode	Vdd= 5V 4Mhz	-	7	15	mA
			Vdd= 3V 4Mhz	-	1.7	3.5	mA
			Vdd= 3V 32768Hz	-	50	100	uA
	Idd2	Slow mode (Stop High Clock)	Vdd= 5V 32KHz Int RC	-	80	150	uA
			Vdd= 3V 16KHz Int RC	-	15	30	uA
	Idd3	Sleep mode	Vdd= 5V	-	9	18	uA
			Vdd= 3V (SN8P1602)	-	2.5	6	uA
LVD Detect Voltage	Vdet	Low voltage detect level		2.4	2.5	2.8	V
LVD current	Ilvd	LVD enable operating current		-	100	180	uA

SN8P1603

(All of voltages referenced to Vss, Vdd = 5.0V, fosc = 3.579545 MHz, ambient temperature is 25°C unless otherwise notice.)

PARAMETER	SYM.	DESCRIPTION		MIN.	TYP.	MAX.	UNIT
Operating voltage	Vdd	Normal mode, Vpp = Vdd (Check OSG note)		3	5.0	5.5	V
		Programming mode, Vpp = 12.5V		4.5	5.0	5.5	
RAM Data Retention voltage	Vdr			-	1.5	-	V
Input Low Voltage	ViL1	All input pins except those specified below		Vss	-	0.3Vdd	V
	ViL2	Input with Schmitt trigger buffer - Port0		Vss	-	0.2Vdd	V
	ViL3	Reset pin ; Xin (in RC mode)		Vss	-	0.3Vdd	V
	ViL4	Xin (in X'tal mode)		Vss	-	0.3Vdd	V
Input High Voltage	ViH1	All input pins except those specified below		0.7Vdd	-	Vdd	V
	ViH2	Input with Schmitt trigger buffer –Port0		0.8Vdd	-	Vdd	V
	ViH3	Reset pin ; Xin (in RC mode)		0.9Vdd	-	Vdd	V
	ViH4	Xin (in X'tal mode)		0.7Vdd	-	Vdd	V
Reset pin leakage current	Ilekg	Vin = Vdd		-	-	1	uA
I/O port input leakage current	Ilekg	Pull-up resistor disable, Vin = Vdd		-	-	2	uA
Port1 output source current	IoH	Vop = Vdd – 0.5V		-	15	-	mA
sink current	IoL	Vop = Vss + 0.5V		-	15	-	
Port2 output source current	IoH	Vop = Vdd – 0.5V		-	15	-	mA
sink current	IoL	Vop = Vss + 0.5V		-	15	-	
INTn trigger pulse width	Tint0	INT0 interrupt request pulse width		2/fcpu	-	-	cycle
Oscillator Frequency	Fosc	Crystal type or ceramic resonator		32768	4M	16M	Hz
		VDD = 3V, RC type for external mode		-	6M	-	
		VDD = 5V, RC type for external mode		-	10M	-	
Supply Current (LVD ON)	Idd1	Run Mode	Vdd= 5V 4Mhz	-	7.5	15	mA
			Vdd= 3V 4Mhz	-	1.8	3.6	mA
			Vdd= 3V 32768Hz	-	130	300	uA
	Idd2	Slow mode (Stop High Clock)	Vdd= 5V 32KHz Int RC	-	300	600	uA
			Vdd= 3V 16KHz Int RC	-	100	200	uA
	Idd3	Sleep mode	Vdd= 5V	-	200	400	uA
			Vdd= 3V (SN8P1603)	-	70	150	uA
LVD Detect Voltage	Vdet	Low voltage detect level		2.4	2.5	2.8	V

SN8P1604

(All of voltages referenced to Vss, Vdd = 5.0V, fosc = 3.579545 MHz, ambient temperature is 25℃ unless otherwise notice.)

PARAMETER	SYM.	DESCRIPTION		MIN.	TYP.	MAX.	UNIT
Operating voltage	Vdd	Normal mode, Vpp = Vdd (Check OSG note)		2.2	5.0	5.5	V
		Programming mode, Vpp = 12.5V		4.5	5.0	5.5	
RAM Data Retention voltage	Vdr			-	1.5	-	V
Input Low Voltage	ViL1	All input pins except those specified below		Vss	-	0.3Vdd	V
	ViL2	Input with Schmitt trigger buffer - Port0		Vss	-	0.2Vdd	V
	ViL3	Reset pin ; Xin (in RC mode)		Vss	-	0.3Vdd	V
	ViL4	Xin (in X'tal mode)		Vss	-	0.3Vdd	V
Input High Voltage	ViH1	All input pins except those specified below		0.7Vdd	-	Vdd	V
	ViH2	Input with Schmitt trigger buffer –Port0		0.8Vdd	-	Vdd	V
	ViH3	Reset pin ; Xin (in RC mode)		0.9Vdd	-	Vdd	V
	ViH4	Xin (in X'tal mode)		0.7Vdd	-	Vdd	V
Reset pin leakage current	Ilekg	Vin = Vdd		-	-	1	uA
I/O port pull-up resistor	Rup	Vin = Vss , Vdd = 5V		-	50	-	KΩ
I/O port input leakage current	Ilekg	Pull-up resistor disable, Vin = Vdd		-	-	2	uA
Port1 output source current	IoH	Vop = Vdd – 0.5V		-	15	-	mA
sink current	IoL	Vop = Vss + 0.5V		-	15	-	
Port2 output source current	IoH	Vop = Vdd – 0.5V		-	15	-	mA
sink current	IoL	Vop = Vss + 0.5V		-	15	-	
INTn trigger pulse width	Tint0	INT0 ~ INT2 interrupt request pulse width		2/fcpu	-	-	cycle
Oscillator Frequency	Fosc	Crystal type or ceramic resonator		32768	4M	16M	Hz
		VDD = 3V, RC type for external mode		-	6M	-	
		VDD = 5V, RC type for external mode		-	10M	-	
Supply Current (LVD OFF)	Idd1	Run Mode	Vdd= 5V 4Mhz	-	7	15	mA
			Vdd= 3V 4Mhz	-	1.7	3.5	mA
			Vdd= 3V 32768Hz	-	50	100	uA
	Idd2	Slow mode (Stop High Clock)	Vdd= 5V 32KHz Int RC	-	80	150	uA
			Vdd= 3V 16KHz Int RC	-	15	30	uA
	Idd3	Sleep mode	Vdd= 5V	-	9	18	uA
			Vdd= 3V (SN8P1602)	-	2.5	6	uA
LVD Detect Voltage	Vdet	Low voltage detect level		2.4	2.5	2.8	V
LVD current	Ilvd	LVD enable operating current		-	100	180	uA

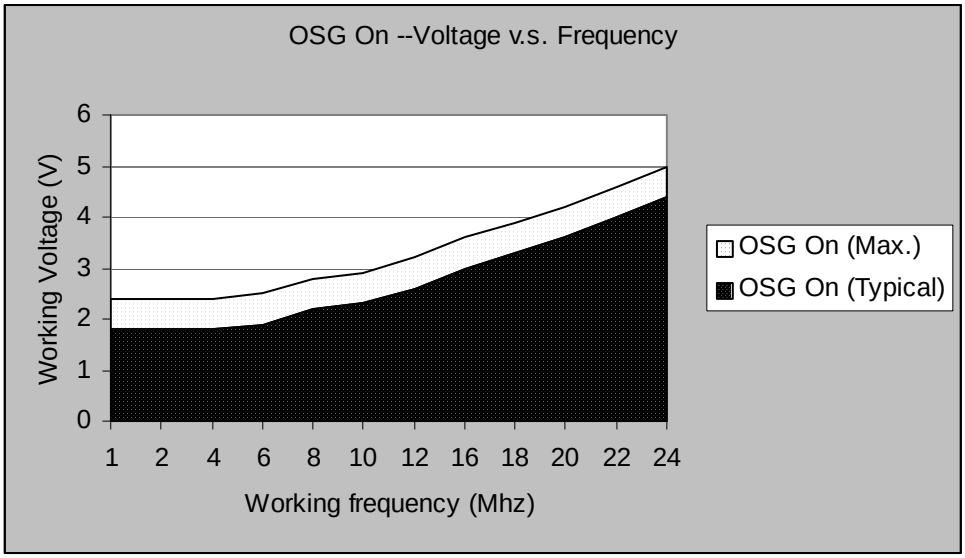
OSG Note :

Notice : The minimum working voltage is depended on the working frequency and OSG status.

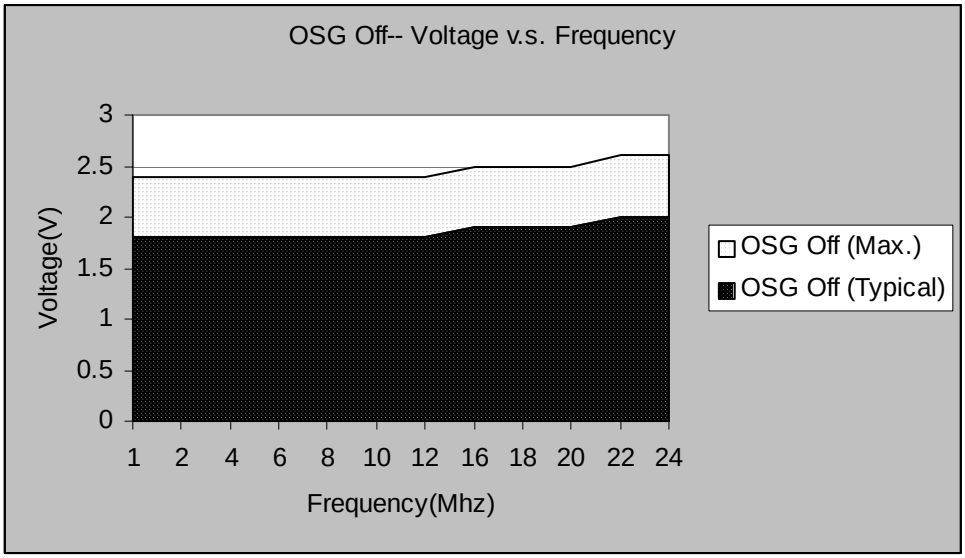
Turn on the OSG code option in the SN8ASM will improve the EMI performance, but it will increase the lowest valid working voltage level.

The safe operating voltage must above the curve (max.) to get a stable power environment.

Typical VDD vs. Frequency (OSG On)



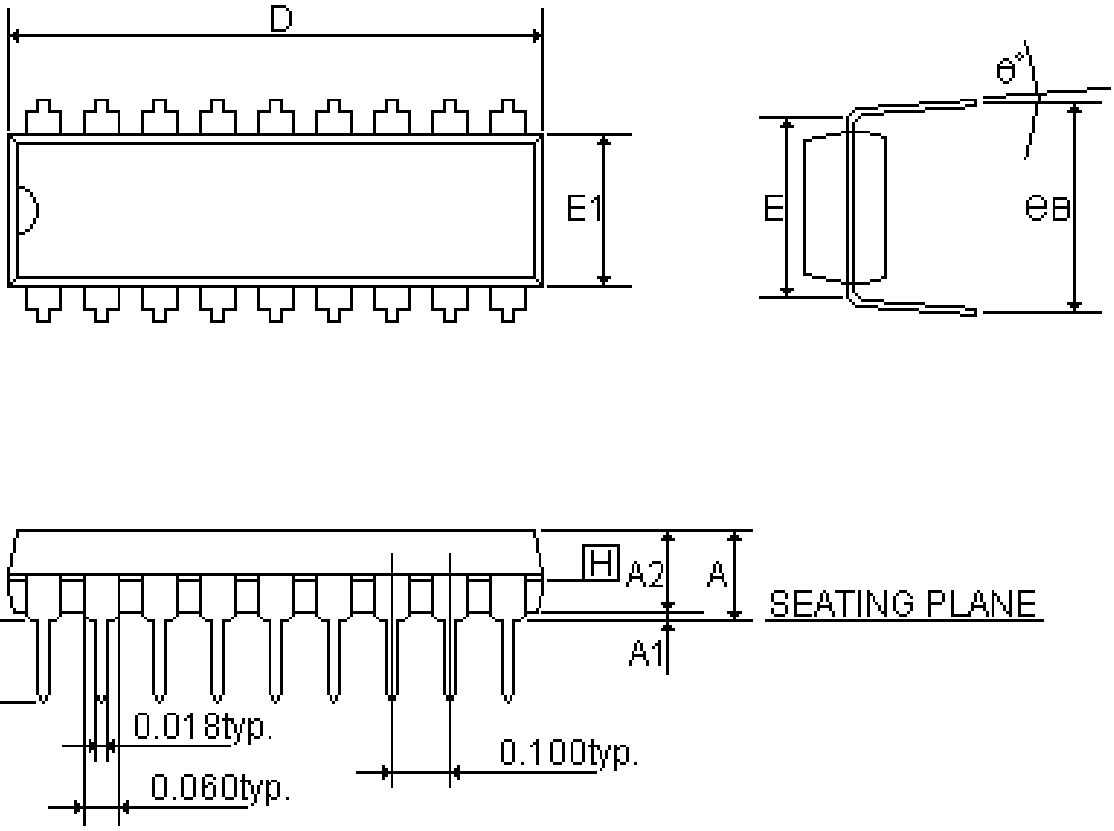
Typical VDD vs. Frequency (OSG Off)



Notice : The system working frequency is only warranty under 16Mhz.

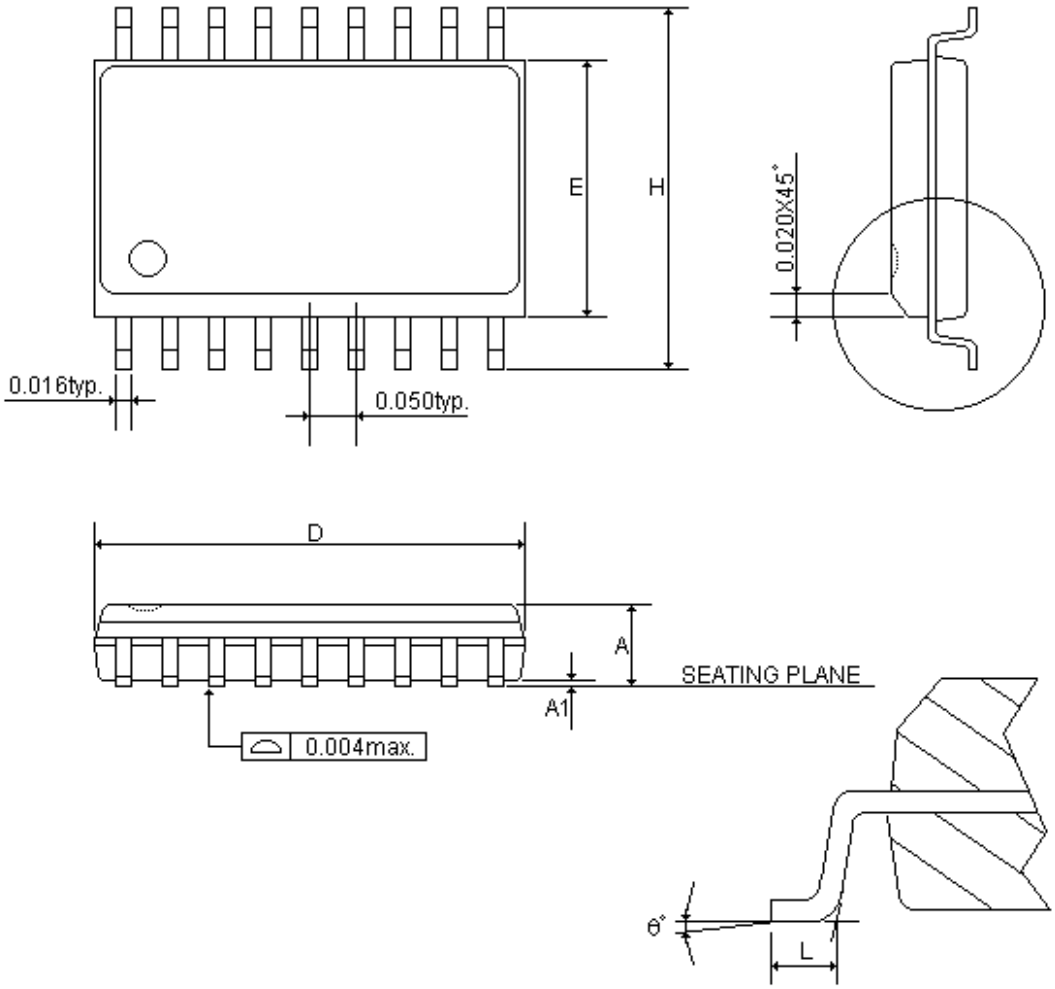
15 PACKAGE INFORMATION

P-DIP 18 PIN



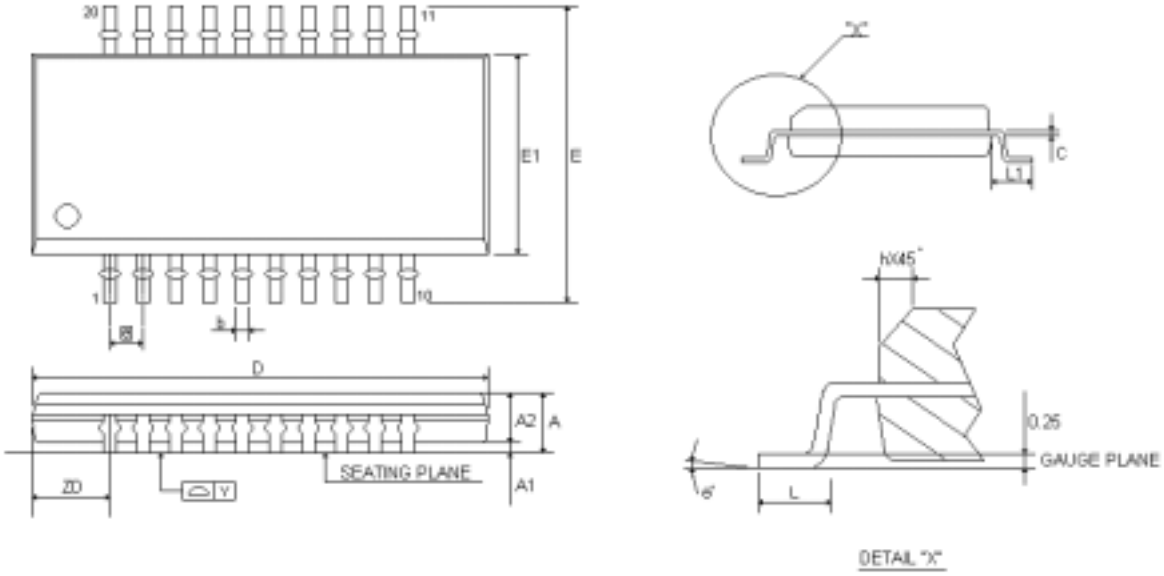
SYMBOLS	MIN	NOR	MAX	MIN	NOR	MAX
	(inch)			(mm)		
A	-	-	0.210	-	-	5.334
A1	0.015	-	-	0.381	-	-
A2	0.125	0.130	0.135	3.175	3.302	3.429
D	0.880	0.900	0.920	22.352	22.860	23.368
E		0.300			7.620	
E1	0.245	0.250	0.255	6.223	6.350	6.477
L	0.115	0.130	0.150	2.921	3.302	3.810
eB	0.335	0.355	0.375	8.509	9.017	9.525
θ°	0°	7°	15°	0°	7°	15°

SOP 18 PIN



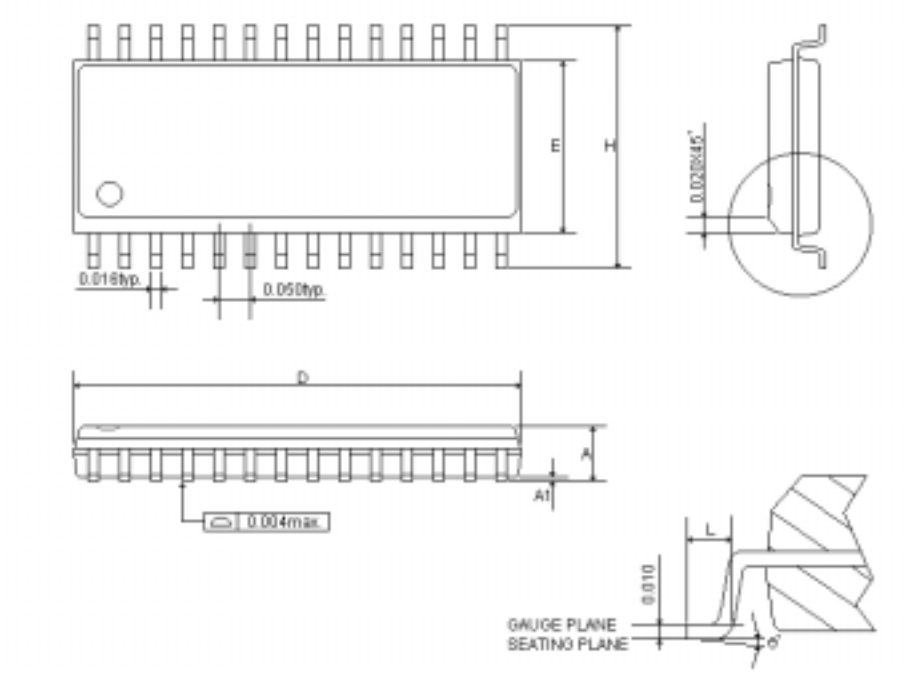
SYMBOLS	MIN	NOR	MAX	MIN	NOR	MAX
	(inch)			(mm)		
A	0.093	0.099	0.104	2.362	2.502	2.642
A1	0.004	0.008	0.012	0.102	0.203	0.305
D	0.447	0.455	0.463	11.354	11.557	11.760
E	0.291	0.295	0.299	7.391	7.493	7.595
H	0.394	0.407	0.419	10.008	10.325	10.643
L	0.016	0.033	0.050	0.406	0.838	1.270
θ°	0°	4°	8°	0°	4°	8°

SSOP 20 PIN



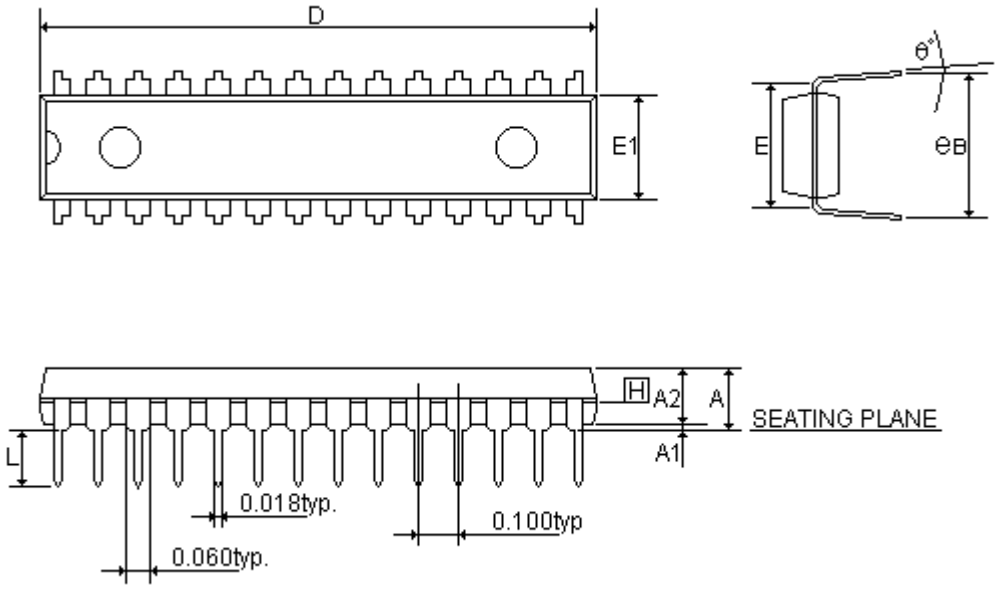
SYMBOLS	MIN	NOR	MAX	MIN	NOR	MAX
	(inch)			(mm)		
A	0.053	0.063	0.069	1.350	1.600	1.750
A1	0.004	0.006	0.010	0.100	0.150	0.250
A2	-	-	0.059	-	-	1.500
b	0.008	0.010	0.012	0.200	0.254	0.300
c	0.007	0.008	0.010	0.180	0.203	0.250
D	0.337	0.341	0.344	8.560	8.660	8.740
E	0.228	0.236	0.244	5.800	6.000	6.200
E1	0.150	0.154	0.157	3.800	3.900	4.000
[e]		0.025			0.635	
h	0.010	0.017	0.020	0.250	0.420	0.500
L	0.016	0.025	0.050	0.400	0.635	1.270
L1	0.039	0.041	0.043	1.000	1.050	1.100
ZD		0.059			1.500	
Y	-	-	0.004	-	-	0.100
θ°	0°	-	8°	0°	-	8°

SOP28PIN



SYMBOLS	MIN	NOR	MAX	MIN	NOR	MAX
	(inch)			(mm)		
A	0.093	0.099	0.104	2.362	2.502	2.642
A1	0.004	0.008	0.012	0.102	0.203	0.305
D	0.697	0.705	0.713	17.704	17.907	18.110
E	0.291	0.295	0.299	7.391	7.493	7.595
H	0.394	0.407	0.419	10.008	10.325	10.643
L	0.016	0.033	0.050	0.406	0.838	1.270
θ°	0°	4°	8°	0°	4°	8°

SK-DIP28PIN



SYMBOLS	MIN	NOR	MAX	MIN	NOR	MAX
	(inch)			(mm)		
A	-	-	0.210	-	-	5.334
A1	0.015	-	-	0.381	-	-
A2	0.114	0.130	0.135	2.896	3.302	3.429
D	1.390	1.390	1.400	35.306	35.306	35.560
E	0.310			7.874		
E1	0.283	0.288	0.293	7.188	7.315	7.442
L	0.115	0.130	0.150	2.921	3.302	3.810
eB	0.330	0.350	0.370	8.382	8.890	9.398
θ°	0°	7°	15°	0°	7°	15°

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