SN8P1800 Series

USER'S MANUAL

General Release Specification

SN8P1808

SONiX 8-Bit Micro-Controller

AMENDMENT HISTORY

Version	Date	Description
VER 1.90	Sep. 2002	V1.90 first issue
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. Add ADC grade. 4. Change bit name and initial value of RBANK register. 5. Change “ACC can’t be access by “B0MOV” instruction” to “ACC can’t be access by “B0MOV” instruction during the instant addressing mode”. 6. Correct the description of STKnH. 7. Correct the bit definition of INTEN register. 8. Change “The low-speed clock frequency is supplied through on-chip RC oscillator circuit” to “The low-speed clock frequency is supplied through external low clock oscillator (32.768K) by crystal or RC mode”. 9. Change all “internal low-speed clock” to “external low-speed clock”. 10. Correct the description of “TC0 CLOCK FREQUENCY OUTPUT” section. 11. Correct the description of “TC1 CLOCK FREQUENCY OUTPUT” section. 12. SCKMD = 1 means SIO is in SLAVE mode. SCKMD = 0 means SIO is in MASTER mode. 13. Remove “SIO clock and SPI clock are compatible”. 14. Remove this line: “B0MOV A, P2”. P2 of SN8P1808 is output only. 15. Note: The clock source of LCD driver is external low clock. 16. Modify the description ADR register. 17. Modify ADB’s output data table. 18. Correct an error of template code: “b0bclr FWDRST” → “b0bset FWDRST”. 19. Add a notice about OSCM register access cycle. 20. Add slow mode (high clock stop and LVD OFF) operating current.
VER 1.94	Sep. 2003	1. Correct RAM Bank value. 2. Correct EOC description. 3. Correct watchdog timer overflow time. 4. Correct POP operand. 5. Correct ADCKS table. 6. Modify figure 11-1 (adjust circuit of LCD contrast) and related description. 7. Add new section about checksum calculate must avoid 04H~07H 8. Correct description of Port6 as I/O port in Chapter 10. 9. Add WTCKS bit in OSCM register 10. Add TC0CKS/TC1CKS in TC0M/TC1M 11. Reserved Last 16 word ROM addresses

		12. Add SIOM Table and Note for SIO Rate 13. Remove register bit description 14. Modify TC0M description 15. Modify TC1M description 16. Modify PWM description 17. Modify ADC Frequency description. 18. Change watchdog over flow table. 19. Change Code option table Chapter 2 20. Add ADC current consumption 21. Add LVD detect voltage 22. Modify electrical characteristic table 23. Remove approval sheet. 24. Remove PCB layout notice section. 25. Modify the description of INTRQ register. 26. Modify the calculation formula of SIOR and SIO clock.

Table of Contents

1 PRODUCT OVERVIEW

GENERAL DESCRIPTION

The SN8P1800 is an series of 8-bit micro-controller including SN8P1808. This series is utilized with CMOS technology fabrication and featured with low power consumption and high performance by its unique electronic structure.

These chips are designed with the excellent IC structure including the large program memory OTP ROM, the massive data memory RAM, one 8-bit basic timer (T0), two 8-bit timer counters (TC0, TC1), high performance of real time clock timer (RTC) , a watchdog timer, up to seven interrupt sources (T0, TC0, TC1, SIO, INT0, INT1, INT2), an 8-channel ADC converter with 8-bit/12-bit resolution, two channel PWM output (PWM0, PWM1), tw0 channel buzzer output (BZ0, BZ1) and 8-level stack buffers.

Besides, the user can choose desired oscillator configurations for the controller. There are four oscillator configurations to select for generating system clock, including High/Low speed crystal, ceramic resonator or cost-saving RC. SN8P1800 series is a dual clock system using a hi-speed crystal for normal mode operation and an external low speed crystal for slow mode, real time clock and LCD function.

FEATURES SELECTION TABLE

CHIP	ROM	RAM	Stack	Timer			I/O	ADC	DAC	PWM	SIO	Wakeup	Package
				T0	TC0	TC1				Buzzer		Pin no.	
SN8P1808	4K*16	256	8	V	V	V	47	8ch		2	1	10	LQPF64

Table 1-1. Selection Table of SN8P1800

ADC GRADE TABLE

CHIP	PARAMETER	MIN	MAX	UNITS	REMARK
SN8P1808	Resolution		12	Bits	
	No Mission Code	8	12	Bits	
	Differential Nonlinearity (DNL)		16	LSB	
SN8P1808-12	Resolution		12	Bits	
	No Mission Code	10	12	Bits	
	Differential Nonlinearity (DNL)		4	LSB	

Table 1-2. ADC Grade Table

FEATURES

➢ ***SN8P1808***

◆ ***Memory configuration***
OTP ROM size: 4K * 16 bits
RAM size: 256 * 8 bits (bank 0 and bank 1)
LCD RAM size: 24 * 3 bits

◆ ***I/O pin configuration***
Input only: P0, P3
Output only: P2 shared with LCD segment
Bi-directional: P1, P4, P5, P6
Wakeup: P0, P1
Pull-up resisters: P0, P1, P3, P4, P5, P6
External interrupt: P0
Port 3 shared with LCD segment
All LCD pins shared with the I/O pins

◆ ***59 powerful instructions***
Four clocks per instruction cycle
All of instructions are one word length.
Most of instructions are one cycle only.
Maximum instruction cycle is two.
All ROM area JMP instruction.
All ROM area lookup table function (MOVC)
Support hardware multiplier (MUL).

◆ ***Seven interrupt sources***
Four internal interrupts: T0, TC0, TC1, SIO
Three external interrupts: INT0, INT1, INT2

◆ ***A real time clock timer***
◆ ***An 8-bit basic timer with green mode wakeup function***
◆ ***Two 8-bit timer counters with PWM or buzzer***
◆ ***On chip watchdog timer***

◆ ***Eight levels stack buffer***
◆ ***An 8-channel ADC with 8-bit/12-bit resolution***
◆ ***SIO function***
◆ ***LCD driver: 1/3 duty, 1/2 bias. 3 common * 24 segment***

◆ ***Dual clock system offers four operating modes***
External high clock: RC type up to 10 MHz
External high clock: Crystal type up to 16 MHz
External Low clock: Crystal 32768Hz
Normal mode: Both high and low clock active.
Slow mode: Low clock only.
Sleep mode: Both high and low clock stop.
Green mode: Periodical wakeup by timer.

◆ ***Package***
Chip form: LQFP 64 pins

SYSTEM BLOCK DIAGRAM

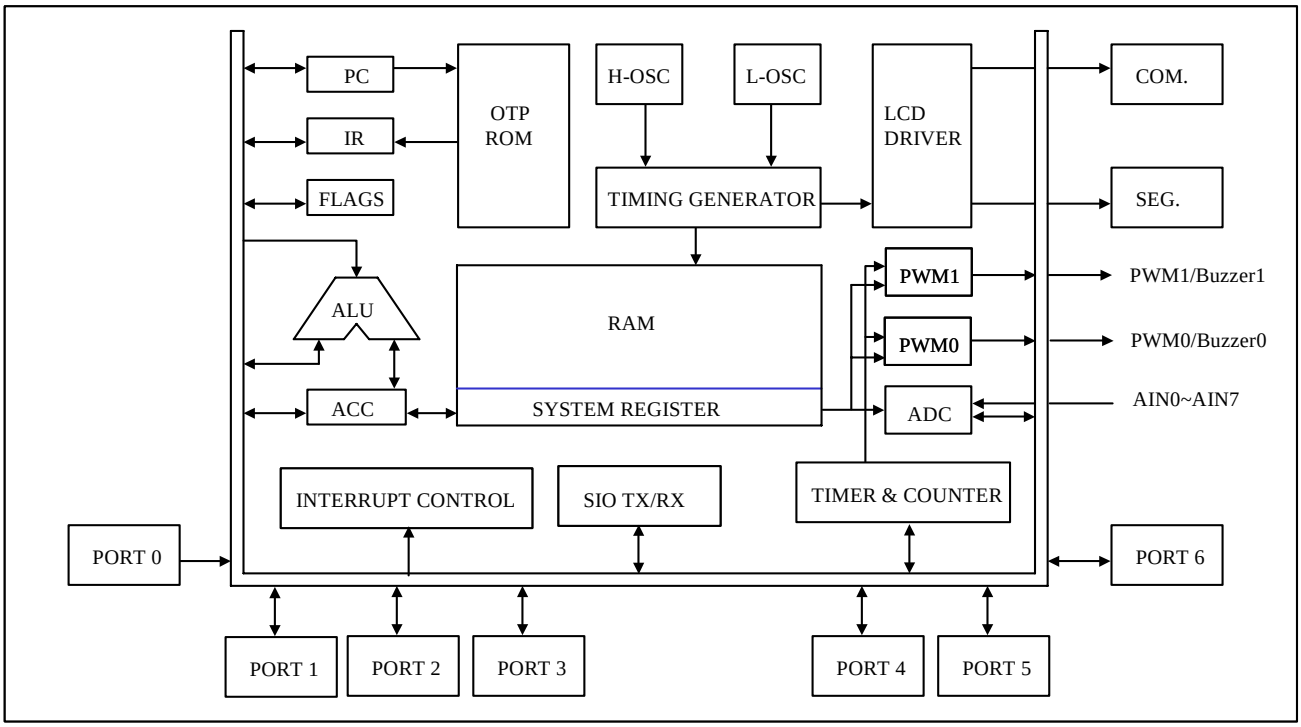
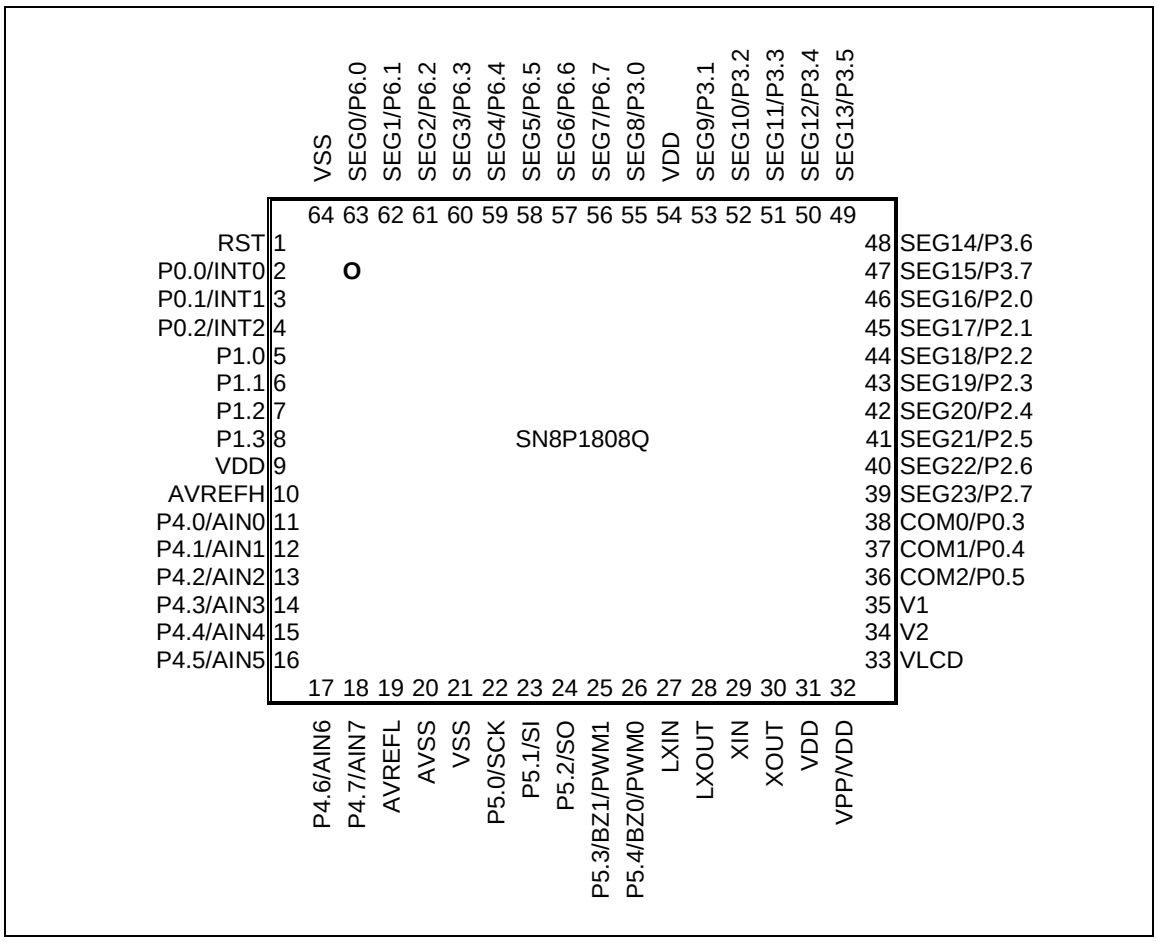


Figure 1-1.Simplified System Block Diagram

PIN ASSIGNMENT

SN8P1808 (LQFP64)



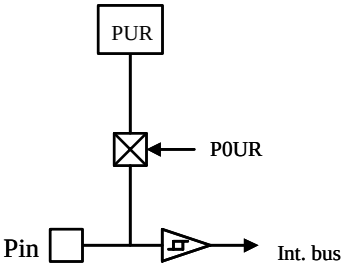
PIN DESCRIPTIONS

PIN NAME	TYPE	DESCRIPTION
VDD, VSS	P	Power supply input pins for digital circuit.
AVDD, AVSS	P	Power supply input pins for analog circuit.
VPP	P	OTP ROM programming pin. Connect to VDD in normal operation.
RST	I	System reset input pin. Schmitt trigger structure, active “low”, normal stay to “high”.
XIN, XOUT	I, O	External oscillator pins. RC mode from XIN.
LXIN, LXOUT	I, O	Low speed (32768 Hz) oscillator pins. RC mode from LXIN.
P0.0 / INT0	I	Port 0.0 and shared with INT0 trigger pin. (Schmitt trigger) / Built-in pull-up resisters.
P0.1 / INT1	I	Port 0.1 and shared with INT1 trigger pin. (Schmitt trigger) / Built-in pull-up resisters.
P0.2 / INT2	I	Port 0.2 and shared with INT2 trigger pin. (Schmitt trigger) / Built-in pull-up resisters.
P0.3~ P0.5	I	Port 0.3~Port 0.5 input pins and shared with LCD’s COM0~COM2. (Schmitt trigger). Built-in pull-up resisters.
P1.0 ~ P1.3	I/O	Port 1.0~Port 1.3 bi-direction pins / Built-in pull-up resisters.
P2.0 ~ P2.7	O	Port 2.0~Port 2.7 output only port and shared with LCD’s SEG16~SEG23.
P3.0 ~ P3.7	I	Port 3.0~Port 3.7 input port with pull-up resister and shared with LCD’s SEG8~SEG15. Built-in pull-up resisters.
P4.0 ~ P4.7	I/O	Port 4.0~Port 4.7 bi-direction pins / Built-in pull-up resisters.
P5.0 / SCK	I/O	Port 5.0 bi-direction pin and SIO’s clock input/output / Built-in pull-up resisters.
P5.1 / SI	I/O	Port 5.1 bi-direction pin and SIO’s data input / Built-in pull-up resisters.
P5.2 / SO	I/O	Port 5.2 bi-direction pin and SIO’s data output / Built-in pull-up resisters.
P5.3 / BZ1 / PWM1	I/O	Port 5.3 bi-direction pin, TC1 ÷ 2 signal output pin or PWM1 output pin. Built-in pull-up resisters.
P5.4 / BZ0 / PWM0	I/O	Port 5.4 bi-direction pin, TC0 ÷ 2 signal output pin or PWM0 output pin. Built-in pull-up resisters.
P6.0 ~ P6.7	I/O	Port 6.0 ~ Port 6.7 bi-direction pins and shared with LCD’s SEG0~SEG7. Enable pull-up resisters in input mode automatically.
AIN0 ~ AIN7	I	Analog signal input pins for ADC converter.
COM0 ~ COM2	O	LCD driver common pins.
SEG0 ~ SEG23	O	LCD driver segment pins.
AvrefH,AverfL	I	ADC’s reference high / low voltage input pins.

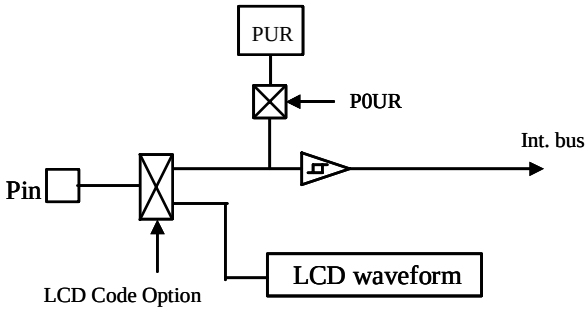
Table 1-3. SN8P1800 Pin Description

PIN CIRCUIT DIAGRAMS

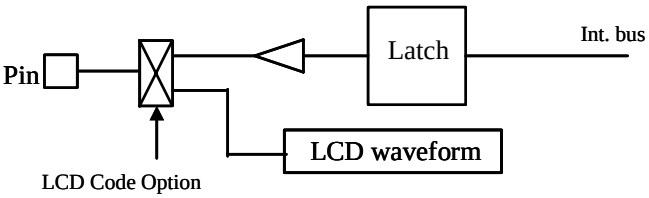
Port0.0~P0.2 structure



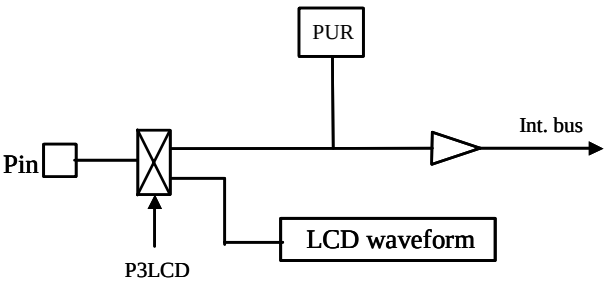
Port0.3~P0.5 structure



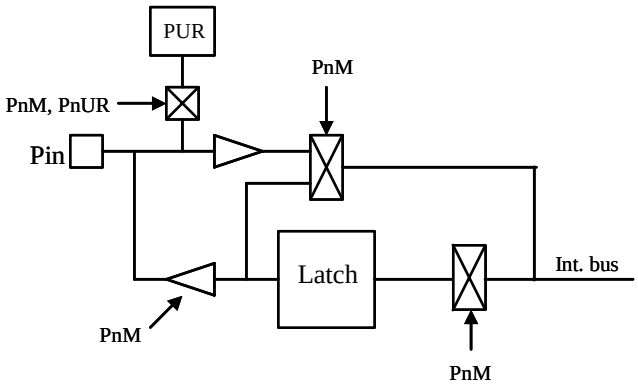
Port2 structure



Port3 structure



Port1,Port4,Port5 structure



Port6 structure

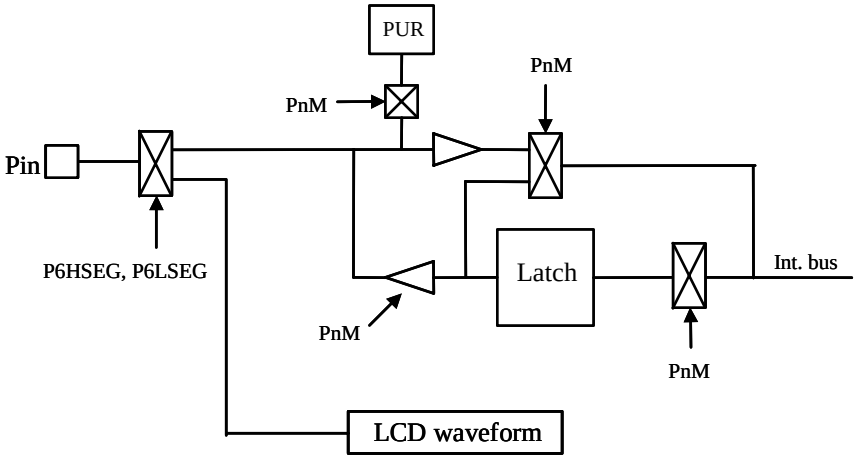


Figure 1-2. Pin Circuit Diagram

2 CODE OPTION TABLE

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
LCD	Enable	Enable LCD function
	Disable	Disable LCD function

Table 2-1. Code Option Table of SN8P1800

3 ADDRESS SPACES

PROGRAM MEMORY (ROM)

OVERVIEW

ROM Maps for SN8P1800 devices provide 4K x 16 OTP memory that programmable by user. The SN8P1800 program memory is able to fetch instructions through 12-bit wide PC (Program Counter) and can look up ROM data by using ROM code registers (R, X, Y, Z). In standard configuration, the device's 4,096 x 16-bit program memory has four areas:

- 1-word reset vector addresses
- 1-word Interrupt vector addresses
- 5-words reserved area
- 4K words general purpose area

All of the program memory is partitioned into two coding areas, located from 0000H to 0008H and from 0009H to 0FFEH. The former area is assigned for executing reset vector and interrupt vector. The later area is for storing instruction's OP-code and lookup table's data. User's program is in the last area (0010H~0FFEH).

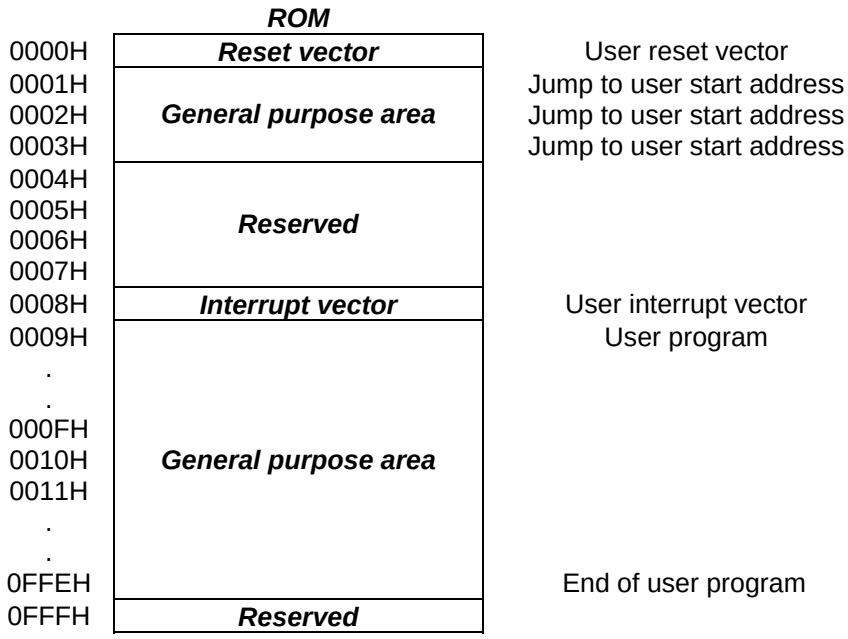


Figure 3-1. ROM Address Structure

USER RESET VECTOR ADDRESS (0000H)

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

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

```
CHIP SN8P1808
                ORG     0           ; 0000H
                JMP     START       ; Jump to user program address.
                .                   ; 0001H ~ 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 is executed, the program counter (PC) value is stored in stack buffer and points to 0008h of program memory to execute the vectored interrupt. Users have to define the interrupt vector. 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 SN8P1808
                ORG     0           ; 0000H
                JMP     START       ; Jump to user program address.
                .                   ; 0001H ~ 0007H are reserved

                ORG     8           ; Interrupt service routine
                B0XCH   A, ACCBUF   ; B0XCH doesn't change C, Z flag
                PUSH                ; Push 80H ~ 87H system registers
                .
                .
                .
                POP                 ; Pop 80H ~ 87H system registers
                B0XCH   A, ACCBUF
                RETI                ; End of interrupt service routine

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


                ENDP                ; End of program
```

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

```
CHIP SN8P1808
                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
                PUSH                ; Push 80H ~ 87H system registers
                .
                .
                .
                POP                 ; Pop 80H ~ 87H system registers
                B0XCH   A, ACCBUF
                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 4089-word at ROM locations 0010H~0FFEH are used as general-purpose memory. The area is stored instruction's op-code and look-up table data. The SN8P1800 includes jump table function by using program counter (PC) and look-up table function by using ROM code registers (R, X, Y, Z).

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

LOOKUP TABLE DESCRIPTION

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

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

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

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

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

➲ **Example: INC_YZ Macro**

```
INC_YZ          MACRO
                INCMS   Z               ; Z+1
                JMP     @F              ; Not overflow

                INCMS   Y               ; Y+1
                NOP                     ; Not overflow
@@:
                ENDM
```

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

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

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

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

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

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

JUMP TABLE DESCRIPTION

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

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

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

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

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

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

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

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

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

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

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

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

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

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

DATA MEMORY (RAM)

OVERVIEW

The SN8P1808 has internally built-in the huge data memory up to 256 bytes for storing general purpose data and featured with LCD memory space up to 24 locations (24 * 3 bits) for displaying pattern.

- 256 * 8-bit general purpose area
- 128 * 8-bit system register area
- 24 * 3-bit LCD memory space

These memory are separated into bank 0, bank1 and bank 15. The user can program RBANK register of RAM bank selection bit to access all data in any of the three RAM banks. The bank 0 and bank1, using the first 128-byte location assigned as general-purpose area, and the remaining 128-byte in bank 0 as system register. The bank 15 is LCD RAM area designed for storing LCD display data.

		RAM location	
BANK 0	000h	General purpose area	; 000h~07Fh of Bank 0 = To store general
		.	; purpose data (128 bytes).
	07Fh	.	
	080h	System register	; 080h~0FFh of Bank 0 = To store system
		.	; registers (128 bytes).
	0FFh	End of bank 0 area	
BANK 1	100h	General purpose area	; Bank 1 = To store general purpose data.
		.	
	17Fh	End of bank 1 area	; Bank 1 only has 128 bytes RAMs.
	200h	“	; reserved
		“	; “
	280h	“	; “
		“	; “
	300h	“	; “
		“	; “
	380h	“	; “
		“	; “
BANK 15	F00h	LCD RAM area	; Bank 15 = To store LCD display data
		.	; (24 bytes).
	F17h	End of LCD Ram	;

Figure 3-2 RAM Location of SN8P1808

- ***Note:1. The undefined locations of system register area are logic “high” after executing read instruction “MOV A, M”.***
- ***Note:2. The lower 24 locations of bank15 are used to store LCD display data and the other locations are reserved. The RAMs of LCD data area only have lowest 3-bit to be used. The highest 5-bit are undefined.***

RAM BANK SELECTION

The RBANK is a 1-bit register located at 87H in RAM bank 0. The user can access RAM data by using this register pointing to working RAM bank for ACC to read/write RAM data.

RBANK initial value = xxxx 0000

087H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
RBANK	-	-	-	-	RBNKS3	RBNKS2	RBNKS1	RBNKS0
	-	-	-	-	R/W	R/W	R/W	R/W

RBNKn: RAM bank selecting control bit. 0 = bank 0, 1 = bank 1.

➲ **Example: RAM bank selecting.**

```
; BANK 0
                CLR     RBANK
                .
; BANK 1
                MOV     A, #1
                B0MOV   RBANK, A
                .
```

➢ ***Note: "B0MOV" instruction can access the RAM of bank 0 in other bank situation directly.***

➲ **Example: Access RAM bank 0 in RAM bank 1.**

```
; BANK 1
                B0BSET  RBNKS0      ; Get into RAM bank 1
                B0MOV   A, BUF0     ; Read BUF0 data. BUF0 is in RAM bank0.
                MOV     BUF1, A     ; Write BUF0 data to BUF1. BUF1 is in RAM bank1.
                .
                .       .
                MOV     A, BUF1     ; Read BUF1 data and store in ACC.
                B0MOV   BUF0, A     ; Write ACC data to BUF0.
```

Under bank 1 situation, using "B0MOV" instruction is an easy way to access RAM bank 0 data. User can make a habit to read/write system register (0087H~00FFH). Then user can access system registers without switching RAM bank.

➲ **Example: To Access the system registers in bank 1 situation.**

```
; BANK 1
                B0BSET  RBNKS0      ; Get into RAM bank 1
                .       .
                MOV     A, #0FFH    ; Set all pins of P1 to be logic high.
                B0MOV   P1, A
                .
                B0MOV   A, P0       ; Read P0 data and store into BUF1 of RAM bank 1.
                MOV     BUF1, A
```

WORKING REGISTERS

The locations 80H to 86H of RAM bank 0 in data memory stores the specially defined registers such as register H, L, R, X, Y, Z and PFLAG, respectively shown in the following table. These registers can use as the general purpose of working buffer and can also be used to access ROM's and RAM's data. For instance, all of the ROM's table can be looked-up with R, X, Y and Z registers. And the data of RAM memory can be indirectly accessed with H, L, Y and Z registers.

	80H	81H	82H	83H	84H	85H	86H	
RAM	L	H	R	Z	Y	X	PFLAG	
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	

H, L REGISTERS

The H and L are 8-bit register with two major functions. One is to use the registers as working register. The other is to use the registers as data pointer to access RAM's data. The @HL that is data point_0 index buffer located at address E6H in RAM bank_0. It employs H and L registers to addressing RAM location in order to read/write data through ACC. The Lower 4-bit of H register is pointed to RAM bank number and L register is pointed to RAM address number, respectively. The higher 4-bit data of H register is truncated in RAM indirectly access mode.

H initial value = 0000 0000

081H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
H	HBIT7	HBIT6	HBIT5	HBIT4	HBIT3	HBIT2	HBIT1	HBIT0
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

L initial value = 0000 0000

080H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
L	LBIT7	LBIT6	LBIT5	LBIT4	LBIT3	LBIT2	LBIT1	LBIT0
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

➲ **Example: If want to read a data from RAM address 20H of bank_0, it can use indirectly addressing mode to access data as following.**

```
B0MOV   H, #00H     ; To set RAM bank 0 for H register
B0MOV   L, #20H     ; To set location 20H for L register
B0MOV   A, @HL      ; To read a data into ACC
```

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

```
        CLR     H           ; H = 0, bank 0
        MOV     A, #07FH
        B0MOV   L, A        ; L = 7FH, the last address of the data memory area
CLR_HL_BUF:
        CLR     @HL         ; Clear @HL to be zero
        DECMS   L           ; L – 1, if L = 0, finish the routine
        JMP     CLR_HL_BUF  ; Not zero

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

Y, Z REGISTERS

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

Y initial value = 0000 0000

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

Z initial value = 0000 0000

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

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

➲ **Example: If want to read a data from RAM address 25H of bank 1, it can use indirectly addressing mode to access data as following.**

```
B0MOV   Y, #01H     ; To set RAM bank 1 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 1 using @YZ register.**

```
        MOV     A, #1
        B0MOV   Y, A            ; Y = 1, bank 1
        MOV     A, #07FH
        B0MOV   Z, A            ; Y = 7FH, the last address of the data memory area

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

        DECMS   Z               ; Y – 1, if Y= 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.***

X REGISTERS

The X register is the 8-bit buffer. There are two major functions of the register. First, X register can be used as working registers. Second, the X registers can be address ROM location in order to look-up ROM data. The SN8P1800's program counter only has 12-bit. In look-up table function, users can omit X register.

X initial value = 0000 0000

085H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
X	XBIT7	XBIT6	XBIT5	XBIT4	XBIT3	XBIT2	XBIT1	XBIT0
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

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

R REGISTERS

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

R initial value = 0000 0000

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

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

PROGRAM FLAG

The PFLAG includes 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.

PFLAG initial value = xxxx x000

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

CARRY FLAG

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

DECIMAL CARRY FLAG

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

ZERO FLAG

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

ACCUMULATOR

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

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

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

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

            MOV         BUF, A
            .           .

; Write a immediate data into ACC

            MOV         A, #0FH
            .           .

; Write ACC data from BUF data memory

            MOV         A, BUF
            .           .
```

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

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

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

INT_SERVICE:
            B0XCH   A, ACCBUF   ; Store ACC value

            PUSH.   .           ; Push instruction
            .       .
            .
            .
            POP                 ; Pop instruction

            B0XCH   A, ACCBUF   ; Re-load ACC

            RETI                ; Exit interrupt service vector
```

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

STACK OPERATIONS

OVERVIEW

The stack buffer of SN8P1800 has 8-level high area and each level is 12-bits length. This buffer is designed to save and restore program counter's (PC) data when interrupt service is executed. The STKP register is a pointer designed to point active level in order to save or restore data from stack buffer for kernel circuit. The STKnH and STKnL are the 12-bit stack buffers to store program counter (PC) data.

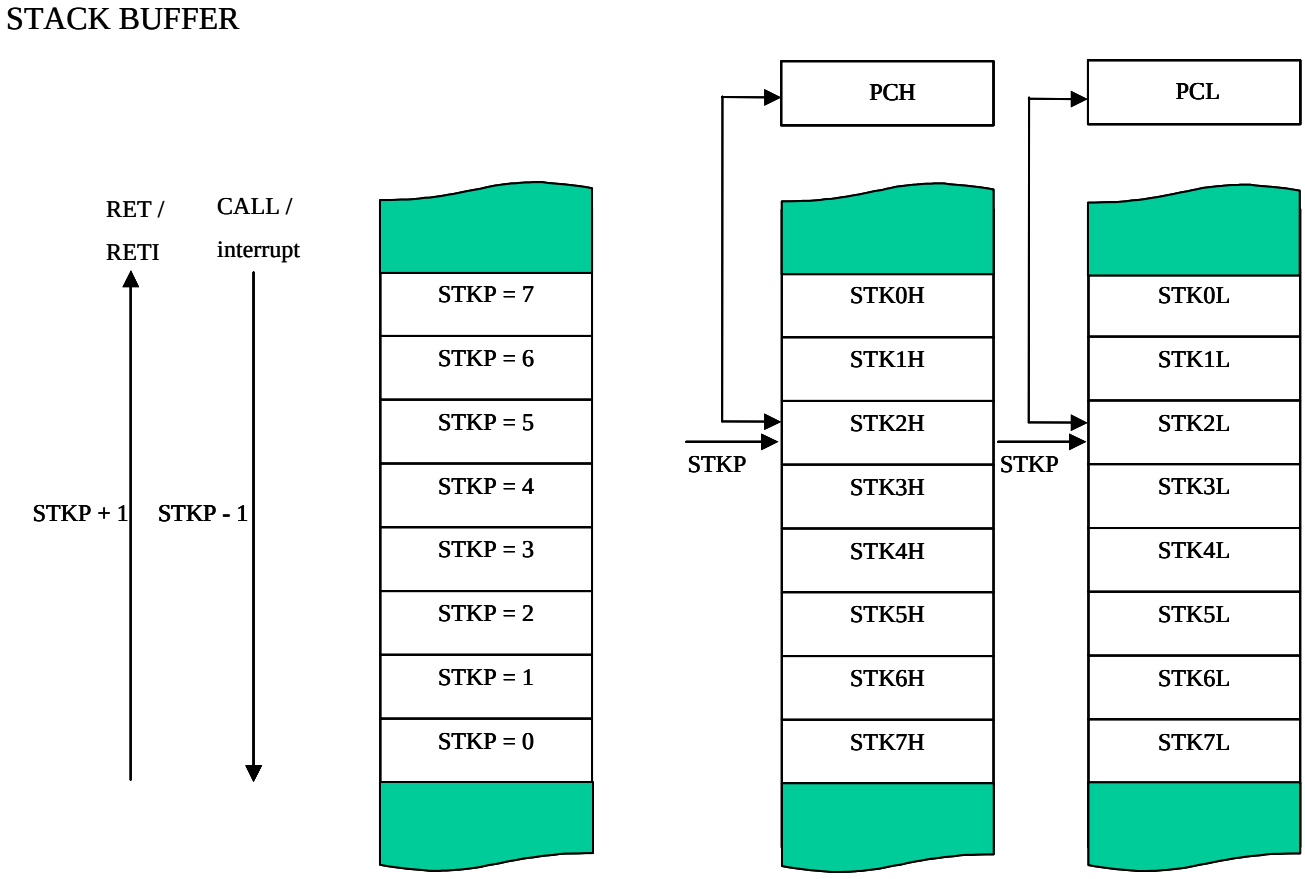


Figure 3-3 Stack-Save and Stack-Restore Operation

STACK REGISTERS

The stack pointer (STKP) is a 4-bit register to store the address used to access the stack buffer, 12-bits data memory (STKnH and STKnL) set aside for temporary storage of stack addresses.

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

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

STKP (stack pointer) initial value = 0xxx 1111

0DFH	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
STKP	GIE	-	-	-	STKPB3	STKPB2	STKPB1	STKPB0
	R/W	-	-	-	R/W	R/W	R/W	R/W

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

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

```
MOV     A, #00001111B
B0MOV   STKP, A
```

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

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

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

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

STACK OPERATION EXAMPLE

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

Stack Level	STKP Register				Stack Buffer		Description
	STKPB3	STKPB2	STKPB1	STKPB0	High Byte	Low Byte	
0	1	1	1	1	STK0H	STK0L	-
1	1	1	1	0	STK1H	STK1L	-
2	1	1	0	1	STK2H	STK2L	-
3	1	1	0	0	STK3H	STK3L	-
4	1	0	1	1	STK4H	STK4L	-
5	1	0	1	0	STK5H	STK5L	-
6	1	0	0	1	STK6H	STK6L	-
7	1	0	0	0	STK7H	STK7L	-
>8	-	-	-	-	-	-	**Stack Overflow**

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

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

Stack Level	STKP Register				Stack Buffer		Description
	STKPB3	STKPB2	STKPB1	STKPB0	High Byte	Low Byte	
7	1	0	0	0	STK7H	STK7L	-
6	1	0	0	1	STK6H	STK6L	-
5	1	0	1	0	STK5H	STK5L	-
4	1	0	1	1	STK4H	STK4L	-
3	1	1	0	0	STK3H	STK3L	-
2	1	1	0	1	STK2H	STK2L	-
1	1	1	1	0	STK1H	STK1L	-
0	1	1	1	1	STK0H	STK0L	-

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

PROGRAM COUNTER

The program counter (PC) is a 12-bit binary counter separated into the high-byte 4 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 11.

PC Initial value = xxxx 0000 0000 0000

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

PCH Initial value = xxxx 0000

0CFH	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
PCH	-	-	-	-	PC11	PC10	PC9	PC8
	-	-	-	-	R/W	R/W	R/W	R/W

PCL Initial value = 0000 0000

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

ONE ADDRESS SKIPPING

There are 9 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          ; Skip next instruction, if Carry_flag = 1
            JMP       C0STEP      ; Else jump to C0STEP.
            .
C0STEP:     NOP

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

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

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

If the result after increasing 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, A instruction (M = PCL) to activate multi-address jumping function. If carry signal occurs after execution of ADD PCL, A, the carry signal will not affect PCH register.

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

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

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

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

4 ADDRESSING MODE

OVERVIEW

The SN8P1800 provides three addressing modes to access RAM data, including immediate addressing mode, directly addressing mode and indirectly address mode. The main purpose of the three different modes is described in the following:

IMMEDIATE ADDRESSING MODE

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

Immediate addressing mode

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

DIRECTLY ADDRESSING MODE

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

Directly addressing mode

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

INDIRECTLY ADDRESSING MODE

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

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

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

MOV     A, #01H
B0MOV   Y, A            ; To set Y = 1 for accessing RAM bank 1.
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.

MOV     A, #0FH
B0MOV   Y, A            ; To set Y = 15 for accessing RAM bank 15.
B0MOV   Z, #12H         ; To set an immediate data 12H into Z register.
B0MOV   A, @YZ          ; Use data pointer @YZ reads a data from RAM location 012H
                        ; Into ACC.
```

TO ACCESS DATA in RAM BANK 0

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

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

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

➲ **Example 2: To use directly addressing mode (Through RBANK register).**

```
B0MOV   RBANK, #00H     ; To set RAM bank = 0
MOV     A, 12H          ; To move content from location 12H of RAM bank 0 to ACC
```

➲ **Example 3: To use indirectly addressing mode with @YZ register.**

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

TO ACCESS DATA in RAM BANK 1

In the RAM bank 1, this area memory can be read/written by these two access methods.

➲ **Example 1: To use directly addressing mode (Through RBANK register).**

```
B0MOV   RBANK, #01H     ; To set RAM bank = 1
MOV     A, 12H          ; To move content from location 12H of RAM bank 1 to ACC
```

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

```
MOV     A, #01H
B0MOV   Y, A            ; To set Y = 1 for accessing RAM bank 1.
B0MOV   Z, #12H         ; To set an immediate data 12H into Z register.
B0MOV   A, @YZ          ; Use data pointer @YZ reads a data from RAM location
                        ; 012H into ACC.
```

TO ACCESS DATA in RAM BANK 15 (LCD RAM)

In the RAM bank 15, this area memory can be read/written by these two access methods.

Example 1: To use directly addressing mode (Through RBANK register).

```
B0MOV   RBANK,#0FH      ; To set RAM bank = 15
MOV     A,12H           ; To move content from location 12H of RAM bank 15 to ACC
```

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

```
MOV     A,#0FH
B0MOV   Y,A             ; To set Y = 15 for accessing RAM bank 15.
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, SIO, ADC, PWM, LCD, timers and counters by programming. The Memory map provides an easy and quick reference source for writing application program. To accessing these system registers is controlled by the select memory bank (RBANK = 0) or the bank 0 read/write instruction (B0MOV, B0BSET, B0BCLR…).

SYSTEM REGISTER ARRANGEMENT (BANK 0)

BYTES of SYSTEM REGISTER

SN8P1808

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
8	L	H	R	Z	Y	X	PFLAG	RBANK	OPTION	-	-	-	-	-	-	-
9	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
A	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
B	-	ADM	ADB	ADR	SIOM	SIOR	SIOB	-	-	-	-	-	-	-	-	-
C	P1W	P1M	-	-	P4M	P5M	P6M	-	INTRQ	INTEN	OSCM	LCDM	-	TC0R	PCL	PCH
D	P0	P1	P2	P3	P4	P5	P6	-	T0M	T0C	TC0M	TC0C	TC1M	TC1C	TC1R	STKP
E	P0UR	P1UR	-	-	P4UR	P5UR	@HL	@YZ	-	-	-	-	-	-	-	-
F	STK7	STK7	STK6	STK6	STK5	STK5	STK4	STK4	STK3	STK3	STK2	STK2	STK1	STK1	STK0	STK0

Table 5-1. RAM Register Arrangement of SN8P1808

Description

L, H = Working & @HL addressing register
R = Working register and ROM lookup data buffer
X = Working and ROM address register
Y, Z = Working, @YZ and ROM addressing register
PFLAG = ROM page and special flag register
RBANK= RAM Bank Select register
OPTION= P3LCD and RCLK options.
ADM = ADC's mode register
ADR = ADC's resolution selects register
ADB = ADC's data buffer
SIOM = SIO mode control register
SIOR = SIO's clock reload buffer
SIOB = SIO's data buffer
P1W = Port 1 wakeup register
PnM = Port n input/output mode register
PnUR = Port n pull-up register
Pn = Port n data buffer
INTRQ = Interrupts' request register
INTEN = Interrupts' enable register
OSCM = Oscillator mode register
LCDM= LCD mode register
PCH, PCL = Program counter
T0M = Timer 0 mode register
TC0M = Timer/Counter 0 mode register
T0C = Timer 0 counting register
TC0C = Timer/Counter 0 counting register
TC1M = Timer/Counter 1 mode register
TC0R = Timer/Counter 0 auto-reload data buffer
TC1C = Timer/Counter 1 counting register
TC1R = Timer/Counter 1 auto-reload data buffer
STKP = Stack pointer buffer
STK0~STK7 = Stack 0 ~ stack 7 buffer
@HL = RAM HL indirect addressing index pointer
@YZ = RAM YZ indirect addressing index pointer

BITS of SYSTEM REGISTER

Address	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	R/W	Remarks
080H	LBIT7	LBIT6	LBIT5	LBIT4	LBIT3	LBIT2	LBIT1	LBIT0	R/W	L
081H	HBIT7	HBIT6	HBIT5	HBIT4	HBIT3	HBIT2	HBIT1	HBIT0	R/W	H
082H	RBIT7	RBIT6	RBIT5	RBIT4	RBIT3	RBIT2	RBIT1	RBIT0	R/W	R
083H	ZBIT7	ZBIT6	ZBIT5	ZBIT4	ZBIT3	ZBIT2	ZBIT1	ZBIT0	R/W	Z
084H	YBIT7	YBIT6	YBIT5	YBIT4	YBIT3	YBIT2	YBIT1	YBIT0	R/W	Y
085H	XBIT7	XBIT6	XBIT5	XBIT4	XBIT3	XBIT2	XBIT1	XBIT0	R/W	X
086H	-	-	-	-	-	C	DC	Z	R/W	PFLAG
087H	-	-	-	-	RBNKS3	RBNKS2	RBNKS1	RBNKS0	R/W	RBANK
088H	-	-	-	-	-	-	P3LCD	RCLK	R/W	OPTION
0B1H	ADENB	ADS	EOC	GCHS	-	CHS2	CHS1	CHS0	R/W	ADM mode register
0B2H	ADB11	ADB10	ADB9	ADB8	ADB7	ADB6	ADB5	ADB4	R	ADB data buffer
0B3H	-	ADCKS1	ADLEN	ADCKS0	ADB3	ADB2	ADB1	ADB0	R/W	ADR register
0B4H	SENB	START	SRATE1	SRATE0	0	SCKMD	SEDGE	TXRX	R/W	SIOM mode register
0B5H	SIOR7	SIOR6	SIOR5	SIOR4	SIOR3	SIOR2	SIOR1	SIOR0	W	SIOR reload buffer
0B6H	SIOB7	SIOB6	SIOB5	SIOB4	SIOB3	SIOB2	SIOB1	SIOB0	R/W	SIOB data buffer
0C0H	0	0	0	0	P13W	P12W	P11W	P10W	W	P1W wakeup register
0C1H	0	0	0	0	P13M	P12M	P11M	P10M	R/W	P1M I/O direction
0C4H	P47M	P46M	P45M	P44M	P43M	P42M	P41M	P40M	R/W	P4M I/O direction
0C5H	0	0	0	P54M	P53M	P52M	P51M	P50M	R/W	P5M I/O direction
0C6H	P67M	P66M	P65M	P64M	P63M	P62M	P61M	P60M	R/W	P6M I/O direction
0C8H	0	TC1IRQ	TC0IRQ	T0IRQ	SIOIRQ	P02IRQ	P01IRQ	P00IRQ	R/W	INTRQ
0C9H	0	TC1IEN	TC0IEN	T0IEN	SIOEN	P02IEN	P01IEN	P00IEN	R/W	INTEN
0CAH	WTCKS	WDRST	Wdrate	CPUM1	CPUM0	CLKMD	STPHX	0	R/W	OSCM
0CBH	-	-	BLANK	-	LENB	-	P6HSEG	P6LSEG	R/W	LCDM
0CDH	TC0R7	TC0R6	TC0R5	TC0R4	TC0R3	TC0R2	TC0R1	TC0R0	W	TC0R
0CEH	PC7	PC6	PC5	PC4	PC3	PC2	PC1	PC0	R/W	PCL
0CFH	-	-	-	-	PC11	PC10	PC9	PC8	R/W	PCH
0D0H	-	-	P05	P04	P03	P02	P01	P00	R	P0 data buffer
0D1H	-	-	-	-	P13	P12	P11	P10	R/W	P1 data buffer
0D2H	P27	P26	P25	P24	P23	P22	P21	P20	R/W	P2 data buffer
0D3H	P37	P36	P35	P34	P33	P32	P31	P30	R	P3 data buffer
0D4H	P47	P46	P45	P44	P43	P42	P41	P40	R/W	P4 data buffer
0D5H	-	-	-	P54	P53	P52	P51	P50	R/W	P5 data buffer
0D6H	P67	P66	P65	P64	P63	P62	P61	P60	R/W	P6 data buffer
0D8H	T0ENB	T0rate2	T0rate1	T0rate0	-	-	-	T0TB	R/W	T0M
0D9H	T0C7	T0C6	T0C5	T0C4	T0C3	T0C2	T0C1	T0C0	R/W	T0C
0DAH	TC0ENB	TC0rate2	TC0rate1	TC0rate0	TC0CKS	Aload0	TC0out	PWM0	R/W	TC0M
0DBH	TC0C7	TC0C6	TC0C5	TC0C4	TC0C3	TC0C2	TC0C1	TC0C0	R/W	TC0C
0DCH	TC1ENB	TC1rate2	TC1rate1	TC1rate0	TC1CKS	Aload1	TC1OUT	PWM1	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	-	-	-	STKPB3	STKPB2	STKPB1	STKPB0	R/W	STKP stack pointer
0E0H	-	-	P05R	P04R	P03R	P02R	P01R	P00R	W	P0UR
0E1H	-	-	-	-	P13R	P12R	P11R	P10R	W	P1UR
0E4H	P47R	P46R	P45R	P44R	P43R	P42R	P41R	P40R	W	P4UR
0E5H	-	-	-	P54R	P53R	P52R	P51R	P50R	W	P5UR
0E6H	@HL7	@HL6	@HL5	@HL4	@HL3	@HL2	@HL1	@HL0	R/W	@HL index pointer
0E7H	@YZ7	@YZ6	@YZ5	@YZ4	@YZ3	@YZ2	@YZ1	@YZ0	R/W	@YZ index pointer

(To be continued)

Address	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	R/W	Remarks
0F0H	S7PC7	S7PC6	S7PC5	S7PC4	S7PC3	S7PC2	S7PC1	S7PC0	R/W	STK7L
0F1H	-	-	-	-	S7PC11	S7PC10	S7PC9	S7PC8	R/W	STK7H
0F2H	S6PC7	S6PC6	S6PC5	S6PC4	S6PC3	S6PC2	S6PC1	S6PC0	R/W	STK6L
0F3H	-	-	-	-	S6PC11	S6PC10	S6PC9	S6PC8	R/W	STK6H
0F4H	S5PC7	S5PC6	S5PC5	S5PC4	S5PC3	S5PC2	S5PC1	S5PC0	R/W	STK5L
0F5H	-	-	-	-	S5PC11	S5PC10	S5PC9	S5PC8	R/W	STK5H
0F6H	S4PC7	S4PC6	S4PC5	S4PC4	S4PC3	S4PC2	S4PC1	S4PC0	R/W	STK4L
0F7H	-	-	-	-	S4PC11	S4PC10	S4PC9	S4PC8	R/W	STK4H
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

Table 5-2. Bit System Register Table of SN8P1808

- ***Note:***
  - ***a). All of register names had been declared in SN8ASM assembler.***
  - ***b). One-bit name had been declared in SN8ASM assembler with "F" prefix code.***
  - ***c). It will get logic "H" data, when use instruction to check empty location.***
  - ***d). The low nibble of ADR register is read only.***
  - ***e). "b0bset", "b0bclr", "bset", "bclr" of instructions just only support "R/W" registers.***
  - ***f). For detail description please refer file of "System Register Quick Reference Table"***

6 POWER ON RESET

OVERVIEW

SN8P1800 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 following.

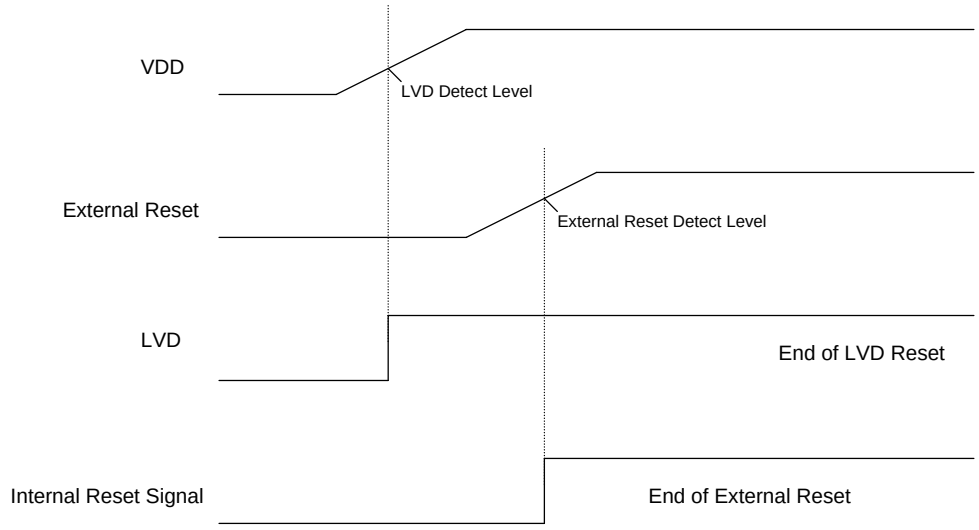


Figure 6-1 Power on Reset Timing Diagram

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.

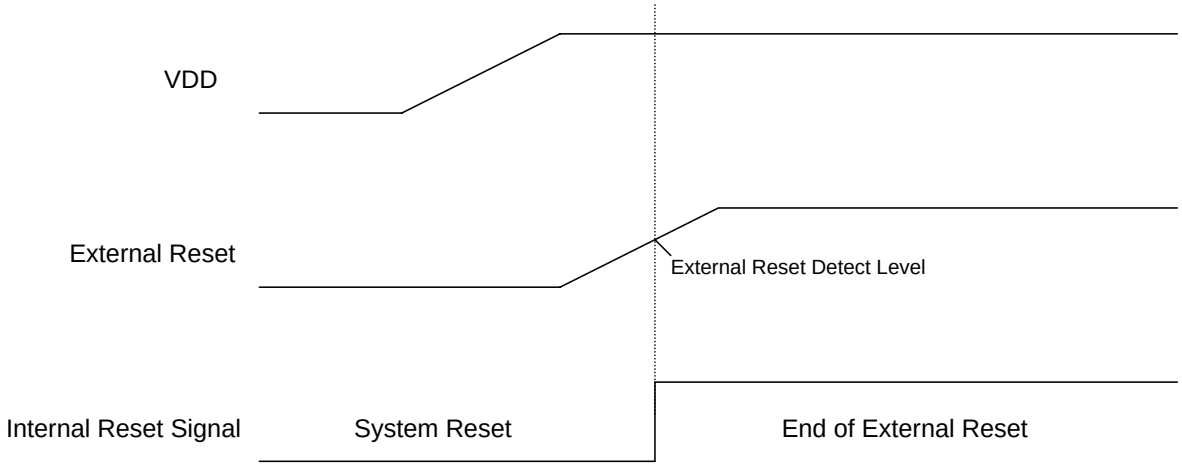


Figure 6-2 External Reset Timing Diagram

Users must to be sure the VDD stable earlier than external reset (Figure 5-2) or the external reset will fail. The external reset circuit is a simple RC circuit as following.

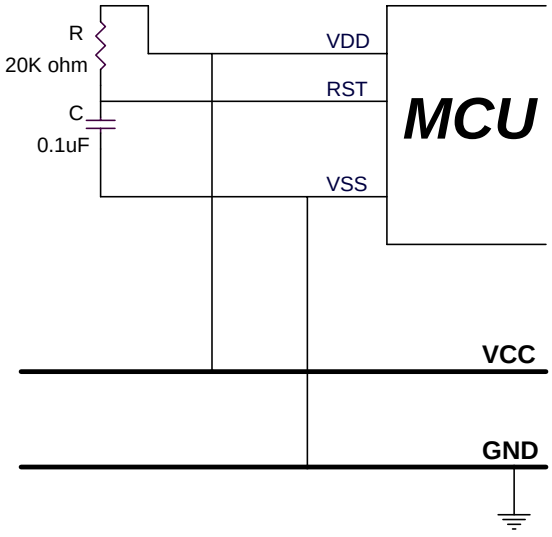


Figure 6-3. External Reset Circuit

In worse-power 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. The circuit can force the capacitor to release electric charge and drop the voltage, and solve the error.

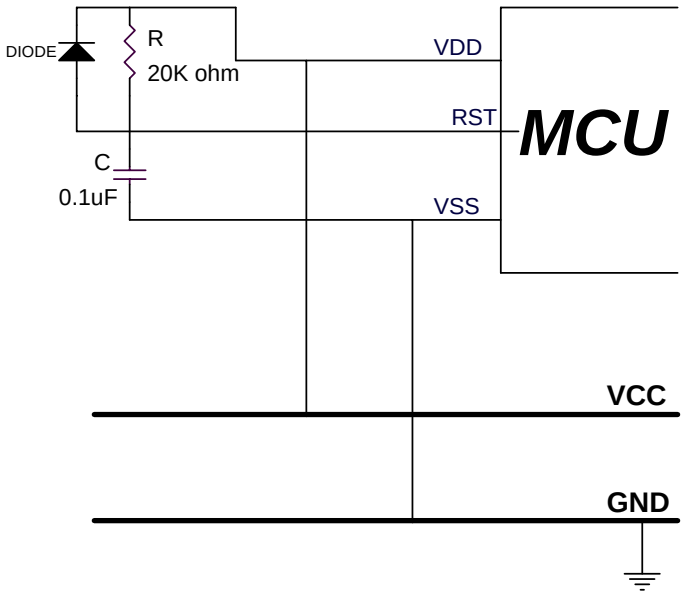


Figure 6-4. External Reset Circuit with Diode

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 worse power condition. LVD work with external reset function. They are OR active.

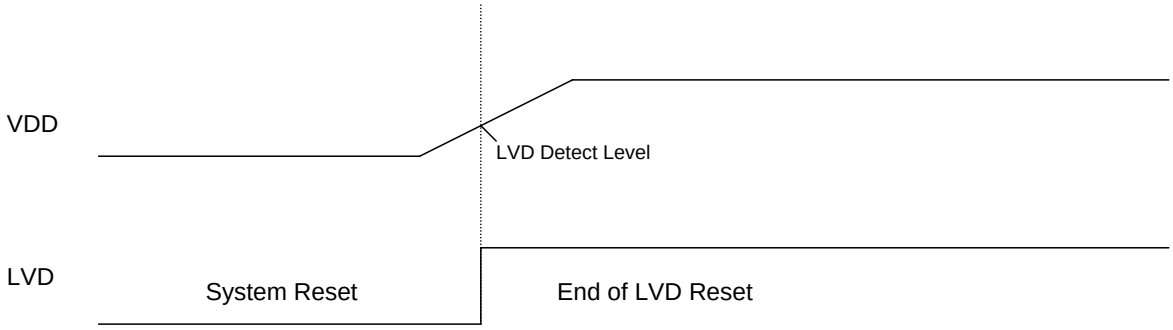


Figure 6-5. LVD Timing Diagram

The LVD can protect system to work well under brownout 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. So the LVD supports AC system well.

➢ ***Notice: LVD is selected by code option.***

7 OSCILLATORS

OVERVIEW

The SN8P1800 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 external low clock oscillator (32.768K) by crystal or RC mode. Because **Real-Time-Clock (RTC) used low-speed clock for timer, 32768Hz X'tal usually used for low-speed clock to an exact Real-Time-Clock**.

The external high-speed clock and the external low-speed clock can be system clock (Fosc). And 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:

- ✓ ***Basic timer (T0)***
- ✓ ***Timer counter 0 (TC0)***
- ✓ ***Timer counter 1 (TC1)***
- ✓ ***Watchdog timer***
- ✓ ***Serial I/O interface (SIO)***
- ✓ ***AD converter***
- ✓ ***PWM output (PWM0, PWM1)***
- ✓ ***Buzzer output (TC0OUT, TC1OUT)***

CLOCK BLOCK DIAGRAM

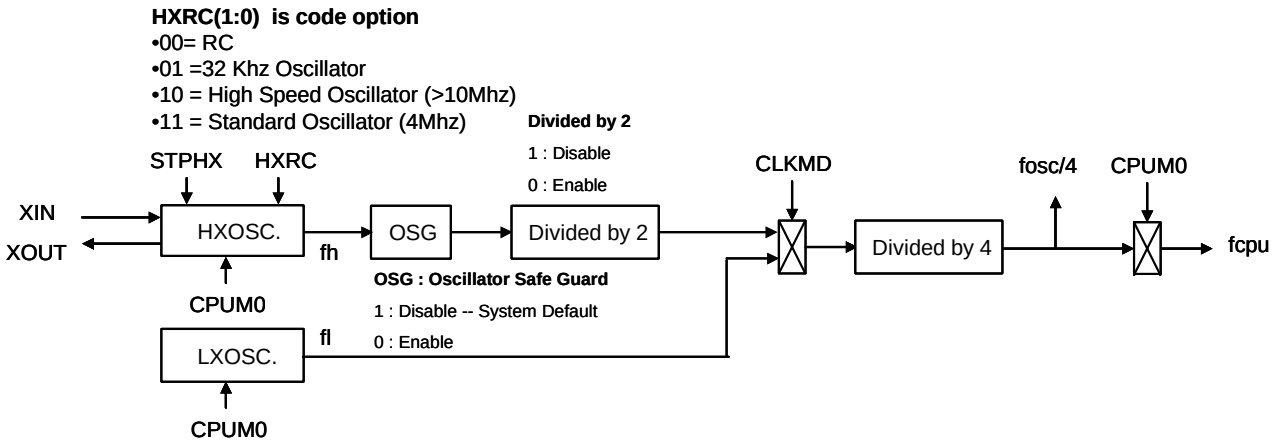


Figure 7-1. Clock Block Diagram

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

OSCM REGISTER DESCRIPTION

The OSCM register is a oscillator control register. It can control oscillator select, system mode, watchdog timer clock source and rate.

OSCM initial value = 0000 000x

0CAH	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
OSCM	WTCKS	WDRST	Wdrate	CPUM1	CPUM0	CLKMD	STPHX	-
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	-

STPHX: Eternal high-speed oscillator control bit. 0 = free run, 1 = stop. This bit just only controls external high-speed oscillator. If STPHX=1, the external low-speed RC oscillator is still running.
CLKMD: System high/Low speed mode select bit. 0 = normal (dual) mode, 1 = slow mode.
CPUM1,CPUM0: CPU operating mode control bit. 00=normal, 01= sleep (power down) mode to turn off both high/low clock, 10=green mode, 11=reserved
WTCKS: Watchdog clock source select bit. 0 = fcpu, 1 = internal RC low clock.

OPTION REGISTER DESCRIPTION

OPTION initial value = xxxx xx00

088H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
OPTION	-	-	-	-	-	-	P3LCD	RCLK
	-	-	-	-	-	-	R/W	R/W

RCLK: External low oscillator type control bit. 0 = Crystal mode , 1 = RC mode.
P3LCD: P3 I/O function control bit. 0 = LCD segment, 1 = input mode with pull-up resistor..

EXTERNAL HIGH-SPEED OSCILLATOR

SN8P1800 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 external low-speed
                    ; oscillator called power down mode (sleep mode).
```

OSCILLATOR MODE CODE OPTION

SN8P1800 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. High-speed crystal needs more current but the low one doesn't. 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. The table is as follow.

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

SN8P1800 has an external clock divide by 2 function. It is a code option 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

SN8P1800 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

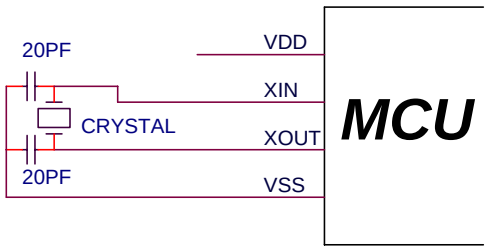


Figure 7-2. Crystal/Ceramic Oscillator

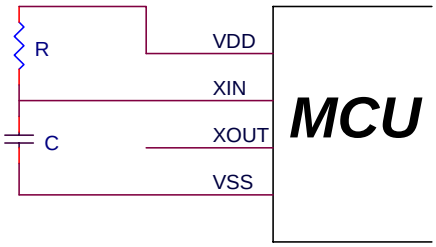


Figure 7-3. RC Oscillator

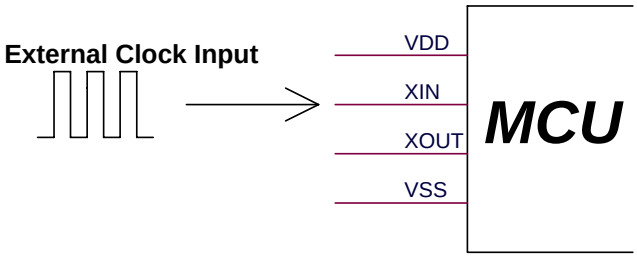


Figure 7-4. External clock input

- ***Note1: The VDD and VSS of external oscillator circuit must be from 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's frequency is divided by 2.***
- ***Note4: The power and ground of external oscillator circuit must be connected from the micro-controller's VDD and VSS. It is necessary to step up the performance of the whole system.***

External RC Oscillator Frequency Measurement

There 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
```

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)
- Green mode

In actual application, the user can adjust the chip's controller to work in these four modes by using OSCM register. At the high-speed mode, the instruction cycle (Fcpu) is Fosc/4. At the low-speed mode and 3V, 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, slow mode and green mode.

SLOW MODE

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

GREEN MODE

The green mode is a less power consumption mode. Under green mode, there are only T0 still counting and the other hardware stopping. The external high-speed oscillator or external low-speed oscillator is operating. To set CPUM1 = 1 and CPUM0 = 0, the system gets into green mode. The system can be waked up to last system mode by T0 timer timeout and P0, P1 trigger signal.

The green mode provides a time-variable wakeup function. Users can decide wakeup time by setting T0 timer. There are two channels into green mode. One is normal mode and the other is slow mode. In normal mode, the T0 timers overflow time is very short. In slow mode, the overflow time is longer. Users can select appropriate situation for their applications. Under green mode, the power consumption is 5u amp in 3V condition.

POWER DOWN MODE

The power down mode is also called sleep mode. The chip stops working as sleeping status. The power consumption is very less almost to zero. The power down mode is usually applied to low power consuming system as battery power productions. To set CUPM0 = 1, the system gets into power down mode. The external high-speed and low-speed oscillators are turned off. The system can be waked up by P0, P1 trigger signal.

SYSTEM MODE CONTROL

SN8P1800 SYSTEM MODE BLOCK DIAGRAM

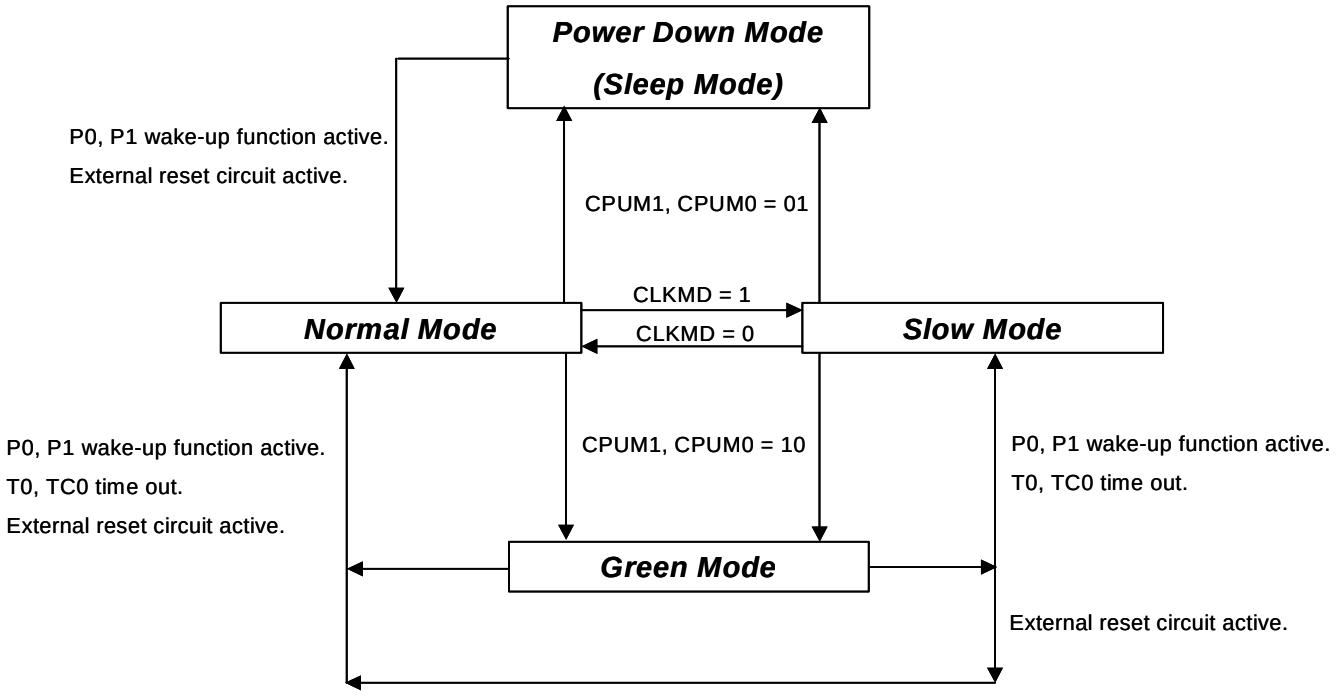


Figure 7-5. SN8P1800 System Mode Block Diagram

MODE	NORMAL	SLOW	Green	SLEEP	REMARK
HX osc.	Running	By STPHX	By STPHX	Stop	
LX osc.	Running	Running	Running	Stop	
CPU instruction	Executing	Executing	Stop	Stop	
T0 timer	*Active	*Active	*Active	Inactive	* Active by program
TC0 timer	*Active	*Active	*Active	Inactive	
TC1 timer	*Active	*Active	Inactive	Inactive	
Watchdog timer	Active	Active	Inactive	Inactive	
Internal interrupt	All active	All active	TC0	All inactive	
External interrupt	All active	All active	All Active	All inactive	
Wakeup source	-	-	Port0, Port1, T0, Reset	P0, P1, Reset	

Table 7-1. Oscillator Operating Mode Description

➢ ***Note: In the green mode, T0 trigger signals can switch CPU return to the last mode. If the system was into green mode from normal mode, the system returns to normal mode. If the system was into green mode from slow mode, the system returns to slow mode.***

SYSTEM MODE SWITCHING

Normal/Slow mode to power down (sleep) mode.
CPUM0 = 1)

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

➢ ***Note: In normal mode and slow mode, the CPUM1 = 0 and can omit to set CPUM1 = 0 routine.***

Normal mode to slow mode.

```
        B0BSET      FCLKMD          ;To set CLKMD = 1
        B0BSET      FSTPHX          ;To stop external high-speed oscillator.
```

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

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

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

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

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

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

Normal/Slow mode to green mode.
CPUM1, CPUM0 = 10
System can return to the last mode by P0, P1 and T0 wakeup function.

➲ **Example: Go into Green mode.**

```
        B0BSET      FCPUM1          ; To set CPUM1, CPUM0 = 10
```

➢ ***Note: In normal mode or slow mode, the CPUM0 = 0 and can omit to set CPUM0 = 0 routine.***

➲ **Example: Go into Green mode and enable T0 wakeup function.**

```
; Set T0 timer wakeup functon.
        B0BCLR      FT0IEN          ; To disable T0 interrupt service
        B0BCLR      FT0ENB          ; To disable T0 timer
        MOV         A,#20H          ;
        B0MOV       T0M,A           ; To set T0 clock = fcpu / 64
        MOV         A,#74H
        B0MOV       T0C,A           ; To set T0C initial value = 74H (To set T0 interval = 10 ms)
        B0BCLR      FT0IEN          ; To disable T0 interrupt service
        B0BCLR      FT0IRQ          ; To clear T0 interrupt request
        B0BSET      FT0ENB          ; To enable T0 timer
; Go into green mode
        B0BCLR      FCPUM0          ;To set CPUMx = 10
        B0BSET      FCPUM1
```

➢ ***Note: If T0ENB = 0, T0 is without wakeup from green mode to normal/slow mode function.***

WAKEUP TIME

OVERVIEW

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

There are two conditions need wakeup time. One is power down mode to normal mode. The other one is slow mode to normal mode. For the first case, SN8P1800 provides 2048 oscillator clocks to be the wakeup time. But in the last case, users need to make the wakeup time by themselves.

HARDWARE WAKEUP

When the system is in power down mode (sleep mode), the external high-speed oscillator stops. For wakeup into normal, SN8P1800 provides 2048 external high-speed oscillator clocks to be the wakeup time for warming up the oscillator circuit. After the wakeup time, the system goes into the normal mode. The value of the wakeup time is as following.

The wakeup time = 1/Fosc * 2048 (sec)

➲ **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 following.**

*The wakeup time = 1/Fosc * 2048 = 0.57 ms (Fosc = 3.58MHz)*

*The wakeup time = 1/Fosc * 2048 = 62.5 ms (Fosc=32768Hz)*

Under power down mode (sleep mode), there are only I/O ports with wakeup function making the system to return normal mode. The Port 0 and Port 1 have wakeup function. Port 0's wakeup function always enables. The Port 1 controls by the P1W register.

P1W initial value = xx00 0000

0C0H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
P1W	-	-	-	-	P13W	P12W	P11W	P10W
	-	-	-	-	W	W	W	W

P10W~P13W: Port 1 wakeup function control bits. 0 = none wakeup function, 1 = Enable each pin of Port 1 wakeup function.

8 TIMERS COUNTERS

WATCHDOG TIMER (WDT)

The watchdog timer (WDT) is a binary up counter designed for monitoring program execution. If the program get into the unknown status by noise interference, WDT's overflow signal will reset this chip and restart operation. The instruction that clear the watch-dog timer (B0BSET FWDRST) should be executed at proper points in a program within a given 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 with reset status. In order to generate different output timings, the user can control watchdog timer by modifying Wdrate control bits of OSCM register. The watchdog timer will be disabled at green and power down modes.

OSCM initial value = 0000 000x

0CAH	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
OSCM	WTCKS	WDRST	Wdrate	CPUM1	CPUM0	CLKMD	STPHX	0
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	-

Wdrate: Watchdog timer rate select bit. 0 =14^{th}, 1 = 8^{th}.
WDRST : Watch dog timer reset bit. 0 = Non reset, 1 = clear the watchdog timer's counter.
WTCKS: Watchdog clock source select bit. 0 = Fcpu, 1 = internal RC low clock.

WTCKS	WTRATE	CLKMD	Watchdog Timer Overflow Time
0	0	0	1 / (fcpu ÷ 2^{14} ÷ 16) = 293 ms, Fosc=3.58MHz
0	1	0	1 / (fcpu ÷ 2^{8} ÷ 16) = 500 ms, Fosc=32768Hz
0	0	1	1 / (fcpu ÷ 2^{14} ÷ 16) = 32s, Fosc=32768Hz
0	1	1	1 / (fcpu ÷ 2^{8} ÷ 16) = 500ms, Fosc=32768Hz
1	-	-	1 / (32768 ÷ 512 ÷ 16) ~ 0.25s

Figure 8-1. Watchdog timer overflow time table

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

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

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

BASIC TIMER 0 (T0)

OVERVIEW

The basic timer (T0) is an 8-bit binary up counter. It uses T0M register to select T0C's input clock for counting a precision time. If the T0 timer has occur an overflow (from FFH to 00H), it will continue counting and issue a time-out signal to trigger T0 interrupt to request interrupt service. The main purposes of the T0 basic timer is as following.

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

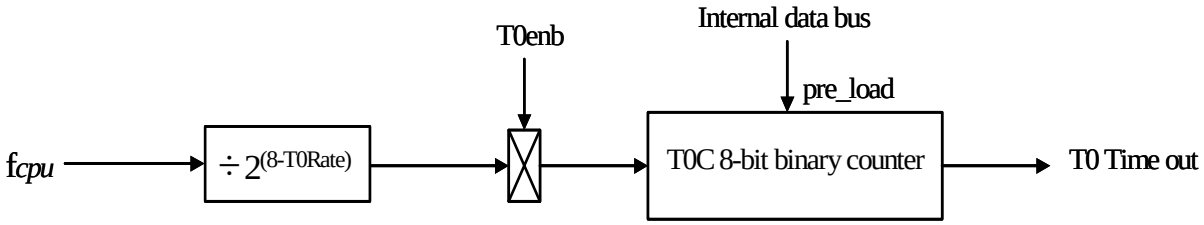


Figure 8-2. Basic Timer T0 Block Diagram

T0M REGISTER DESCRIPTION

The T0M is the basic timer mode register which is a 8-bit read/write register and only used the high nibble. By loading different value into the T0M register, users can modify the basic timer clock dynamically as program executing.

Eight rates for T0 timer can be selected by T0RATE0 ~ T0RATE2 bits. The range is from fcpu/2 to fcpu/256. The T0M initial value is zero and the rate is fcpu/256. The bit7 of T0M called T0ENB is the control bit to start T0 timer. The combination of these bits is to determine the T0 timer clock frequency and the intervals.

T0M initial value = 0000 xxxx

0D8H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
T0M	T0ENB	T0RATE2	T0RATE1	T0RATE0	0	0	0	T0TB
	R/W	R/W	R/W	R/W	-	-	-	R/W

T0ENB: T0 timer control bit. 0 = disable, 1 = enable.
T0RATE2~T0RATE0: The T0 timer's clock source select bits. 000 = fcpu/256, 001 = fcpu/128, … , 110 = fcpu/4, 111 = fcpu/2.
T0TB : Timer 0 as the Real-Time clock time base.
0 = Timer 0 function as a normal timer system.
1 = Timer 0 function as a Real-Time Clock. The clock source of timer 0 will be switched to external low clock (32.768K crystal oscillator).

- ***Note: 1. Register setting for Timer 0 as Real-Timer clock:***

Register	Bit	Logic	Description
OPTION	RCLK	0	External low oscillator type = Crystal mode
T0M	T0TB	1	Timer 0 function = Real-Time Clock

- ***Note:2. Interrupt/Wakeup period of Real-Time clock is 0.5 second in 32768hz X'tal. (32768hz / 64 / 256 = 2 hz).***

T0C COUNTING REGISTER

T0C is an 8-bit counter register for the basic timer (T0). T0C must be reset whenever the T0ENB is set "1" to start the basic timer. T0C is incremented by one with every clock pulse which frequency is determined by T0RATE0 ~ T0RATE2. When T0C has incremented to "0FFH", it will be cleared to "00H" in next clock and an overflow generated. Under T0 interrupt service request (T0IEN) enable condition, the T0 interrupt request flag will be set "1" and the system executes the interrupt service routine. The T0C has no auto reload function. After T0C overflow, the T0C is continuing counting. Users need to reset T0C value to get a accurate time.

T0C initial value = xxxx xxxx

0D9H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
T0C	T0C7	T0C6	T0C5	T0C4	T0C3	T0C2	T0C1	T0C0
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

T0RATE	T0CLOCK	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

Figure 8-3. The Timing Table of Basic Timer T0.

The equation of T0C initial value is as following.

T0C initial value = 256 - (T0 interrupt interval time * input clock)

➲ **Example : To set 10ms interval time for T0 interrupt at 3.58MHz high-speed mode. T0C value (74H) = 256 - (10ms * fcpu/64)**

$$\begin{aligned} \textit{T0C initial value} &= 256 - (\textit{T0 interrupt interval time} * \textit{input clock}) \\ &= 256 - (10ms * 3.58 * 10^6 / 4 / 64) \\ &= 256 - (10^{-2} * 3.58 * 10^6 / 4 / 64) \\ &= 116 \\ &= 74H \end{aligned}$$

T0 BASIC TIMER OPERATION SEQUENCE

The T0 basic timer's sequence of operation can be following.

- Set the T0C initial value to setup the interval time.
- Set the T0ENB to be "1" to enable T0 basic timer.
- T0C is incremented by one with each clock pulse which frequency is corresponding to T0M selection.
- T0C overflow when T0C from FFH to 00H.
- When T0C overflow occur, the T0IRQ flag is set to be "1" by hardware.
- Execute the interrupt service routine.
- Users reset the T0C value and resume the T0 timer operation.

➲ **Example: Setup the T0M and T0C.**

```
        B0BCLR      FT0IEN          ; To disable T0 interrupt service
        B0BCLR      FT0ENB          ; To disable T0 timer
        MOV         A,#20H          ;
        B0MOV       T0M,A           ; To set T0 clock = fcpu / 64
        MOV         A,#74H
        B0MOV       T0C,A           ; To set T0C initial value = 74H (To set T0 interval = 10 ms)
        B0BSET      FT0IEN          ; To enable T0 interrupt service
        B0BCLR      FT0IRQ          ; To clear T0 interrupt request
        B0BSET      FT0ENB          ; To enable T0 timer
```

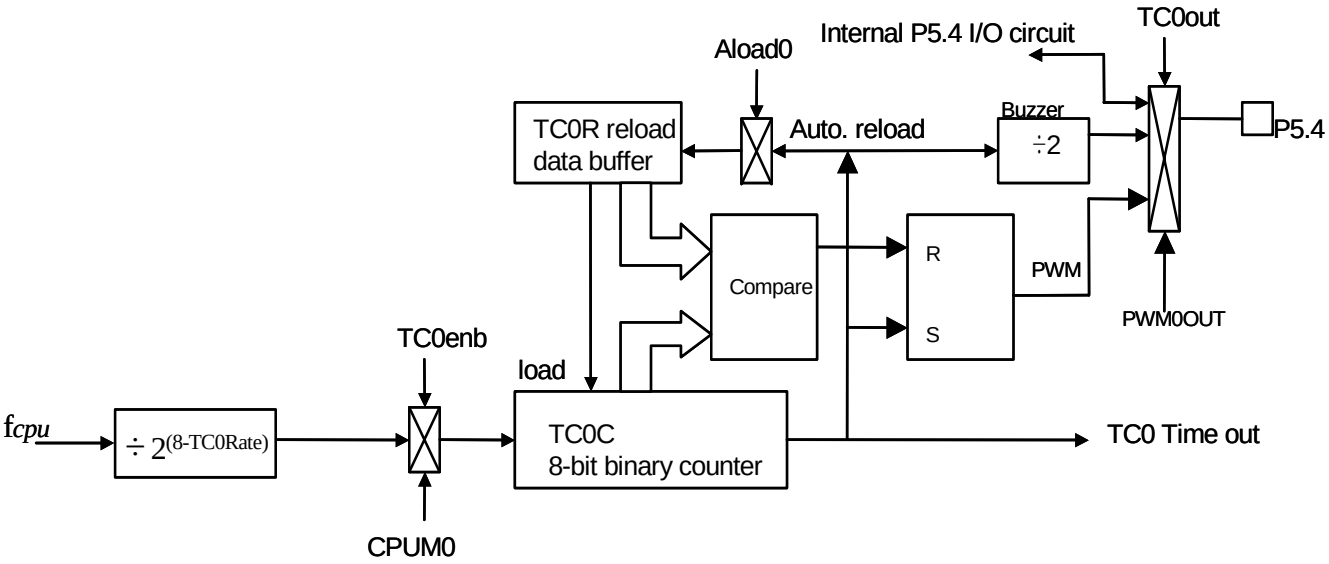
➲ **Example: T0 interrupt service routine.**

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

        B0XCH       A, ACCBUF       ; Store ACC value.
        PUSH                        ; Push

        B0BTS1      FT0IRQ          ; Check T0IRQ
        JMP         EXIT_INT        ; T0IRQ = 0, exit interrupt vector

        B0BCLR      FT0IRQ          ; Reset T0IRQ
        MOV         A,#74H          ; Reload T0C
        B0MOV       T0C,A
        .           .               ; T0 interrupt service routine
        .           .
        JMP         EXIT_INT        ; End of T0 interrupt service routine and exit interrupt vector
        .           .
        .           .
EXIT_INT:

        POP                         ; POP
        B0XCH       A, ACCBUF       ; Restore ACC value.

        RETI                        ; Exit interrupt vector
```

TIMER COUNTER 0 (TC0)

OVERVIEW

The timer counter 0 (TC0) is used to generate an interrupt request when a specified time interval has elapsed. TC0 has a auto re-loadable counter that consists of two parts: an 8-bit reload register (TC0R) into which you write the



counter reference value, and an 8-bit counter register (TC0C) whose value is automatically incremented by counter logic.

Figure 8-4. Timer Count TC0 Block Diagram

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

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

TC0M MODE REGISTER

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

Eight rates for TC0 timer can be selected by TC0RATE0 ~ TC0RATE2 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 called 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.

TC0M initial value = 0000 0000

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

TC0ENB: TC0 counter/BZ0/PWM0OUT 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.
TC0CKS: TC0 clock source select bit. “0” = Fcpu, “1” = External clock come from INT0/P0.0 pin. TC0 will be an event counter.
ALOAD0: TC0 auto-reload function control bit. 0 = none auto-reload, 1 = auto-reload.
TC0OUT: TC0 time-out toggle signal output control bit. 0 = To disable TC0 signal output and to enable P5.4’s I/O function, 1 = To enable TC0’s signal output and to disable P5.4’s I/O function. (Auto-disable the PWM0OUT function.)
PWM0OUT: TC0’s PWM output control bit. 0 = To disable the PWM output, 1 = To enable the PWM output (The TC0OUT control bit must = 0)

- ***Note1: When TC0CKS=1, TC0 became an external event counter. No more P0.0 interrupt request will be raised. (P0.0IRQ will be always 0)***

- ***Note2: The ICE S8KC do not support the PWM0OUT and TC0OUT Function. The PWM0OUT and TC0OUT must use the S8KD ICE (or later) to verify the function.***

TC0C COUNTING REGISTER

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

TC0C initial value = xxxx xxxx

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

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

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

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)**

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

TC0R AUTO-LOAD REGISTER

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

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

TC0R initial value = xxxx xxxx

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

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

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

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

TC0 TIMER COUNTER OPERATION SEQUENCE

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

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

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

```
B0BCLR      FTC0IEN     ; To disable TC0 interrupt service
B0BCLR      FTC0ENB     ; To disable TC0 timer
MOV         A,#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: Setup the TC0M and TC0C with auto-reload function.**

```
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)
B0MOV       TC0R,A      ; To set TC0R auto-reload register

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

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

```
                ORG         8               ; Interrupt vector
                JMP         INT_SERVICE
INT_SERVICE:
                B0XCH       A, ACCBUF       ; B0XCH doesn't change C, Z flag
                PUSH                        ; Push
                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:
                POP                         ; Pop
                B0XCH       A, ACCBUF       ; Restore ACC value.
                RETI                        ; Exit interrupt vector
```

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

```
                ORG         8               ; Interrupt vector
                JMP         INT_SERVICE
INT_SERVICE:
                B0XCH       A, ACCBUF       ; B0XCH doesn't change C, Z flag
                PUSH                        ; Push
                B0BTS1      FTC0IRQ         ; Check TC0IRQ
                JMP         EXIT_INT        ; TC0IRQ = 0, exit interrupt vector
                B0BCLR      FTC0IRQ         ; Reset TC0IRQ
                .           .               ; TC0 interrupt service routine
                .           .
                JMP         EXIT_INT        ; End of TC0 interrupt service routine and exit interrupt
                                            vector
                .           .
                .           .
EXIT_INT:
                POP                         ; Pop
                B0XCH       A, ACCBUF       ; Restore ACC value.
                RETI                        ; Exit interrupt vector
```

TC0 CLOCK FREQUENCY OUTPUT (BUZZER)

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

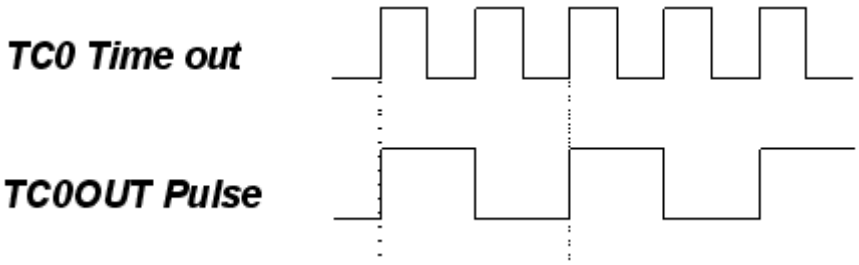


Figure 8-5. The TC0OUT Pulse Frequency

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

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

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

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

TC0OUT FREQUENCY TABLE

Fosc = 4MHz, TC0 Rate = Fcpu/8

TC0R	TC0OUT (KHz)	TC0R	TC0OUT (KHz)	TC0R	TC0OUT (KHz)	TC0R	TC0OUT (KHz)	TC0R	TC0OUT (KHz)
0	0.2441	56	0.3125	112	0.4340	168	0.7102	224	1.9531
1	0.2451	57	0.3141	113	0.4371	169	0.7184	225	2.0161
2	0.2461	58	0.3157	114	0.4401	170	0.7267	226	2.0833
3	0.2470	59	0.3173	115	0.4433	171	0.7353	227	2.1552
4	0.2480	60	0.3189	116	0.4464	172	0.7440	228	2.2321
5	0.2490	61	0.3205	117	0.4496	173	0.7530	229	2.3148
6	0.2500	62	0.3222	118	0.4529	174	0.7622	230	2.4038
7	0.2510	63	0.3238	119	0.4562	175	0.7716	231	2.5000
8	0.2520	64	0.3255	120	0.4596	176	0.7813	232	2.6042
9	0.2530	65	0.3272	121	0.4630	177	0.7911	233	2.7174
10	0.2541	66	0.3289	122	0.4664	178	0.8013	234	2.8409
11	0.2551	67	0.3307	123	0.4699	179	0.8117	235	2.9762
12	0.2561	68	0.3324	124	0.4735	180	0.8224	236	3.1250
13	0.2572	69	0.3342	125	0.4771	181	0.8333	237	3.2895
14	0.2583	70	0.3360	126	0.4808	182	0.8446	238	3.4722
15	0.2593	71	0.3378	127	0.4845	183	0.8562	239	3.6765
16	0.2604	72	0.3397	128	0.4883	184	0.8681	240	3.9063
17	0.2615	73	0.3415	129	0.4921	185	0.8803	241	4.1667
18	0.2626	74	0.3434	130	0.4960	186	0.8929	242	4.4643
19	0.2637	75	0.3453	131	0.5000	187	0.9058	243	4.8077
20	0.2648	76	0.3472	132	0.5040	188	0.9191	244	5.2083
21	0.2660	77	0.3492	133	0.5081	189	0.9328	245	5.6818
22	0.2671	78	0.3511	134	0.5123	190	0.9470	246	6.2500
23	0.2682	79	0.3531	135	0.5165	191	0.9615	247	6.9444
24	0.2694	80	0.3551	136	0.5208	192	0.9766	248	7.8125
25	0.2706	81	0.3571	137	0.5252	193	0.9921	249	8.9286
26	0.2717	82	0.3592	138	0.5297	194	1.0081	250	10.4167
27	0.2729	83	0.3613	139	0.5342	195	1.0246	251	12.5000
28	0.2741	84	0.3634	140	0.5388	196	1.0417	252	15.6250
29	0.2753	85	0.3655	141	0.5435	197	1.0593	253	20.8333
30	0.2765	86	0.3676	142	0.5482	198	1.0776	254	31.2500
31	0.2778	87	0.3698	143	0.5531	199	1.0965	255	62.5000
32	0.2790	88	0.3720	144	0.5580	200	1.1161		
33	0.2803	89	0.3743	145	0.5631	201	1.1364		
34	0.2815	90	0.3765	146	0.5682	202	1.1574		
35	0.2828	91	0.3788	147	0.5734	203	1.1792		
36	0.2841	92	0.3811	148	0.5787	204	1.2019		
37	0.2854	93	0.3834	149	0.5841	205	1.2255		
38	0.2867	94	0.3858	150	0.5896	206	1.2500		
39	0.2880	95	0.3882	151	0.5952	207	1.2755		
40	0.2894	96	0.3906	152	0.6010	208	1.3021		
41	0.2907	97	0.3931	153	0.6068	209	1.3298		
42	0.2921	98	0.3956	154	0.6127	210	1.3587		
43	0.2934	99	0.3981	155	0.6188	211	1.3889		
44	0.2948	100	0.4006	156	0.6250	212	1.4205		
45	0.2962	101	0.4032	157	0.6313	213	1.4535		
46	0.2976	102	0.4058	158	0.6378	214	1.4881		
47	0.2990	103	0.4085	159	0.6443	215	1.5244		
48	0.3005	104	0.4112	160	0.6510	216	1.5625		
49	0.3019	105	0.4139	161	0.6579	217	1.6026		
50	0.3034	106	0.4167	162	0.6649	218	1.6447		
51	0.3049	107	0.4195	163	0.6720	219	1.6892		
52	0.3064	108	0.4223	164	0.6793	220	1.7361		
53	0.3079	109	0.4252	165	0.6868	221	1.7857		
54	0.3094	110	0.4281	166	0.6944	222	1.8382		
55	0.3109	111	0.4310	167	0.7022	223	1.8939		

Table 8-2. TC0OUT Frequency Table for Fosc = 4MHz, TC0 Rate = Fcpu/8

Fosc = 16MHz, TC0 Rate = Fcpu/8

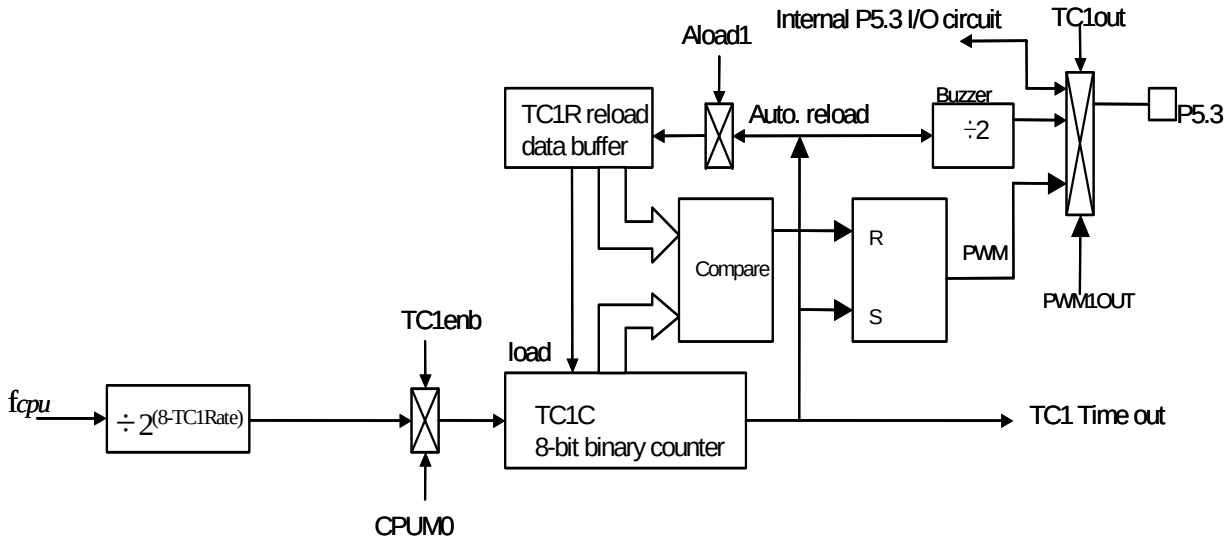
TC0R	TC0OUT (KHz)	TC0R	TC0OUT (KHz)	TC0R	TC0OUT (KHz)	TC0R	TC0OUT (KHz)	TC0R	TC0OUT (KHz)
0	0.9766	56	1.2500	112	1.7361	168	2.8409	224	7.8125
1	0.9804	57	1.2563	113	1.7483	169	2.8736	225	8.0645
2	0.9843	58	1.2626	114	1.7606	170	2.9070	226	8.3333
3	0.9881	59	1.2690	115	1.7730	171	2.9412	227	8.6207
4	0.9921	60	1.2755	116	1.7857	172	2.9762	228	8.9286
5	0.9960	61	1.2821	117	1.7986	173	3.0120	229	9.2593
6	1.0000	62	1.2887	118	1.8116	174	3.0488	230	9.6154
7	1.0040	63	1.2953	119	1.8248	175	3.0864	231	10.0000
8	1.0081	64	1.3021	120	1.8382	176	3.1250	232	10.4167
9	1.0121	65	1.3089	121	1.8519	177	3.1646	233	10.8696
10	1.0163	66	1.3158	122	1.8657	178	3.2051	234	11.3636
11	1.0204	67	1.3228	123	1.8797	179	3.2468	235	11.9048
12	1.0246	68	1.3298	124	1.8939	180	3.2895	236	12.5000
13	1.0288	69	1.3369	125	1.9084	181	3.3333	237	13.1579
14	1.0331	70	1.3441	126	1.9231	182	3.3784	238	13.8889
15	1.0373	71	1.3514	127	1.9380	183	3.4247	239	14.7059
16	1.0417	72	1.3587	128	1.9531	184	3.4722	240	15.6250
17	1.0460	73	1.3661	129	1.9685	185	3.5211	241	16.6667
18	1.0504	74	1.3736	130	1.9841	186	3.5714	242	17.8571
19	1.0549	75	1.3812	131	2.0000	187	3.6232	243	19.2308
20	1.0593	76	1.3889	132	2.0161	188	3.6765	244	20.8333
21	1.0638	77	1.3966	133	2.0325	189	3.7313	245	22.7273
22	1.0684	78	1.4045	134	2.0492	190	3.7879	246	25.0000
23	1.0730	79	1.4124	135	2.0661	191	3.8462	247	27.7778
24	1.0776	80	1.4205	136	2.0833	192	3.9063	248	31.2500
25	1.0823	81	1.4286	137	2.1008	193	3.9683	249	35.7143
26	1.0870	82	1.4368	138	2.1186	194	4.0323	250	41.6667
27	1.0917	83	1.4451	139	2.1368	195	4.0984	251	50.0000
28	1.0965	84	1.4535	140	2.1552	196	4.1667	252	62.5000
29	1.1013	85	1.4620	141	2.1739	197	4.2373	253	83.3333
30	1.1062	86	1.4706	142	2.1930	198	4.3103	254	125.0000
31	1.1111	87	1.4793	143	2.2124	199	4.3860	255	250.0000
32	1.1161	88	1.4881	144	2.2321	200	4.4643		
33	1.1211	89	1.4970	145	2.2523	201	4.5455		
34	1.1261	90	1.5060	146	2.2727	202	4.6296		
35	1.1312	91	1.5152	147	2.2936	203	4.7170		
36	1.1364	92	1.5244	148	2.3148	204	4.8077		
37	1.1416	93	1.5337	149	2.3364	205	4.9020		
38	1.1468	94	1.5432	150	2.3585	206	5.0000		
39	1.1521	95	1.5528	151	2.3810	207	5.1020		
40	1.1574	96	1.5625	152	2.4038	208	5.2083		
41	1.1628	97	1.5723	153	2.4272	209	5.3191		
42	1.1682	98	1.5823	154	2.4510	210	5.4348		
43	1.1737	99	1.5924	155	2.4752	211	5.5556		
44	1.1792	100	1.6026	156	2.5000	212	5.6818		
45	1.1848	101	1.6129	157	2.5253	213	5.8140		
46	1.1905	102	1.6234	158	2.5510	214	5.9524		
47	1.1962	103	1.6340	159	2.5773	215	6.0976		
48	1.2019	104	1.6447	160	2.6042	216	6.2500		
49	1.2077	105	1.6556	161	2.6316	217	6.4103		
50	1.2136	106	1.6667	162	2.6596	218	6.5789		
51	1.2195	107	1.6779	163	2.6882	219	6.7568		
52	1.2255	108	1.6892	164	2.7174	220	6.9444		
53	1.2315	109	1.7007	165	2.7473	221	7.1429		
54	1.2376	110	1.7123	166	2.7778	222	7.3529		
55	1.2438	111	1.7241	167	2.8090	223	7.5758		

Table 8-3. TC0OUT Frequency Table for Fosc = 16MHz, TC0 Rate = Fcpu/8

TIMER COUNTER 1 (TC1)

OVERVIEW

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



counter reference value, and an 8-bit counter register (TC1C) whose value is automatically incremented by counter logic.

Figure 8-6. Timer Count TC1 Block Diagram

The main purposes of the TC1 timer is as following.

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

TC1M MODE REGISTER

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

Eight rates for TC1 timer can be selected by TC1RATE0 ~ TC1RATE2 bits. 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 called TC1ENB is the control bit to start TC1 timer. The combination of these bits is to determine the TC1 timer clock frequency and the intervals.

TC1M initial value = 0000 0000

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

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.
TC1CKS: TC1 clock source select bit. “0” = Fcpu, “1” = External clock come from INT1/P0.1 pin. TC1 will be an event counter..
ALOAD1: TC1 auto-reload function control bit. 0 = none auto-reload, 1 = auto-reload.
TC1OUT: TC1 time-out toggle signal output control bit. 0 = To disable TC1 signal output and to enable P5.3’s I/O function, 1 = To enable TC1’s signal output and to disable P5.3’s I/O function. (Auto-disable the PWM1OUT function.)
PWM1OUT: TC1’s PWM output control bit. 0 = To disable the PWM output, 1 = To enable the PWM output (The TC1OUT control bit must = 0)

- ***Note1: When TC1CKS=1, TC0 became an external event counter. No more P0.1 interrupt request will be raised. (P0.1IRQ will be always 0)***

- ***Note2: The ICE S8KC do not support the PWM0OUT and TC0OUT Function. The PWM0OUT and TC0OUT must use the S8KD ICE (or later) to verify the function.***

TC1C COUNTING REGISTER

TC1C is an 8-bit counter register for the timer counter (TC1). TC1C must be reset whenever the TC1ENB is set "1" to start the timer. TC0C is incremented by one with a clock pulse which the frequency is determined by TC0RATE0 ~ TC0RATE2. When TC0C has incremented to "0FFH", it is will be cleared to "00H" in next clock and an overflow is generated. Under TC1 interrupt service request (TC1IEN) enable condition, the TC1 interrupt request flag will be set "1" and the system executes the interrupt service routine.

TC1C initial value = xxxx xxxx

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

The interval time of TC1 basic timer table.

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

Table 8-4. The Timing Table of Timer Count TC1

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.

TC1R initial value = xxxx xxxx

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

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

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

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

TC1 TIMER COUNTER OPERATION SEQUENCE

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

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

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

```
B0BCLR      FTC1IEN     ; To disable TC1 interrupt service
B0BCLR      FTC1ENB     ; To disable TC1 timer
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
                PUSH                        ; Push
                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:
                POP                         ; Pop
                B0XCH       A, ACCBUF       ; Restore ACC value.
                RETI                        ; Exit interrupt vector
```

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

```
                ORG         8               ; Interrupt vector
                JMP         INT_SERVICE
INT_SERVICE:
                B0XCH       A, ACCBUF       ; B0XCH doesn't change C, Z flag
                PUSH                        ; Push
                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:
                POP                         ; Pop
                B0XCH       A, ACCBUF       ; Restore ACC value.
                RETI                        ; Exit interrupt vector
```

TC1 CLOCK FREQUENCY OUTPUT (BUZZER)

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

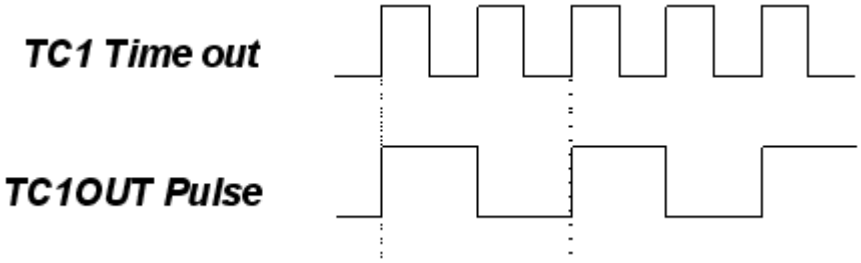


Figure 8-7. The TC1OUT Pulse Frequency

➲ **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
```

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

PWM FUNCTION DESCRIPTION

OVERVIEW

PWM function is generated by TC0/TC1 timer counter and output the PWM signal to PWM0OUT pin (P5.4)/ PWM1OUT pin (P5.3). The 8-bit counter counts modulus 256, from 0-255, inclusive. The value of the 8-bit counter is compared to the contents of the reference register (TC0R/TC1R). When the reference register value (TC0R/TC1R) is equal to the counter value (TC0C/TC1C), the PWM output goes low. When the counter reaches zero, the PWM output is forced high. The low-to-high ratio (duty) of the PWM0/PWM1 output is TC0R/256 and TC1R/256.

All PWM outputs remain inactive during the first 256 input clock signals. Then, when the counter value (TC0C/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 (TC0R/TC1R) and is programmed in increments of 1:256. The 8-bit PWM data register TC0R/TC1R is 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 TC0R/TC1R.

Reference Register Value (TC0R/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

Table 8-5. The PWM Duty Cycle Table

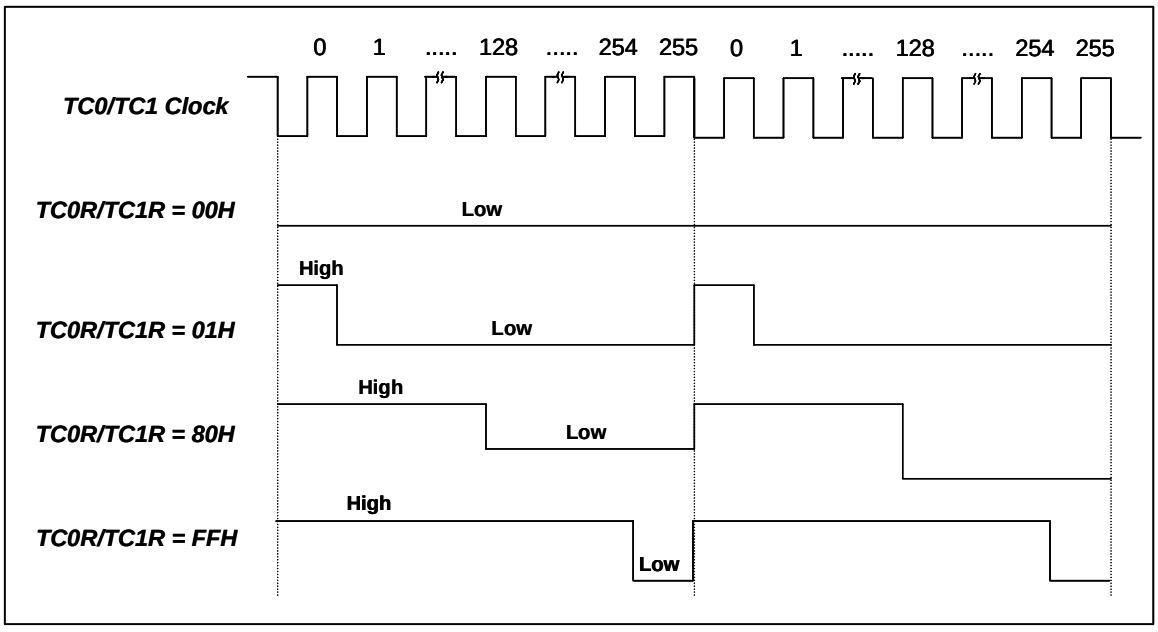


Figure 8-8 The Output of PWM with different TC0R/TC1R.

PWM PROGRAM DESCRIPTION

➲ **Example: Setup PWM0 output from TC0 to PWM0OUT (P5.4). 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. TC0 rate is Fcpu/4. The TC0RATE2~TC0RATE1 = 110. TC0C = TC0R = 30.**

```
MOV     A,#01100000B
B0MOV   TC0M,A          ; Set the TC0 rate to Fcpu/4
MOV     A,#0x00         ;First Time Initial TC0
B0MOV   TC0C,A

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

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

➢ ***Note1: The TC0R and TC1R are write-only registers. Don't process them using INCMS, DECMS instructions.***

➢ ***Note2: Set TC0C at initial is to make first duty-cycle correct. After TC0 is enabled, don't modify TC0R value to avoid duty cycle error of PWM output.***

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

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

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

➢ ***Note2: That is better to set the TC0C and TC0R value together when PWM0 duty modified. It protects the PWM0 signal no glitch as PWM0 duty changing. That is better to set the TC1C and TC1R value together when PWM1 duty modified. It protects the PWM1 signal no glitch as PWM1 duty changing.***

➢ ***Note3: The TC0OUT function must be set "0" when PWM0 output enable. The TC1OUT function must be set "0" when PWM1 output enable.***

➢ ***Note4: The PWM can work with interrupt request.***

9 INTERRUPT

OVERVIEW

The SN8P1800 provides 7[1] interrupt sources, including four internal interrupts (T0, TC0, TC1 & SIO) and three external interrupts (INT0 ~ INT2). These external interrupts can wakeup the chip from power down mode to high-speed normal mode. The external clock input pins of INT0/INT1/INT2 are shared with P0.0/P0.1/P0.2 pins. Once interrupt service is executed, the GIE bit in STKP register will clear to "0" for stopping other interrupt request. When interrupt service exits, the GIE bit will set to "1" to accept the next interrupts' request. All of the interrupt request signals are stored in INTRQ register. The user can program the chip to check INTRQ's content for setting executive priority.

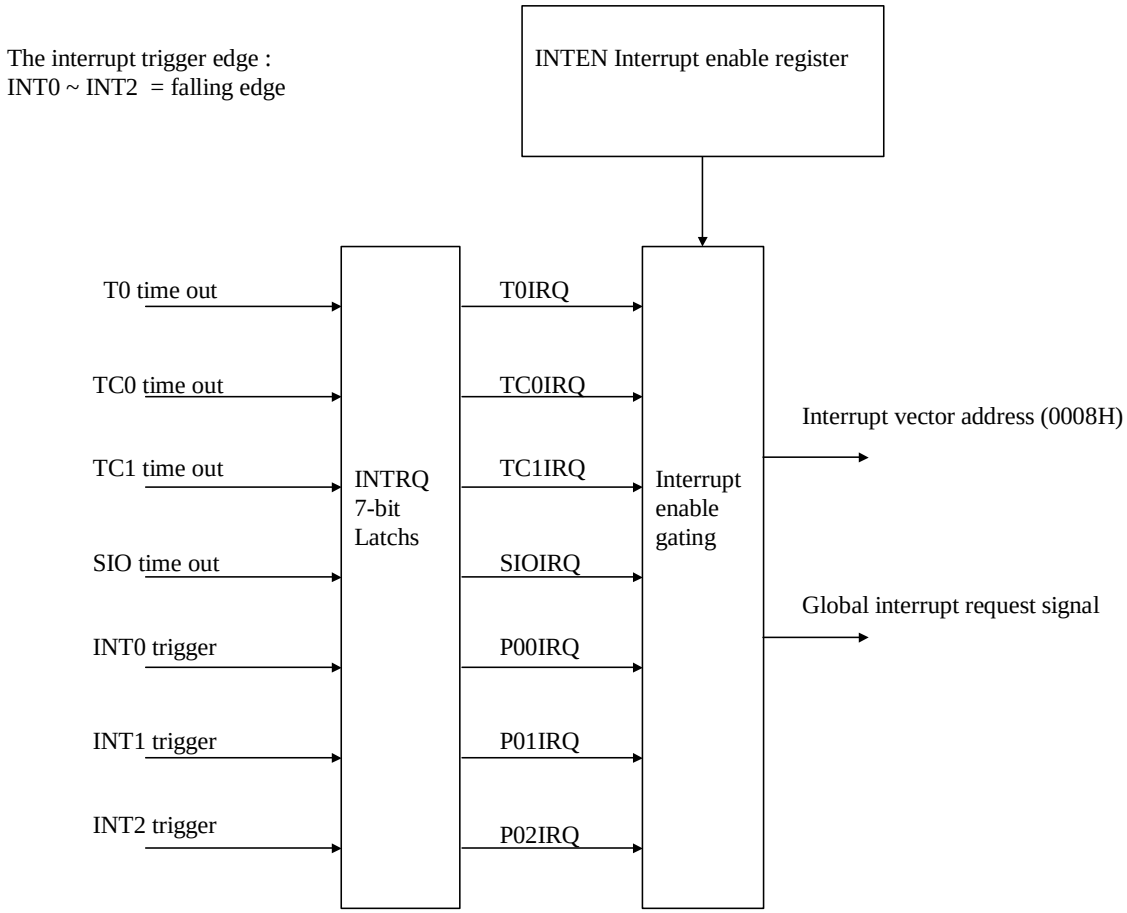


Figure 9-1. The 7 Interrupts of SN8P1800

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

INTEN INTERRUPT ENABLE REGISTER

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

INTEN initial value = x000 0000

0C9H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
INTEN	0	TC1IEN	TC0IEN	T0IEN	SIOIEN	P02IEN	P01IEN	P00IEN
	-	R/W	R/W	R/W	R/W	R/W	R/W	R/W

P00IEN : External P0.0 interrupt control bit. 0 = disable, 1 = enable.
P01IEN : External P0.1 interrupt control bit. 0 = disable, 1 = enable.
P02IEN : External P0.2 interrupt control bit. 0 = disable, 1 = enable.
SIOIEN : SIO interrupt control bit. 0 = disable, 1 = enable.
T0IEN : T0 timer interrupt control bit. 0 = disable, 1 = enable.
TC0IEN : Timer 0 interrupt control bit. 0 = disable, 1 = enable.
TC1IEN : Timer 1 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 these interrupt request occurs, the bit of the INTRQ register would be set "1". The INTRQ value needs to be clear by programming after detecting the flag. In the interrupt vector of program, users know the any interrupt requests occurring by the register and do the routine corresponding of the interrupt request.

INTRQ initial value = x000 0000

0C8H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
INTRQ	0	TC1IRQ	TC0IRQ	T0IRQ	SIOIRQ	P02IRQ	P01IRQ	P00IRQ
	-	R/W	R/W	R/W	R/W	R/W	R/W	R/W

P00IRQ : External P0.0 interrupt request bit. 0 = non-request, 1 = request.
P01IRQ : External P0.1 interrupt request bit. 0 = non-request, 1 = request.
P02IRQ : External P0.2 interrupt request bit. 0 = non-request, 1 = request.
SIOIRQ : SIO interrupt request bit. 0 = non-request, 1 = request.
T0IRQ : T0 timer interrupt request control bit. 0 = non request, 1 = request.
TC0IRQ : TC0 timer interrupt request controls bit. 0 = non request, 1 = request.
TC1IRQ : TC1 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

SN8P1800 provides 7 interrupts. The operation of the 7 interrupts is as following.

GIE GLOBAL INTERRUPT OPERATION

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

STKP initial value = 0xxx 1111

0DFH	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
STKP	GIE	-	-	-	STKPB3	STKPB2	STKPB1	STKPB0
	R/W	-	-	-	R/W	R/W	R/W	R/W

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

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

```
B0BSET      FGIE            ; Enable GIE
```

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

INT0 (P0.0) INTERRUPT OPERATION

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

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

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

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

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

        B0XCH       A, ACCBUF       ; Store ACC value.
        PUSH                        ; Push

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

        B0BCLR      FP00IRQ         ; Reset P00IRQ
        .           .               ; INT0 interrupt service routine
        .           .
EXIT_INT:
        POP                         ; Pop
        B0XCH       A, ACCBUF       ; Restore ACC value.

        RETI                        ; Exit interrupt vector
```

➢ ***Note: The PUSH and POP instruction only save L,H,R,Z,Y,X,PFLAG and RBANK registers but A register. User must save register A by B0XCH instruction when PUSH command is used.***

INT1 (P0.1) INTERRUPT OPERATION

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

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

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

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

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

                B0XCH       A, ACCBUF       ; Store ACC value.
                PUSH                        ; Push

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

                B0BCLR      FP01IRQ         ; Reset P01IRQ
                .           .               ; INT1 interrupt service routine
                .           .
EXIT_INT:
                POP                         ; Pop
                B0XCH       A, ACCBUF       ; Restore ACC value.

                RETI                        ; Exit interrupt vector
```

INT2 (P0.2) INTERRUPT OPERATION

The INT2 is triggered by falling edge. When the INT2 trigger occurs, the P02IRQ will be set to “1” however the P02IEN is enable or disable. If the P02IEN = 1, the trigger event will make the P02IRQ to be “1” and the system enter interrupt vector. If the P02IEN = 0, the trigger event will make the P02IRQ to be “1” but the system will not enter interrupt vector. Users need to care for the operation under multi-interrupt situation.

➲ **Example: INT2 interrupt request setup.**

```
                B0BSET      FP02IEN         ; Enable INT2 interrupt service
                B0BCLR      FP02IRQ         ; Clear INT2 interrupt request flag
                B0BSET      FGIE            ; Enable GIE
```

➲ **Example: INT2 interrupt service routine.**

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

                B0XCH       A, ACCBUF       ; Store ACC value.
                PUSH                        ; Push

                B0BTS1      FP02IRQ         ; Check P02IRQ
                JMP         EXIT_INT        ; P02IRQ = 0, exit interrupt vector

                B0BCLR      FP02IRQ         ; Reset P02IRQ
                .           .               ; INT2 interrupt service routine
                .           .
EXIT_INT:
                POP                         ; Pop
                B0XCH       A, ACCBUF       ; Restore ACC value.

                RETI                        ; Exit interrupt vector
```

T0 INTERRUPT OPERATION

When the T0C counter occurs overflow, the T0IRQ will be set to “1” however the T0IEN is enable or disable. If the T0IEN = 1, the trigger event will make the T0IRQ to be “1” and the system enter interrupt vector. If the T0IEN = 0, the trigger event will make the T0IRQ to be “1” but the system will not enter interrupt vector. Users need to care for the operation under multi-interrupt situation.

➲ **Example: T0 interrupt request setup.**

```
        B0BCLR      FT0IEN      ; Disable T0 interrupt service
        B0BCLR      FT0ENB      ; Disable T0 timer
        MOV         A, #20H     ;
        B0MOV       T0M, A      ; Set T0 clock = Fcpu / 64
        MOV         A, #74H     ; Set T0C initial value = 74H
        B0MOV       T0C, A      ; Set T0 interval = 10 ms

        B0BSET      FT0IEN      ; Enable T0 interrupt service
        B0BCLR      FT0IRQ      ; Clear T0 interrupt request flag
        B0BSET      FT0ENB      ; Enable T0 timer

        B0BSET      FGIE        ; Enable GIE
```

➲ **Example: T0 interrupt service routine.**

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

        B0XCH       A, ACCBUF   ; Store ACC value.
        PUSH                    ; Push

        B0BTS1      FT0IRQ      ; Check T0IRQ
        JMP         EXIT_INT    ; T0IRQ = 0, exit interrupt vector

        B0BCLR      FT0IRQ      ; Reset T0IRQ
        MOV         A, #74H
        B0MOV       T0C, A      ; Reset T0C.
        .           .           ; T0 interrupt service routine
        .           .
EXIT_INT:
        POP                     ; Pop
        B0XCH       A, ACCBUF   ; Restore ACC value.

        RETI                    ; Exit interrupt vector
```

TC0 INTERRUPT OPERATION

When the TC0C counter occurs overflow, the TC0IRQ will be set to “1” however the TC0IEN is enable or disable. If the TC0IEN = 1, the trigger event will make the TC0IRQ to be “1” and the system enter interrupt vector. If the TC0IEN = 0, the trigger event will make the TC0IRQ to be “1” but the system will not enter interrupt vector. Users need to care for the operation under multi-interrupt situation.

➲ **Example: TC0 interrupt request setup.**

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

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

        B0BSET      FGIE            ; Enable GIE
```

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

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

        B0XCH       A, ACCBUF       ; Store ACC value.
        PUSH                        ; Push

        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:
        POP                         ; Pop
        B0XCH       A, ACCBUF       ; Restore ACC value.

        RETI                        ; Exit interrupt vector
```

TC1 INTERRUPT OPERATION

When the TC1C counter occurs overflow, the TC1IRQ will be set to “1” however the TC1IEN is enable or disable. If the TC1IEN = 1, the trigger event will make the TC1IRQ to be “1” and the system enter interrupt vector. If the TC1IEN = 0, the trigger event will make the TC1IRQ to be “1” but the system will not enter interrupt vector. Users need to care for the operation under multi-interrupt situation.

➲ **Example: TC1 interrupt request setup.**

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

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

        B0BSET      FGIE            ; Enable GIE
```

➲ **Example: TC1 interrupt service routine.**

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

        B0XCH       A, ACCBUF       ; Store ACC value.
        PUSH                        ; Push

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

        B0BCLR      FTC1IRQ         ; Reset TC1IRQ
        MOV         A, #74H
        B0MOV       TC1C, A         ; Reset TC1C.
        .           .               ; TC1 interrupt service routine
        .           .
EXIT_INT:
        POP                         ; Pop
        B0XCH       A, ACCBUF       ; Restore ACC value.

        RETI                        ; Exit interrupt vector
```

SIO INTERRUPT OPERATION

When the SIO finished transmitting, the SIOIRQ will be set to “1” however the SIOIEN is enable or disable. If the SIOIEN = 1, the trigger event will make the SIOIRQ to be “1” and the system enter interrupt vector. If the SIOIEN = 0, the trigger event will make the SIOIRQ to be “1” but the system will not enter interrupt vector. Users need to care for the operation under multi-interrupt situation.

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

```
        B0BSET      FSIOIEN         ; Enable SIO interrupt service
        B0BCLR      FSIOIRQ         ; Clear SIO interrupt request flag
        B0BSET      FGIE            ; Enable GIE
```

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

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

        B0XCH       A, ACCBUF       ; Store ACC value.
        PUSH                        ; Push

        B0BTS1      FSIOIRQ         ; Check SIOIRQ
        JMP         EXIT_INT        ; SIOIRQ = 0, exit interrupt vector

        B0BCLR      FSIOIRQ         ; Reset SIOIRQ
        .           .               ; SIO interrupt service routine
        .           .
EXIT_INT:
        POP                         ; Pop
        B0XCH       A, ACCBUF       ; Restore ACC value.

        RETI                        ; Exit interrupt vector
```

MULTI-INTERRUPT OPERATION

In most conditions, the software designer uses more than one interrupt request. Processing multi-interrupt request needs to set the priority of these interrupt requests. The IRQ flags of the 7 interrupt are controlled by the interrupt event occurring. But the IRQ flag set doesn't mean the system to execute the interrupt vector. The IRQ flags can be triggered by the events without interrupt enable. Just only any the event occurs and the IRQ will be logic "1". The IRQ and its trigger event relationship is as the below table.

Interrupt Name	*Trigger Event Description*
P00IRQ	P0.0 trigger. Falling edge.
P01IRQ	P0.1 trigger. Falling edge.
P02IRQ	P0.2 trigger. Falling edge.
T0IRQ	T0C overflow.
TC0IRQ	TC0C overflow.
TC1IRQ	TC1C overflow.
SIOIRQ	End of SIO transmitter operating.

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

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

```
        ORG     8               ; Interrupt vector

        B0XCH   A, ACCBUF       ; Store ACC value.
        PUSH                    ; Push
INTP00CHK:                      ; Check INT0 interrupt request
        B0BTS1  FP00IEN         ; Check P00IEN
        JMP     INTP01CHK       ; Jump check to next interrupt
        B0BTS0  FP00IRQ         ; Check P00IRQ
        JMP     INTP00          ; Jump to INT0 interrupt service routine
INTP01CHK:                      ; Check INT1 interrupt request
        B0BTS1  FP01IEN         ; Check P01IEN
        JMP     INTP02CHK       ; Jump check to next interrupt
        B0BTS0  FP01IRQ         ; Check P01IRQ
        JMP     INTP01          ; Jump to INT1 interrupt service routine
INTP02CHK:                      ; Check INT2 interrupt request
        B0BTS1  FP02IEN         ; Check P02IEN
        JMP     INTT0CHK        ; Jump check to next interrupt
        B0BTS0  FP02IRQ         ; Check P02IRQ
        JMP     INTP02          ; Jump to INT2 interrupt service routine
INTT0CHK:                       ; Check T0 interrupt request
        B0BTS1  FT0IEN          ; Check T0IEN
        JMP     INTTC0CHK       ; Jump check to next interrupt
        B0BTS0  FT0IRQ          ; Check T0IRQ
        JMP     INTT0           ; Jump to T0 interrupt service routine
INTTC0CHK:                      ; Check TC0 interrupt request
        B0BTS1  FTC0IEN         ; Check TC0IEN
        JMP     INTTC1CHK       ; Jump check to next interrupt
        B0BTS0  FTC0IRQ         ; Check TC0IRQ
        JMP     INTTC0          ; Jump to TC0 interrupt service routine
INTTC1HK:                       ; Check TC1 interrupt request
        B0BTS1  FTC1IEN         ; Check TC1IEN
        JMP     INTSIOCHK       ; Jump check to next interrupt
        B0BTS0  FTC1IRQ         ; Check TC1IRQ
        JMP     INTTC1          ; Jump to TC1 interrupt service routine
INTSIOCHK:                      ; Check SIO interrupt request
        B0BTS1  FSIOIEN         ; Check SIOIEN
        JMP     INT_EXIT        ; Jump to exit of IRQ
        B0BTS0  FSIOIRQ         ; Check SIOIRQ
        JMP     INTSIO          ; Jump to SIO interrupt service routine
INT_EXIT:
        POP                     ; Pop
        B0XCH   A, ACCBUF       ; Restore ACC value.

        RETI                    ; Exit interrupt vector
```

10 SERIAL INPUT/OUTPUT TRANSCEIVER (SIO)

OVERVIEW

The SN8P1800provides an 8-bit SIO interface circuit with clock rate selection. The SIOM register can control SIO operating function, such as: transmit/receive, clock rate, transfer edge and starting this circuit. This SIO circuit will TX or RX 8-bit data automatically by setting SENB and START bits in SIOM register. The SIOB is an 8-bit buffer, which is designed to store transfer data. SIOC and SIOR are designed to generate SIO's clock source with auto-reload function. The 3-bit I/O counter can monitor the operation of SIO and announce an interrupt request after transmitting/receiving 8 bits data. After transferring 8-bit data, this circuit will be disabled automatically and re-transfer data by programming SIOM register.

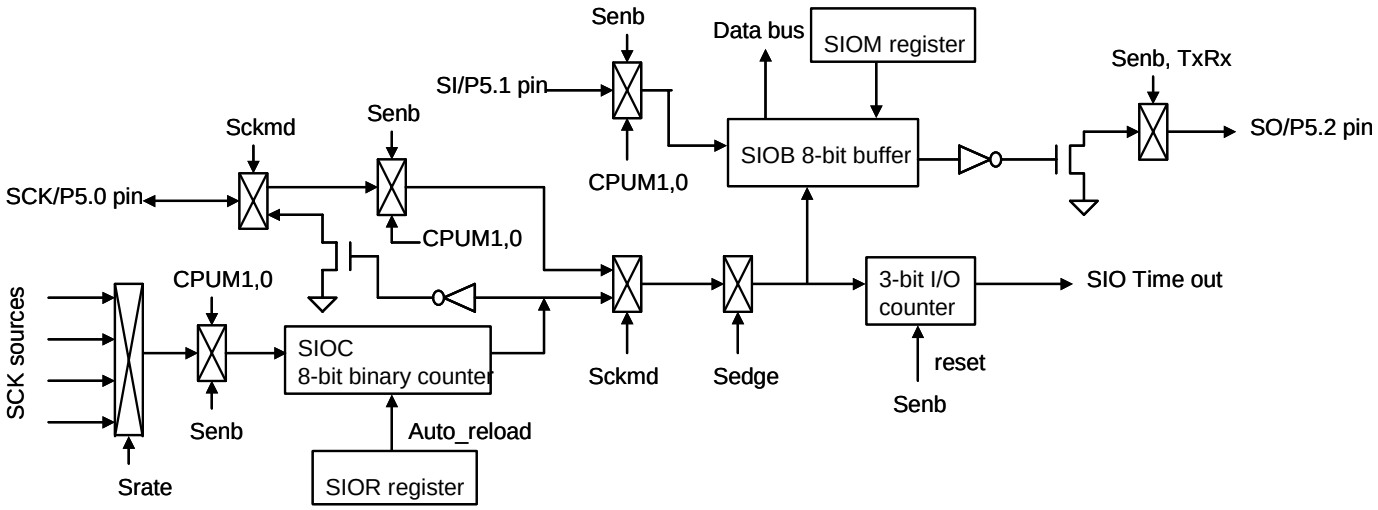


Figure 10-1. SIO Interface Circuit Diagram

Figure 9-2 shows a typical transfer between two micro-controllers. Process 1 sends SCK for initial the data transfer. Both processors must work in the same clock edge direction, then both controllers would send and receive data at the same time.

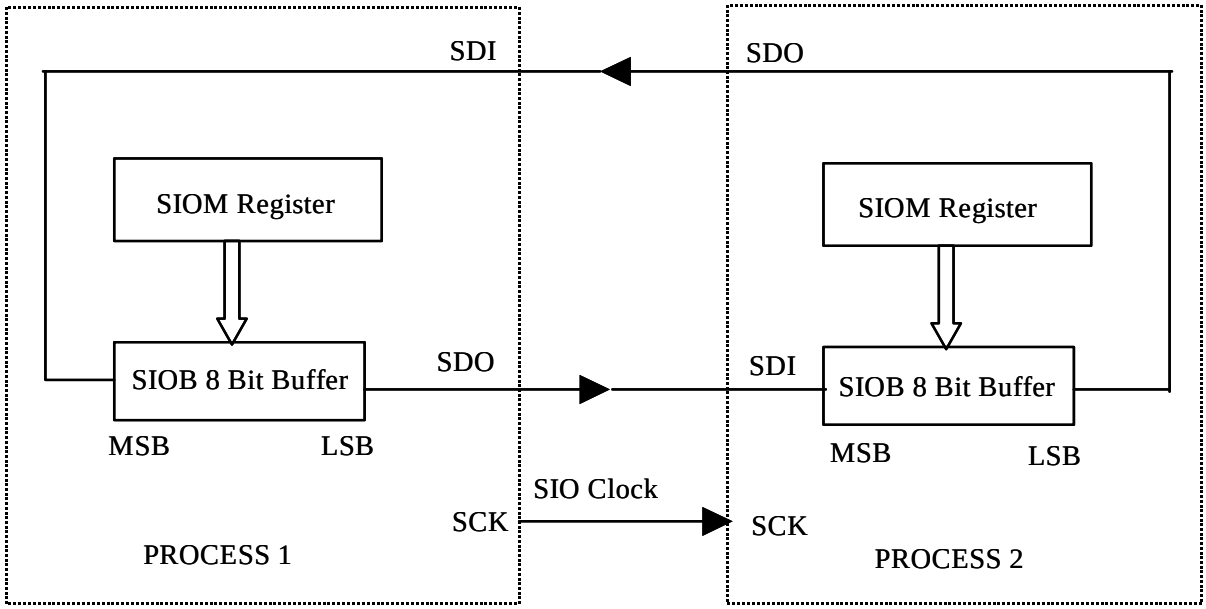


Figure 10-2. SIO Data Transfer Diagram

SIOM MODE REGISTER

SIOM initial value = 0000 x000

0B4H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
SIOM	SENB	START	SRATE1	SRATE0	0	SCKMD	SEDGE	TXRX
	R/W	R/W	R/W	R/W	-	R/W	R/W	R/W

SENB: SIO function control bit. 0 = disable (P5.0~P5.2 is general purpose port), 1 = enable (P5.0~P5.2 is SIO pins).
START: SIO progress control bit. 0 = End of transfer, 1 = progressing.
SRATE1, 0: SIO's transfer rate select bit. 00 = fcpu, 01 = fcpu/32, 10 = fcpu/16, 11 = fcpu/8.
(Note: These 2-bits are workless when SCKMD=1)
SCKMD: SIO's clock mode select bit. 0 = internal, 1 = external mode.
SEDGE: SIO's transfer clock edge select bit. 0 = falling edge, 1 = raising edge.
TXRX: SIO's transfer direction select bit. 0 = receiver only , 1 = transmitter/receiver full duplex.

- ***Note 1: If SCKMD=1 for external clock, the SIO is in SLAVE mode.***
  If SCKMD=0 for internal clock, the SIO is in MASTER mode.
- ***Note 2: Don't set SENB and START bits in the same time. That makes the SIO function error.***

Because SIO function is shared with Port5 for P5.0 as SCK, P5.1 as SI and P5.2 as SO
The following table shown the Port5[2:0] I/O mode behavior and setting when SIO function enable and disable

SENB=1 (SIO Function Enable)		
P5.0/SCK	(SCKMD=1) SIO source = External clock	P5.0 will change to Input mode automatically, no matter what P5M setting
	(SCKMD=0) SIO source = Internal clock	P5.0 will change to Output mode automatically, no matter what P5M setting
P5.1/SI	P5.1 must be set as Input mode in P5M ,or the SIO function will be abnormal	
P5.2/SO	(TXRX=1) SIO = Transmitter/Receiver	P5.2 will change to Output mode automatically, no matter what P5M setting
	(TXRX=0) SIO = Receiver only	P5.2 will change to Input mode automatically, no matter what P5M setting
SENB=0 (SIO Function Disable)		
P5.0/P5.1/P5.2	Port5[2:0] I/O mode are fully controlled by P5M when SIO function Disable	

SIOB DATA BUFFER

SIOB initial value = 0000 0000

0B6H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
SIOB	SIOB7	SIOB6	SIOB5	SIOB4	SIOB3	SIOB2	SIOB1	SIOB0
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

SIOB is the SIO data buffer register. It stores serial I/O transmit and receive data.

SIOR REGISTER DESCRIPTION

SIOR initial value = 0000 0000

0B5H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
SIOR	SIOR7	SIOR6	SIOR5	SIOR4	SIOR3	SIOR2	SIOR1	SIOR0
	W	W	W	W	W	W	W	W

The SIOR is designed for the SIO counter to reload the counted value when end of counting. It is like a post-scaler of SIO clock source and let SIO has more flexible to setting SCK range. Users can set the SIOR value to setup SIO transfer time. To setup SIOR value equation to desire transfer time is as following.

SCK frequency = SIO rate / (256 - SIOR)

SIOR = 256 - (1 / (SCK frequency) * SIO rate / 2)

➲ **Example: Setup the SIO clock to be 5KHz. Fosc = 3.58MHz. SIO's rate = Fcpu = Fosc/4.**

SIOR = 256 – (1/(5KHz) * 3.58MHz/4)
= 256 – 89
= 167

SIO MASTER OPERATING DESCRIPTION

Under master-transmitter situation, the SCK has two directions as following.

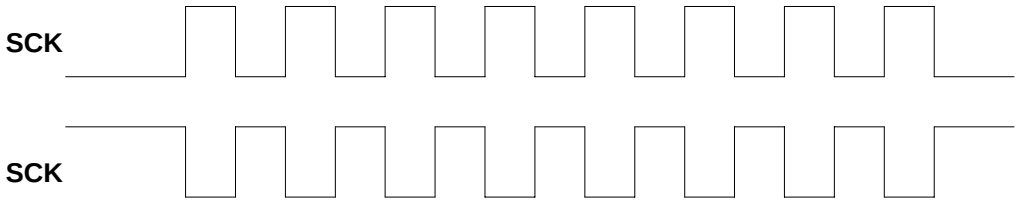


Figure 10-3. The Two SCK Directions of SIO Master Operation

RISING EDGE TRANSMITTER/RECEIVER MODE

➲ **Example: Master Tx/Rx rising edge**

```
        MOV     A,TXDATA    ; Load transmitted data into SIOB register.
        B0MOV   SIOB,A
        MOV     A,#0FFH     ; Set SIO clock with auto-reload function.
        B0MOV   SIOR,A
        MOV     A,#10000011B; Setup SIOM and enable SIO function. Rising edge.
        B0MOV   SIOM,A
        B0BSET  FSTART      ; Start transfer and receiving SIO data.
CHK_END:
        B0BTS0  FSTART      ; Wait the end of SIO operation.
        JMP     CHK_END
        B0MOV   A,SIOB      ; Save SIOB data into RXDATA buffer.
        MOV     RXDATA,A
```

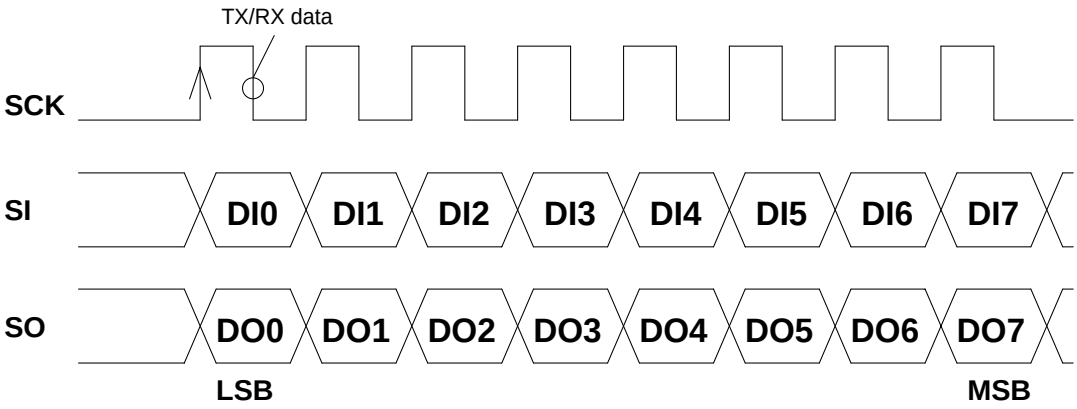


Figure 10-4. The Rising Edge Timing Diagram of Master Transfer and Receiving Operation

FALLING EDGE TRANSMITTER/RECEIVER MODE

➲ **Example: Master Tx/Rx falling edge**

```
        MOV     A,TXDATA        ; Load transmitted data into SIOB register.
        B0MOV   SIOB,A
        MOV     A,#0FFH         ; Set SIO clock with auto-reload function.
        B0MOV   SIOR,A
        MOV     A,#10000001B    ; Setup SIOM and enable SIO function. Falling edge.
        B0MOV   SIOM,A
        B0BSET  FSTART          ; Start transfer and receiving SIO data.
CHK_END:
        B0BTS0  FSTART          ; Wait the end of SIO operation.
        JMP     CHK_END
        B0MOV   A,SIOB          ; Save SIOB data into RXDATA buffer.
        MOV     RXDATA,A
```

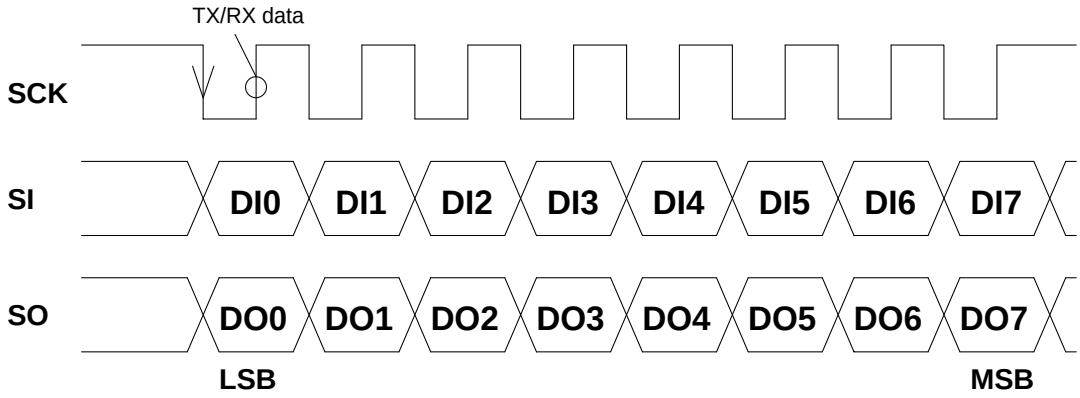


Figure 10-5. The Falling Edge Timing Diagram of Master Transfer and Receiving Operation

RISING EDGE RECEIVER MODE

➲ **Example: Master Rx rising edge**

```
        MOV     A,#0FFH         ; Set SIO clock with auto-reload function.
        B0MOV   SIOR,A
        MOV     A,#10000010B    ; Setup SIOM and enable SIO function. Rising edge.
        B0MOV   SIOM,A
        B0BSET  FSTART          ; Start receiving SIO data.
CHK_END:
        B0BTS0  FSTART          ; Wait the end of SIO operation.
        JMP     CHK_END
        B0MOV   A,SIOB          ; Save SIOB data into RXDATA buffer.
        MOV     RXDATA,A
```

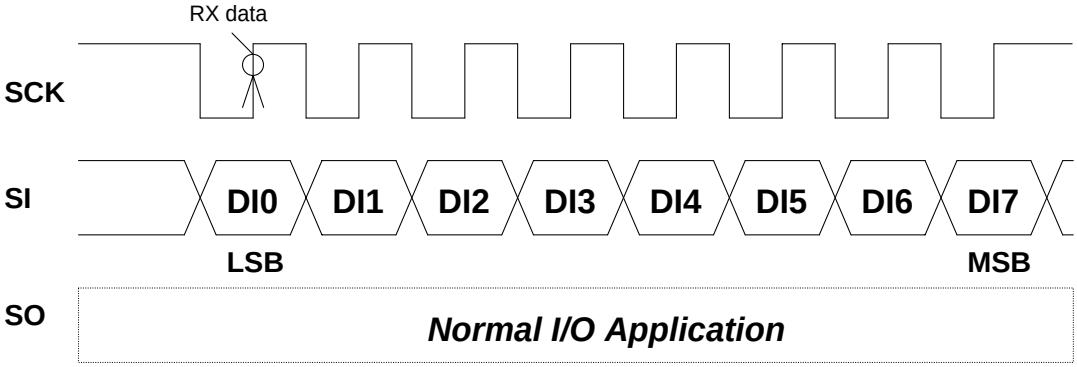


Figure 10-6. The Rising Edge Timing Diagram of Master Receiving Operation

FALLING EDGE RECEIVER MODE

➲ **Example: Master Rx falling edge**

```
        MOV     A,#0FFH         ; Set SIO clock with auto-reload function.
        B0MOV   SIOR,A
        MOV     A,#10000000B    ; Setup SIOM and enable SIO function. Falling edge.
        B0MOV   SIOM,A
        B0BSET  FSTART          ; Start receiving SIO data.
CHK_END:
        B0BTS0  FSTART          ; Wait the end of SIO operation.
        JMP     CHK_END
        B0MOV   A,SIOB          ; Save SIOB data into RXDATA buffer.
        MOV     RXDATA,A
```

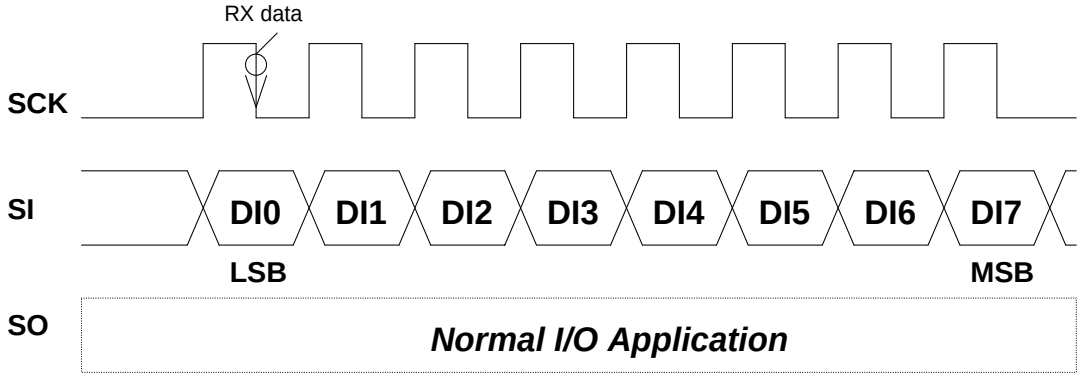


Figure 10-7. The Falling Edge Timing Diagram of Master Receiving Operation

SIO SLAVE OPERATING DESCRIPTION

Under slave-receiver situation, the SCK has four phases as following.

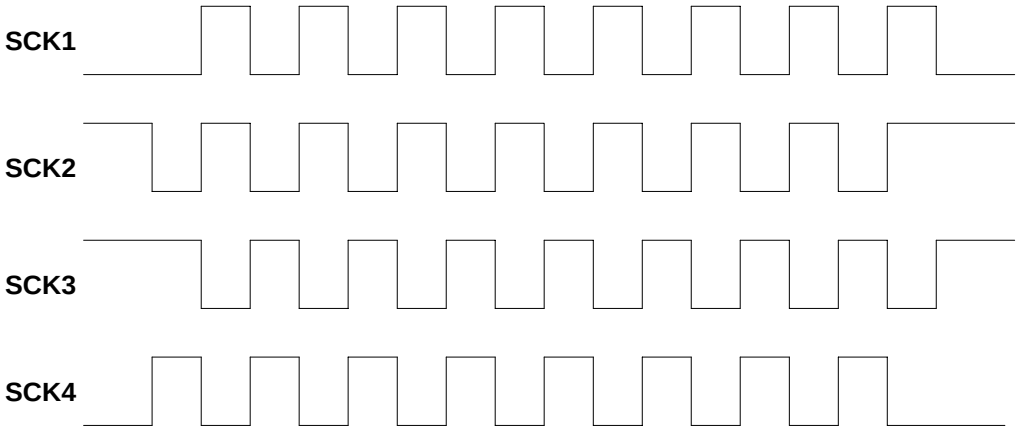


Figure 10-8. The Four Phases SCK clock of SIO Slave Operation.

RISING EDGE TRANSMITTER/RECEIVER MODE

➲ **Example: Slave Tx/Rx rising edge**

```
        MOV     A,TXDATA        ; Load transfer data into SIOB register.
        B0MOV   SIOB,A
        MOV     A,# 10000111B   ; Setup SIOM and enable SIO function. Rising edge.
        B0MOV   SIOM,A
        B0BSET  FSTART          ; Start transfer and receiving SIO data.
CHK_END:
        B0BTS0  FSTART          ; Wait the end of SIO operation.
        JMP     CHK_END
        B0MOV   A,SIOB          ; Save SIOB data into RXDATA buffer.
        MOV     RXDATA,A
```

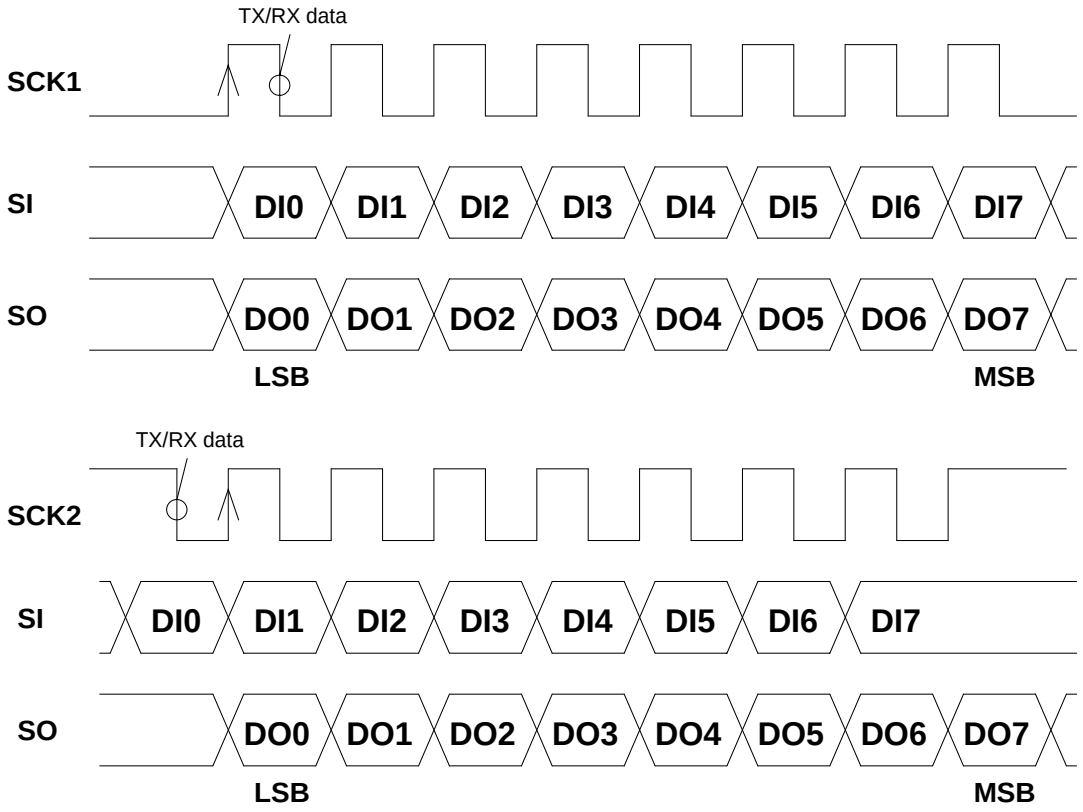


Figure 10-9. The Rising Edge Timing Diagram of Slave Transfer and Receiving Operation

FALLING EDGE TRANSMITTER/RECEIVER MODE

➲ **Example: Slave Tx/Rx falling edge**

```
        MOV     A,TXDATA        ; Load transfer data into SIOB register.
        B0MOV   SIOB,A
        MOV     A,# 10000101B   ; Setup SIOM and enable SIO function. Falling edge.
        B0MOV   SIOM,A
        B0BSET  FSTART          ; Start transfer and receiving SIO data.
CHK_END:
        B0BTS0  FSTART          ; Wait the end of SIO operation.
        JMP     CHK_END
        B0MOV   A,SIOB          ; Save SIOB data into RXDATA buffer.
        MOV     RXDATA,A
```

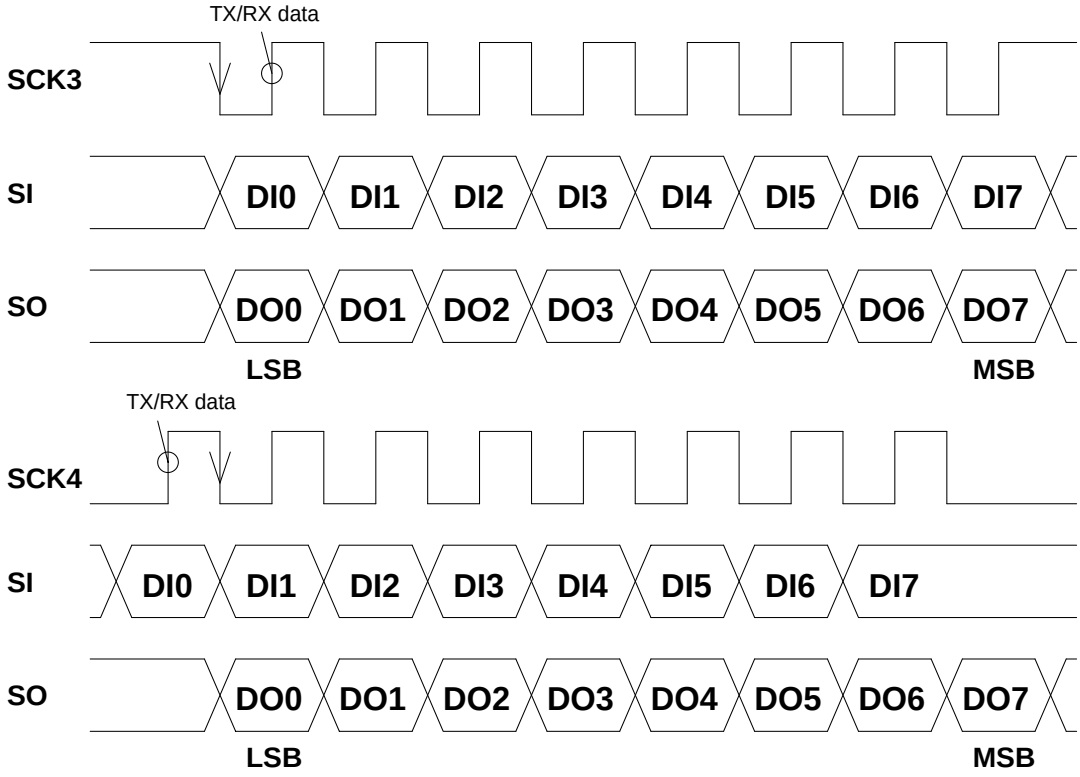


Figure 10-10. The Falling Edge Timing Diagram of Slave Transfer and Receiving Operation

RISING EDGE RECEIVER MODE

➲ **Example: Slave Rx rising edge**

```
        MOV     A,# 10000110B   ; Setup SIOM and enable SIO function. Rising edge.
        B0MOV   SIOM,A
        B0BSET  FSTART          ; Start receiving SIO data.
CHK_END:
        B0BTS0  FSTART          ; Wait the end of SIO operation.
        JMP     CHK_END
        B0MOV   A,SIOB          ; Save SIOB data into RXDATA buffer.
        MOV     RXDATA,A
```

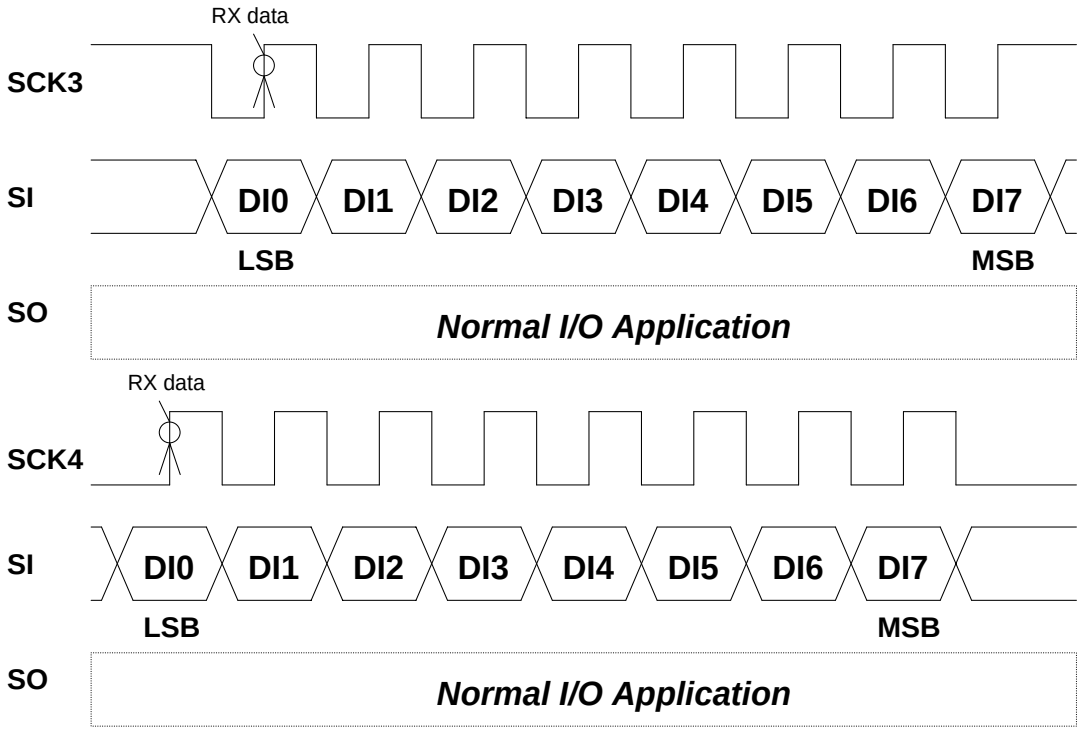


Figure 10-11. The Rising Edge Timing Diagram of Slave Receiving Operation

FALLING EDGE RECEIVER MODE

➲ **Example: Slave Rx falling edge**

```
        MOV     A,# 10000100B   ; Setup SIOM and enable SIO function. Falling edge.
        B0MOV   SIOM,A
        B0BSET  FSTART          ; Start receiving SIO data.
CHK_END:
        B0BTS0  FSTART          ; Wait the end of SIO operation.
        JMP     CHK_END
        B0MOV   A,SIOB          ; Save SIOB data into RXDATA buffer.
        MOV     RXDATA,A
```

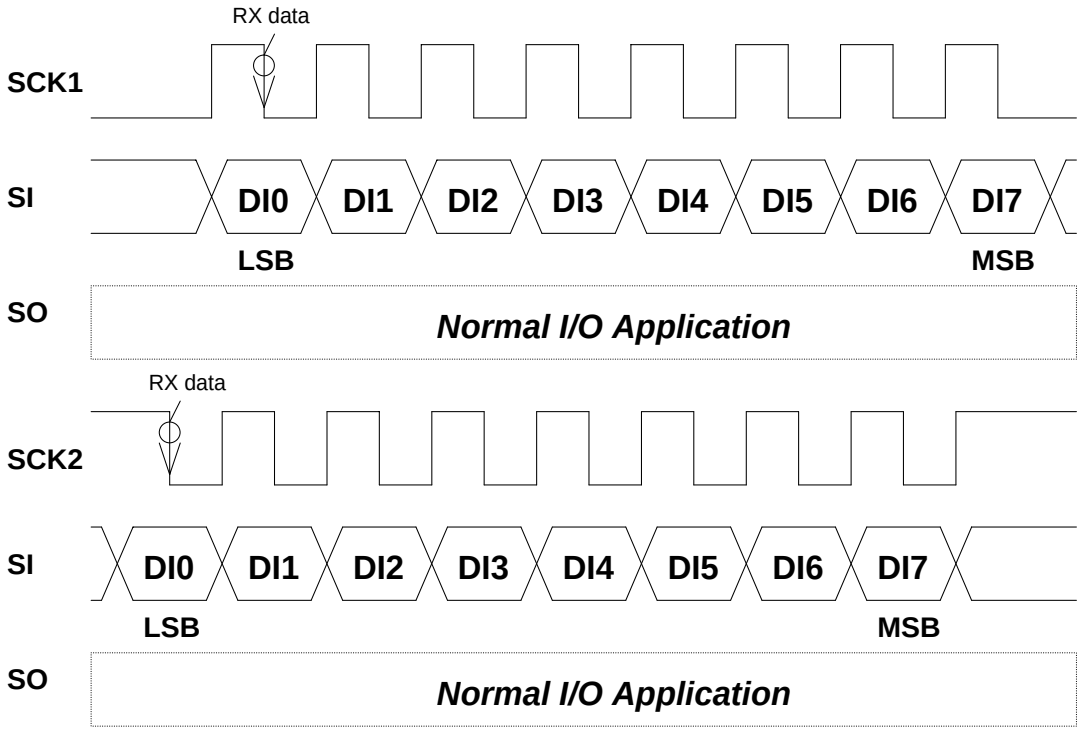


Figure 10-12. The Falling Edge Timing Diagram of Slave Receiving Operation

SIO INTERRUPT OPERATION DESCRIPTION

The SIO provides an interrupt function. Users can process SIO data after the SIO interrupt request occurring. There is a example for the application as following.

➲ **Example: SIO interrupt demo routine.**

```
Main:
        MOV     A,# 10000100B   ; Setup SIOM and enable SIO function. Falling edge.
        B0MOV   SIOM,A
        B0BSET  FSTART          ; Start transfer SIO data.
        .       .
        .       .
        JMP     MAIN



        ORG     8               ; Interrupt vector

        B0XCH   A, ACCBUF
        PUSH

        B0BTS1  FSIOIRQ
        JMP     INT_EXIT
        B0MOV   A,SIOB          ; Save SIOB data into RXDATA buffer.
        MOV     RXDATA,A
        B0BCLR  FSIOIRQ         ; Clear SIO interrupt request flag.

INT_EXIT:
        POP
        B0XCH   A, ACCBUF
```

11 I/O PORT

OVERVIEW

The SN8P1800 provides up to 5 ports for users' application, consisting of one input only port (P0), four I/O ports (P1, P2, P4, P5). The direction of I/O port is selected by PnM register.

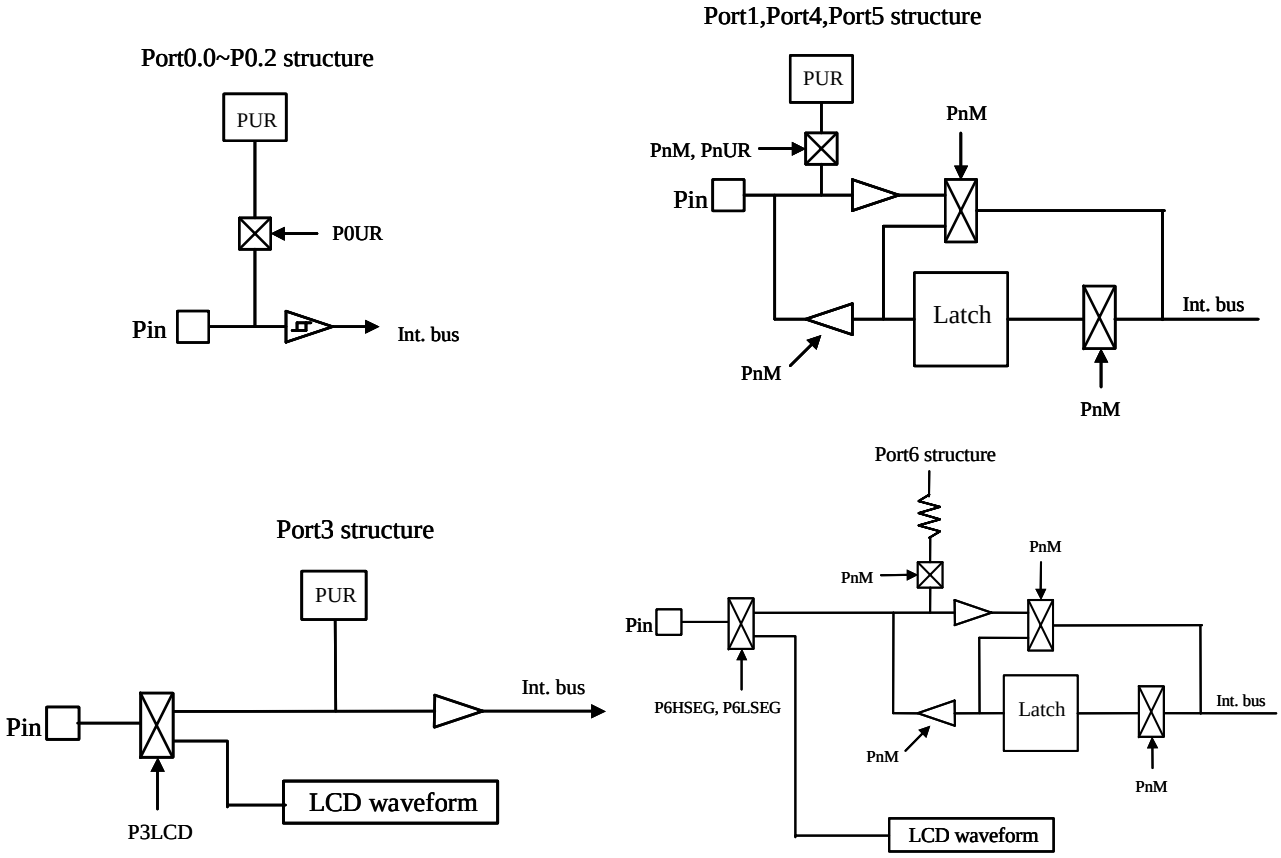


Figure 11-1. The I/O Port Block Diagram

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

I/O PORT FUNCTION TABLE

Port/Pin	I/O	Function Description	Remark
P0.0~P0.2	I	General-purpose input function	Programmable pull-up
		External interrupt (INT0~INT2)	
		Wakeup for power down mode	
P0.3~P0.5	I	General-purpose input function	Programmable pull-up
		Wakeup for power down mode	
		LCD common	
P1.0~P1.5	I/O	General-purpose input/output function	Programmable pull-up
		Wakeup for power down mode	
P2.0~P2.7	O	General-purpose output function	
		LCD segment	
P3.0~P3.7	I/O	General-purpose input function	Pull-up at input mode
		LCD segment	
P4.0~P4.7	I/O	General-purpose input/output function	Programmable pull-up
		ADC analog signal input	
P5.0	I/O	General-purpose input/output function	Programmable pull-up
		SIO clock pin.	
P5.1	I/O	General-purpose input/output function	
	I	SIO data input pin.	***P5M.1 must be set "0"***
P5.2	I/O	General-purpose input/output function	
	O	SIO data output pin.	***P5M.1 must be set "1"***
P5.3~P5.4	I/O	General-purpose input/output function	
P6.0~P6.7	I/O	General-purpose input/output function	Pull-up at input mode
		LCD segment	

Table 11-1. I/O Function Table

PULL-UP RESISTOR (PnUR) REGISTER

PnUR initial value = 0000 0000

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
PnUR	Pn7R	Pn6R	Pn5R	Pn4R	Pn3R	Pn2R	Pn1R	Pn0R
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

PnUR : The n expressed 0, 1, 4, 5.
Pn7R ~ Pn0R : Pull-up resistor control bit. 0 = without pull up resistor, 1 = with pull up resistor.

I/O PORT MODE

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

P1M initial value = xx00 0000

0C1H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
P1M	0	0	0	0	P13M	P12M	P11M	P10M
	-	-	-	-	R/W	R/W	R/W	R/W

P10M~P13M: P1.0~P1.3 I/O direction control bit. 0 = input mode, 1 = output mode.

P4M initial value = 0000 0000

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

P40M~P47M: P4.0~P4.7 I/O direction control bit. 0 = input mode, 1 = output mode.

P5M initial value = 0000 0000

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

P50M~P54M: P5.0~P5.4 I/O direction control bit. 0 = input mode, 1 = output mode.

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.

- ***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     P4M
CLR     P5M


MOV     A, #0FFH        ; Set all ports to be output mode.
B0MOV   P1M, A
B0MOV   P4M, A
B0MOV   P5M, A

B0BCLR  P1M.1           ; Set P1.1 to be input mode.

B0BSET  P1M.1           ; Set P1.1 to be output mode.
```

THE P0.3~P0.5 DISCRIPTION

The P0.3~P0.5 are input pins and shared with COM0~COM2 LCD driver pins. These input pins must work without LCD. To set the LCD code option to disable mode, the P0.3~P0.5's input function will be enabled. Input data be store in P0 data register.

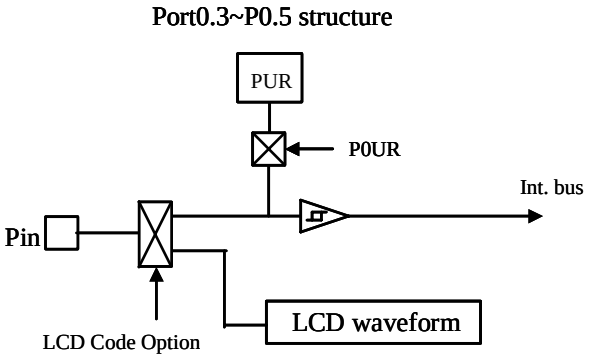


Figure 11-2. P0.3~P0.5 Block Diagram

➲ **Example: Enable P0.3~P0.5 input function.**

Step1: Disable the LCD controller of Code Option.

Step2: Disable the LCD driver by LCD control bit.

```
B0BCLR    FLENB    ; Disable LCD driver.
```

Step3: Now the P0.3~P0.5 general input function is enable. User can read P0.3~P0.5 input value using read instruction.

```
B0MOV     A,P0     ; The bit3~bit5 are P0.3~P0.5 value.
```

THE PORT2 DISCRIPTION

The port 2 is output only pins and shared with SEG16~SEG23 LCD driver pins. The port2 must works without LCD. To set the LCD code option to disable mode, the port 2 output function will be enabled. Output data from P2 is by P2 data register.

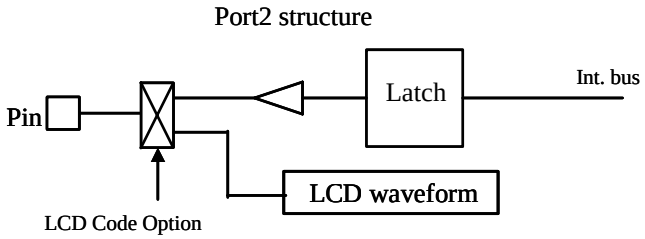


Figure 11-3. Port 2 Block Diagram

➲ **Example: Enable PORT 2 output function.**

Step1: Disable the LCD controller of Code Option.

Step2: Disable the LCD driver by LCD control bit.

```
B0BCLR      FLENB       ; Disable LCD driver.
```

Step3: Now the PORT 2 general output function is enable. User can output data by PORT 2.

```
B0MOV       A, BUF      ; Output BUF value to Port 2.
B0MOV       P2, A
```

➢ ***NOTE: The PORT 2 register (P2) is write-only register and doesn't support bit set/clear/test instruction.***

```
B0BSET      P2.0        ; ERROR!! Bit set instruction.
B0BCLR      P2.0        ; ERROR!! Bit clear instruction.
B0BTS1      P2.0        ; ERROR!! Bit 1 test instruction.
B0BTS0      P2.0        ; ERROR!! Bit 0 test instruction.
```

THE PORT3 DISCRIPTION

The port 3 is input only pins and shared with SEG8~SEG15 LCD driver pins. The port3 can work with LCD working. To set the bit1 of OPTION register (P3LCD) be “1”, the port 3 input function will be enabled. The port3 input is with pull-up resistors.

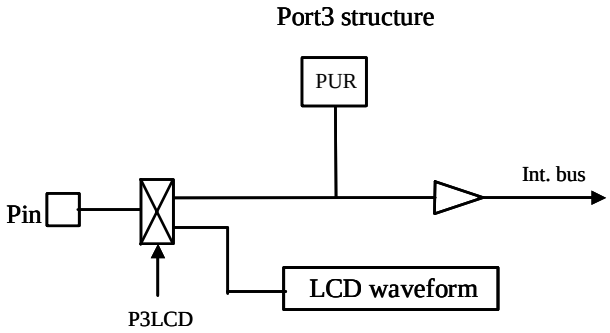


Figure 11-4. Port 3 Block Diagram

OPTION Register

OPTION initial value = xxxx xx00

088H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
OPTION	-	-	-	-	-	-	P3LCD	RLCK
	-	-	-	-	-	-	R/W	R/W

P3LCD: P3 input function control bit. 0 = LCD driver pin, 1 = input port.

➲ **Example: Enable PORT 3 output function.**

Step1: Enable the P3 general input function by P3LCD control bit (“1”).

```
B0BSET      FP3LCD      ; Enable P3 general input function.
```

Step2: Now the PORT 3 general input function is enable. User can read data by PORT 3.

```
B0MOV       A, P3       ; Read P3 value.
```

➢ ***NOTE: The PORT 3 register (P3) is read-only register and doesn’t support bit set/clear instruction.***

```
B0BSET      P3.0        ; ERROR!! Bit set instruction.
B0BCLR      P3.0        ; ERROR!! Bit clear instruction.
```

THE PORT6 DISCRIPTION

The port 6 is I/O pins and shared with SEG0~SEG7 LCD driver pins. The port6 can work with LCD working. To set the bit1 of LCDM register (P6HSEG) be “1”, the high-nibble of port 6 I/O function will be enabled. To set the bit0 of LCDM register (P6LSEG) be “1”, the low-nibble of port 6 I/O function will be enabled. Setting the port6 direction is by P6M register.
When Port6 is set to input mode, a pull-up register is connected automatic.

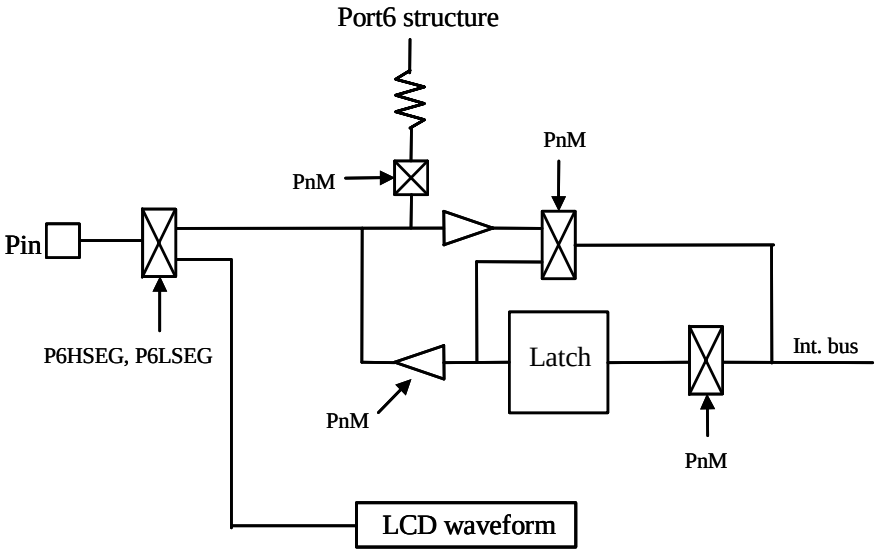


Figure 11-5. Port 6 Block Diagram

LCDM Register

LCDM register initial value = xx0x 0x00

0CBH	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
LCDM	-	-	BLANK	-	LENB	-	P6HSEG	P6LSEG
	-	-	R/W	-	R/W	-	R/W	R/W

P6LSEG : The lower 4 pins of Port6 control bit. 0 = Segment pins, 1 = general purpose I/O pins.
P6HSEG : The higher 4 pins of port6 control bit. 0 = Segment pins, 1 = general purpose I/O pins.

➲ **Example: Enable PORT 6 general input function.**

```
; Set P6 as input port
        B0BSET      FP6HSEG         ; Set P6 high-nibble to be I/O mode.
        B0BSET      FP6LSEG         ; Set P6 low-nibble to be I/O mode.

        CLR         P6M             ; Set P6 to be input

        MOV         A,#0FFH
        B0MOV       A,P6            ; Read data from P6


        .
        .
```

➲ **Example: Enable PORT 6 general output function.**

```
; Set P6 as output port
        B0BSET      FP6HSEG         ; Set P6 high-nibble to be I/O mode.
        B0BSET      FP6LSEG         ; Set P6 low-nibble to be I/O mode.

        MOV         A,#0FFH
        B0MOV       P6M,A           ; Set P6 to be output

        MOV         A,#0FFH
        B0MOV       P6,A            ; Output data to P6


        .
        .
```

I/O PORT DATA REGISTER

P0 initial value = xxx0 0000

0D0H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
P0	-	-	P05	P04	P03	P02	P01	P00
	-	-	R	R	R	R	R	R

P1 initial value = xxxx 0000

0D1H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
P1	-	-	-	-	P13	P12	P11	P10
	-	-	-	-	R/W	R/W	R/W	R/W

P2 initial value = 0000 0000

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

P3 initial value = 0000 0000

0D3H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
P3	P37	P36	P35	P34	P33	P32	P31	P30
	R	R	R	R	R	R	R	R

P4 initial value = 0000 0000

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

P5 initial value = xxx0 0000

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

P6 initial value = 0000 0000

0D6H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
P6	P67	P66	P65	P64	P63	P62	P61	P60
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

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

```
B0MOV   A, P0   ; Read data from Port 0
B0MOV   A, P1   ; Read data from Port 1
B0MOV   A, P4   ; Read data from Port 4
B0MOV   A, P5   ; Read data from Port 5
```

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

```
MOV     A, #55H ; Write data 55H to Port 1, Port 2, Port4 and Port 5
B0MOV   P1, A
B0MOV   P2, A
B0MOV   P4, A
B0MOV   P5, A
```

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

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

B0BCLR  P2.3    ; Set P2.3 and P5.1 to be “0”.
B0BCLR  P5.1
```

➲ **Example: Port bit test.**

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

12 LCD DRIVER

There are 3 common pins and 24 segment pin in the SN8P1808. The LCD scan timing is 1/3 duty and 1/2 bias structure to yield 72 dots LCD driver. Of these pins, eight segment pins are shared with Port 6 and P6/SEG functions can be selected by programming LCDM register.

LCDM REGISTER

LCDM register initial value = xx0x 0x00

0CBH	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
LCDM	-	-	BLANK	-	LENB	-	P6HSEG	P6LSEG
	-	-	R/W	-	R/W	-	R/W	R/W

P6LSEG : The lower 4 pins of Port6 control bit. 0 = Segment pins, 1 = general purpose I/O pins.
P6HSEG : The higher 4 pins of port6 control bit. 0 = Segment pins, 1 = general purpose I/O pins.
LENB : LCD driver enables control bit. 0 = disable, 1 = enable.
BLANK : LCD blanking control bit. 0 = normal display, 1 = all of the LCD dots off.

In following diagram, in order to get suitable contrast level of LCD panel, users can add external resistor to bias pin (V1, V2) to adjust bias voltage and LCD drive current. Too much or less current makes the LCD to bring remnant images. In normal condition, the external bias resistor value is 100K ohm. Users can connect a resistor between VLCD and VDD to adjust the voltage level at VLCD pin or just connect VLCD to VDD directly.

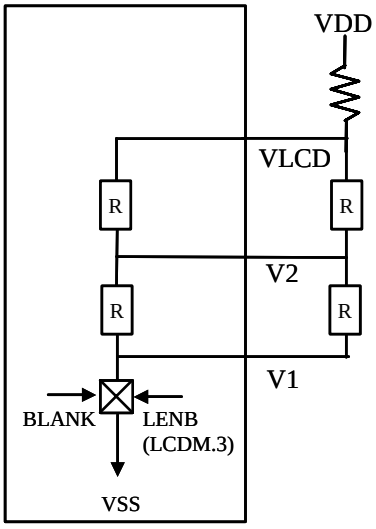
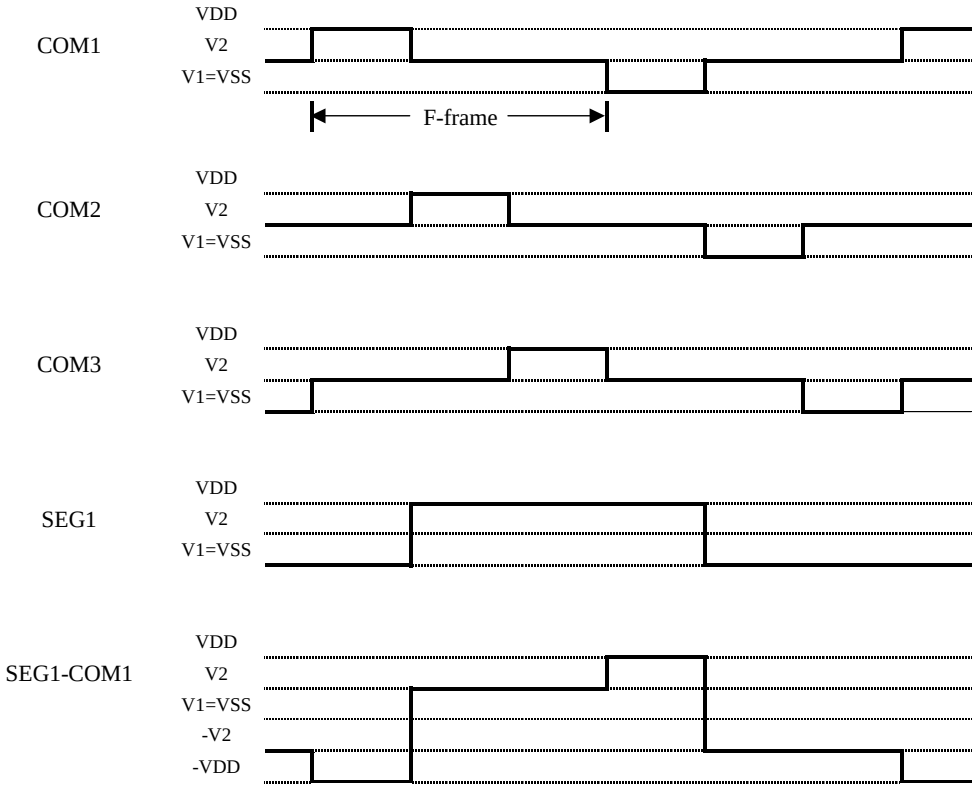


Figure 12-1. Adjust Circuit of LCD contrast

LCD TIMING

F-frame = External Low clock / 384

Ex. External low clock is 32768Hz. The F-frame is 32768Hz/384 = **85.3Hz**.



Note: The clock source of LCD driver is external low clock.

LCD RAM LOCATION

RAM bank 15's address vs. Common/Segment pin location

	COM0	COM1	COM2	-	-	-	-	-
SEG 0	00H.0	00H.1	00H.2	-	-	-	-	-
SEG 1	01H.0	01H.1	01H.2	-	-	-	-	-
SEG 2	02H.0	02H.1	02H.2	-	-	-	-	-
SEG 3	03H.0	03H.1	03H.2	-	-	-	-	-
-	-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-	-
SEG 14	0EH.0	0EH.1	0EH.2	-	-	-	-	-
SEG 15	0FH.0	0FH.1	0FH.2	-	-	-	-	-
SEG 16	10H.0	10H.1	10H.2	-	-	-	-	-
SEG 17	11H.0	11H.1	11H.2	-	-	-	-	-
-	-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-	-
SEG 21	15H.0	15H.1	15H.2	-	-	-	-	-
SEG 22	16H.0	16H.1	16H.2	-	-	-	-	-
SEG 23	17H.0	17H.1	17H.2	-	-	-	-	-
-	-	-	-	-	-	-	-	-

➲ **Example: Enable LCD function.**

Step1: Enable the LCD controller of Code Option to enable COM0~COM2, SEG16~SEG23.

Step2: Enable the segment pin shared with PORT3 and PORT 6.

```
B0BCLR      FP3LCD      ; Enable SEG8~SEG15 shared with PORT 3.
B0BCLR      FP6HSEG     ; Enable SEG4~SEG7 shared with P6.4~P6.7.
B0BCLR      FP6LSEG     ; Enable SEG0~SEG3 shared with P6.0~P6.3.
```

Step3: Now all LCD pins are enabled. Set the LCD control bit (LENB) and program LCD RAM to display LCD panel.

```
B0BSET      FLENB       ; LCD driver.
.
.
```

13 8-CHANNEL ANALOG TO DIGITAL CONVERTER

OVERVIEW

This analog to digital converter of SN8P1800 has 8-input sources with up to 4096-step resolution to transfer analog signal into 12-bits digital data. The sequence of ADC operation is to select input source (AIN0 ~ AIN7) at first, then set GCHS and ADS bit to "1" to start conversion. When the conversion is complete, the ADC circuit will set EOC bit to "1" and final value output in ADB register. This ADC circuit can select between 8-bit and 12-bit resolution operation by programming ADLEN bit in ADR register.

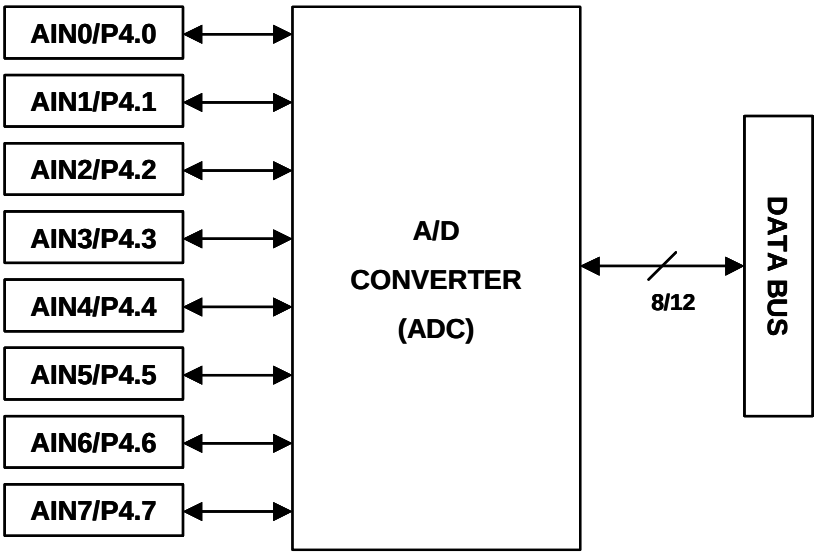


Figure 13-1. AD Converter Function Diagram

- ***Note: For 8-bit resolution the conversion time is 12 steps.***
  For 12-bit resolution the conversion time is 16 steps.
- ***Note: The analog input level must be between the AVREFH and AVSS.***
- ***Note: The AVREFH level must be between the AVDD and AVSS.***

ADM REGISTER

ADM initial value = 0000 x000

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

CHS2, 1, 0: ADC input channels select bit. 000 = AIN0, 001 = AIN1, 010 = AIN2, 011 = AIN3, .. , 111 = AIN7.
GCHS: Global channel select bit. 0 = To disable AIN channel, 1 = To enable AIN channel.
EOC: ADC status bit. 0 = Progressing, 1 = End of converting and reset ADENB bit.
ADS: ADC start bit. 0 = stop, 1 = starting.
ADENB: ADC control bit. 0 = disable, 1 = enable.

ADR REGISTERS

ADR initial value = x000 0000

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

ADBn: ADC data buffer. ADB11~ADB4 bits for 8-bit ADC. ADB11~ADB0 bits for 12-bit ADC.
ADLEN: ADC's resolution select bits. 0 = 8-bit, 1 = 12-bit.
ADCKS0, ADCKS1:

ADCKS1	ADCKS0	ADC Clock Source	Note
0	0	Fcpu/4	Both validate in Normal mode and Slow mode
0	1	Fcpu/2	Both validate in Normal mode and Slow mode
1	0	Fhosc	Only validate in Normal mode
1	1	Fhosc/2	Only validate in Normal mode

ADB REGISTERS

ADB initial value = xxxx xxxx

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

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

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

AIN n	ADB11	ADB10	ADB9	ADB8	ADB7	ADB6	ADB5	ADB4	ADB3	ADB2	ADB1	ADB0
0/4096*AVREFH	0	0	0	0	0	0	0	0	0	0	0	0
1/4096*AVREFH	0	0	0	0	0	0	0	0	0	0	0	1
.	.	.	.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.	.	.	.
4094/4096*AVREFH	1	1	1	1	1	1	1	1	1	1	1	0
4095/4096*AVREFH	1	1	1	1	1	1	1	1	1	1	1	1

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

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

ADC CONVERTING TIME

12-bit ADC conversion time = 1/(ADC clock /4)*16 sec

8-bit ADC conversion time = 1/(ADC clock /4)*12 sec

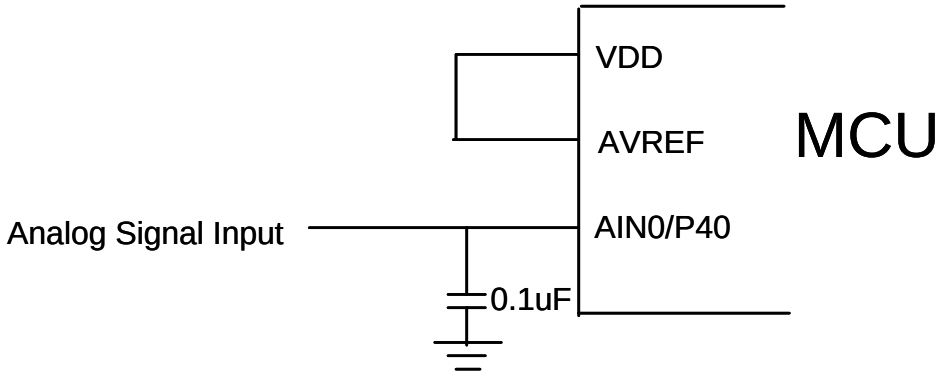
High clock (fosc) is @3.58MHz

ADLEN	ADCKS1	ADCKS0	ADC Clock	ADC conversion time
0 (8-bit)	0	0	fcpu/4	1/((3.58MHz/4)/4/4)*12 = 214.5 us
	0	1	fcpu/2	1/((3.58MHz/4)/2/4)*12 = 107.3 us
	1	0	fhosc	1/(3.58MHz/4)*12 = 13.4 us
	1	1	fhosc/2	1/(3.58MHz/2/4)*12 = 26.8 us
1 (12-bit)	0	0	fcpu/4	1/((3.58MHz/4)/4/4)*16 = 286 us
	0	1	fcpu/2	1/((3.58MHz/4)/2/4)*16 = 143 us
	1	0	fhosc	1/(3.58MHz/4)*16 = 17.9 us
	1	1	fhosc/2	1/(3.58MHz/2/4)*16 = 35.8 us

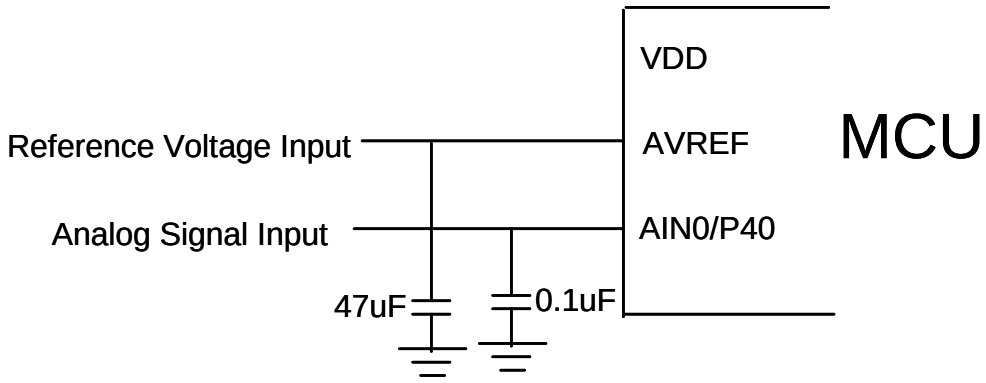
➲ **Example : To set AIN0 ~ AIN1 for ADC input and executing 12-bit ADC**

```
ADC0:
        MOV     A, #60H
        B0MOV   ADR, A          ; To set 12-bit ADC and ADC clock = Fosc.
        MOV     A,#90H
        B0MOV   ADM,A           ; To enable ADC and set AIN0 input
        B0BSET  FADS            ; To start conversion
WADC0:
        B0BTS1  FEOC            ; To skip, if end of converting =1
        JMP     WADC0           ; else, jump to WADC0
        B0MOV   A,ADB           ; To get AIN0 input data
ADC1:
        MOV     A,#91H          ;
        B0MOV   ADM,A           ; To enable ADC and set AIN1 input
        B0BSET  FADS            ; To start conversion
        .       .               .
QEXADC:
        B0BCLR  FGCHS           ; To release AINx input channel
```

ADC CIRCUIT



AVREFH is connected to VDD.



AVREFH is connected to external AD reference voltage.

Figure 13-2. The AINx and AVREFH Circuit of AD Converter

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

14 CODING ISSUE

TEMPLATE CODE

```
;*************************************************************************
; FILENAME  : TEMPLATE.ASM
; AUTHOR    : SONiX
; PURPOSE   : Template Code for SN8X180X
; REVISION  : 09/01/2002 V1.0   First issue
;*************************************************************************
;*  (c) Copyright 2002, SONiX TECHNOLOGY CO., LTD.
;*************************************************************************
CHIP    SN8P1808                    ; Select the CHIP

;-------------------------------------------------------------------------
;                           Include Files
;-------------------------------------------------------------------------
.nolist                             ; do not list the macro file

    INCLUDESTD  MACRO1.H
    INCLUDESTD  MACRO2.H
    INCLUDESTD  MACRO3.H

.list                               ; Enable the listing function

;-------------------------------------------------------------------------
;                           Constants Definition
;-------------------------------------------------------------------------
;   ONE         EQU     1

;-------------------------------------------------------------------------
;                           Variables Definition
;-------------------------------------------------------------------------
.DATA

                org     0h          ;Bank 0 data section start from RAM address 0x000
    Wk00B0      DS      1           ;Temporary buffer for main loop
    Iwk00B0     DS      1           ;Temporary buffer for ISR
    AccBuf      DS      1           ;Accumulater buffer
    PflagBuf    DS      1           ;PFLAG buffer

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

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



;-------------------------------------------------------------------------
```

```
;                           Code section
;-------------------------------------------------------------------------------

.CODE

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

    ORG         10h
;-------------------------------------------------------------------------------
;   Program reset section
;-------------------------------------------------------------------------------
Reset:
    mov         A,#07Fh             ;Initial stack pointer and
    b0mov       STKP,A              ;disable global interrupt
    b0mov       PFLAG,#00h          ;pflag = x,x,x,x,x,c,dc,z
    b0mov       RBANK,#00h          ;Set initial RAM bank in bank 0
    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 and system registers
;-----------------------------------
    b0xch       A,AccBuf            ;B0xch instruction do not change C,Z flag
    push                            ; Save 80h ~ 87h system

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

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

    ;If necessary, insert another interrupt checking here

IntTc0Chk:
    b0bts1      FTC0IEN
    jmp         IsrExit             ;Suppose TC0 is the last interrupt which you
    b0bts0      FTC0IRQ             ;want to check
    jmp         TC0isr

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

IsrExit:

    pop                             ; Restore 80h ~ 87h system registers
    b0xch       A,AccBuf            ;B0xch instruction do not change C,Z flag

    reti                            ;Exit the interrupt routine



;-------------------------------------------------------------------------------
```

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

    ;Process P0.0 external interrupt here

    jmp         IsrExit

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

    ;Process TC0 timer interrupt here

    jmp         IsrExit

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

    ret

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

ClrRAM:

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

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

; RAM Bank 1
    mov         A,#1
    b0mov       Y,A                 ;Select bank 1
    b0mov       Z,#0x7f             ;Set @YZ address from 17fh

ClrRAM20:
    clr         @YZ                 ;Clear @YZ content
    decms       Z                   ;z = z - 1 , skip next if z=0
    jmp         ClrRAM20
    clr         @YZ                 ;Clear address 0x100
    ret

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

CHIP DECLARATION IN ASSEMBLER

Assembler	OTP Device Part Number	MASK Device Part Number
CHIP SN8P1808	SN8P1808	N/A

PROGRAM CHECK LIST

Item	Description
Pull-up Resister	Use @SET_PUR macro or PnUR register to enable or disable on-chip pull-up resisters. Refer I/O port chapter for detailed information.
Undefined Bits	All bits those are marked as “0” (undefined bits) in system registers should be set “0” to avoid unpredicted system errors.
ADC	Set ADC input pin I/O direction as input mode and disable pull-up resister of ADC input pin
SIO Master Mode	Set SCK (P5.0) and SO (P5.2) pin as output mode. Set SI (P5.1) pin as input mode.
SIO Slave Mode	Set SO (P5.2) pin as output mode. Set SCK (P5.0) and SI (P5.1) pin as input mode.
PWM0	Set PWM0 (P5.4) pin as output mode.
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 pull-up or pull-down in input mode, or be set as low in output mode to save current consumption.
Sleep Mode	Enable on-chip pull-up 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

15 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
	MUL	A,M	R, A ← A * M, The LB of product stored in Acc and HB stored in R register. ZF affected by Acc.	-	-	√	2
	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	PUSH		To push working registers (080H~087H) into buffers	-	-	-	1
C	POP		To pop working registers (080H~087H) from buffers	√	√	√	1
	NOP		No operation	-	-	-	1

Table 15-1. Instruction Set Table of SN8P1800

Note: Any instruction that read/write from 0SCM, will add an extra cycle.

16 ELECTRICAL CHARACTERISTIC

ABSOLUTE MAXIMUM RATING

(All of the voltages referenced to Vss)

Supply voltage (Vdd)..... - 0.3V ~ 6.0V
Input in voltage (Vin).....Vss - 0.2V ~ Vdd + 0.2V
Operating ambient temperature (Topr).....-20°C ~ + 70°C
Storage ambient temperature (Tstor).....-30°C ~ + 125°C
Power consumption (Pc).....500 mW

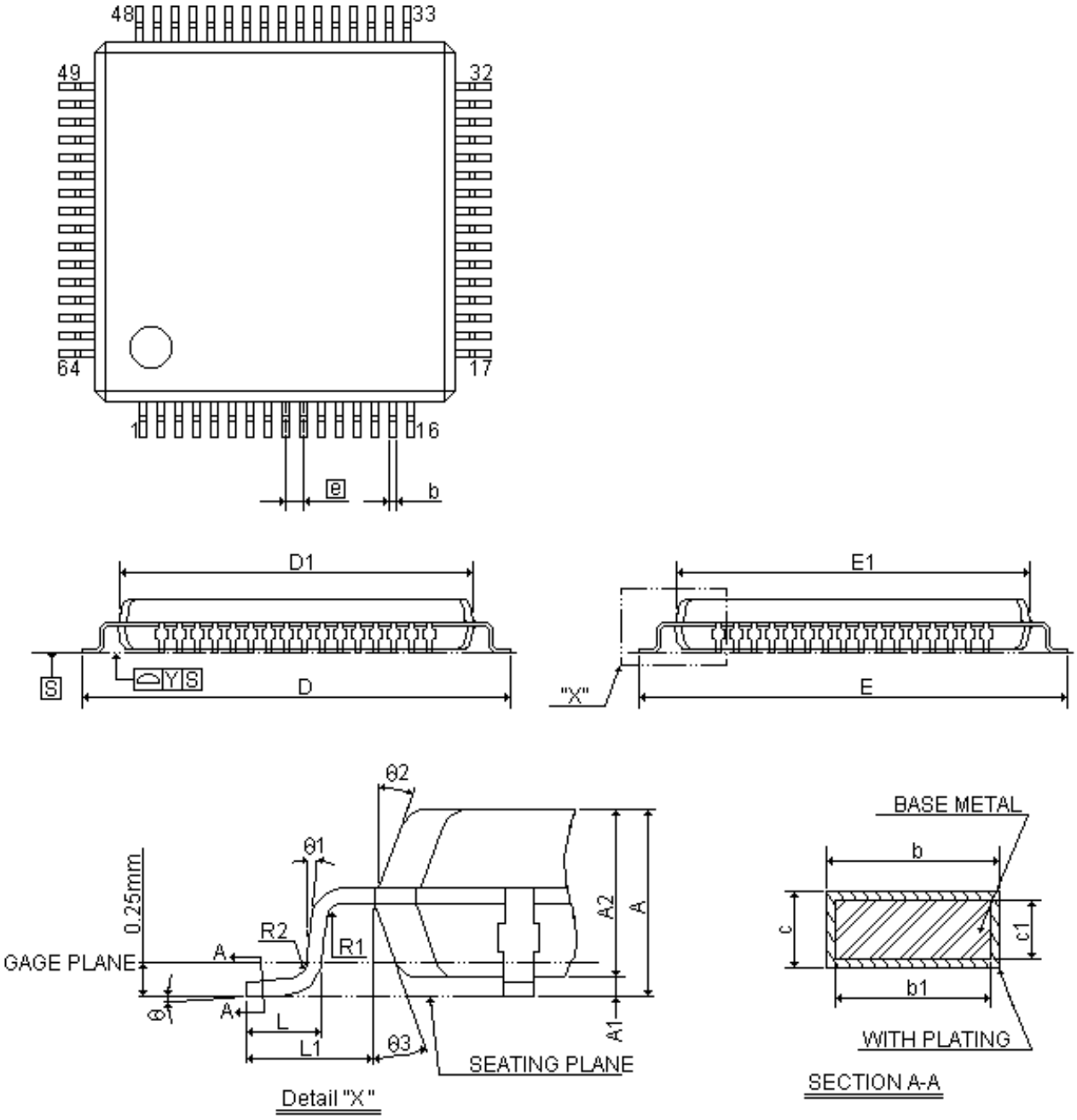
STANDARD ELECTRICAL CHARACTERISTIC

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

PARAMETER	SYM.	DESCRIPTION		MIN.	TYP.	MAX.	UNIT
Operating voltage	Vdd	Normal mode, Vpp = Vdd		2.4	5.0	5.5	V
		Programming mode, Vpp = 12.5V		4.5	5.0	5.5	
RAM Data Retention voltage	Vdr			-	1.5	-	V
Internal POR	Vpor	Vdd rise rate to ensure internal power-on reset		-	0.05	-	V/ms
Input Low Voltage	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	-	0.2Vdd	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		-	-	2	uA
I/O port pull-up resistor	Rup	Vin = Vss , Vdd = 5V		-	100	-	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		-	12	-	mA
sink current	IoL	Vop = Vss + 0.5V		-	15	-	
Port2 output source current	IoH	Vop = Vdd - 0.5V		-	12	-	mA
sink current	IoL	Vop = Vss + 0.5V		-	15	-	
Port4 output source current	IoH	Vop = Vdd - 0.5V		-	12	-	mA
sink current	IoL	Vop = Vss + 0.5V		-	15	-	
Port5 output source current	IoH	Vop = Vdd - 0.5V		-	12	-	mA
sink current	IoL	Vop = Vss + 0.5V		-	15	-	
Port6 output source current	IoH	Vop = Vdd - 0.5V		-	12	-	mA
sink current	IoL	Vop = Vss + 0.5V		-	15	-	
INTn trigger pulse width	Tint0	INT0 ~ INT2 interrupt request pulse width		2/fcpu	-	-	cycle
AVREFH input voltage	Varfh	Vdd = 5.0V		Varfl+1.2V	-	Vdd	V
AVREFL input voltage	Varfl	Vdd = 5.0V		Vss	-	Varfh–1.2V	V
AIN0 ~ AIN7 input voltage	Vani			Varfl	-	Varfh	V
COM, SEG pin Output voltage	V3/3			-	1	-	
	V2/3			-	2/3	-	VLCD
	V1/3			-	1/3	-	
LCD bias dividing resistor	Zlcd	Each section resistor, Vdd = 5.0V		-	1.2	-	MΩ
LCD frame frequency	flcdm1			-	64	-	Hz
LCD Voltage	Vlcd			-	-	Vdd	V
Supply Current (Disable ADC and LVD)	Idd1	Run Mode	Vdd= 5V 4Mhz	-	7	15	mA
			Vdd= 3V 4Mhz	-	1.5	3	mA
			Vdd= 3V 32768Hz	-	50	100	uA
	Idd2	Slow Mode (High clock stop LCD off)	Vdd= 5V 32768Hz	-	100	200	uA
			Vdd= 3V 32768Hz	-	30	60	uA
	Idd3	Sleep mode	Vdd= 5V	-	10	20	uA
			Vdd= 3V	-	3	6	uA
	Idd4	Green Mode (High clock stop LCD off)	Vdd= 5V 32768Hz	-	40	80	uA
			Vdd= 3V 32768Hz	-	10	20	uA
ADC current consumption	IADC	Vdd=5.0V		-	0.6	1	mA
		Vdd=3.0V		-	0.4	0.8	mA
LVD Detect Voltage	Vdet	Low voltage detect level		-	2.4	-	V
Voltage detector current	Ivdet	LVD enable operating current		-	100	180	uA

17 PACKAGE INFORMATION

LQFP64:



SYMBLE	DIMENSION (MM)			DIMENSION (MIL)		
	MIN.	NOM.	MAX.	MIN.	NOM.	MAX.
A			1.60			63
A1	0.05	1.40	0.15	2	55	6
A2	1.36	0.22	1.45	35	9	57
b	0.17	0.22	0.27	7	8	11
b1	0.17		0.23	7		12
c	0.09		0.20	4		8
c1	0.09		0.16	4		6
D	12.00 BSC			472 BSC		
D1	10.00 BSC			394 BSC		
E	12.00 BSC			472 BSC		
E1	10.00 BSC			394 BSC		
[e]	0.50 BSC			20 BSC		
L	0.45	0.60	0.75	18	24	30
L1	1.00 REF			39 REF		
R1	0.08			3		
R2	0.08		0.20	3		8
Y			0.075			3
θ	0°	3.5°	7°	0°	3.5°	7°
θ 1	0°			0°		
θ 2	11°	12°	13°	11°	12°	13°
θ 3	11°	12°	13°	11°	12°	13°

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