SN8P2212 Series

USER'S MANUAL

SN8P2213
SN8P22121
SN8P2212

SONiX 8-Bit Micro-Controller

AMENDMENT HISTORY

Version	Date	Description
VER1.0	2007/2/13	version 1.0

Table of Content

1 PRODUCT OVERVIEW

1.1 FEATURES

- **Memory configuration**
  OTP ROM size: 4K x 16 bits.
  RAM size: 192 x 8 bits.

- **8 levels stack buffer**

- **I/O pin configuration**
  Bi-directional: P0, P1, P5.
  Wake-up: P0/P1 level change.
  Pull-up resistors: P0, P1, P5
  Open-drain: P1.0, P1.1, P5.0, P5.2.
  External interrupt: P0.0 controlled by PEDGE.

- **Full Speed USB 1.1.**
  Conforms to USB specification, Version 2.0.
  3.3V regulator output for USB D+ pin external 1.5k ohm pull-up resistor.
  Integrated USB transceiver.
  Supports 1 Full speed USB device address, 1 control endpoint (endpoint 0) and 2 interrupt endpoints (endpoint 1 and endpoint 2)

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

- **6 interrupt sources.**
  Five internal interrupts: T0, TC0, USB, SIO, Wakeup
  One external interrupt: INT0.

- **One SIO function for data transfer (Serial Peripheral Interface)**

- **Two 8-bit timer counters. (T0, TC0)**

- **On chip watchdog timer.**

- **Two system clocks.**
  External high clock: Crystal type 6MHz.
  Internal low clock: RC type 32KHz @5V.

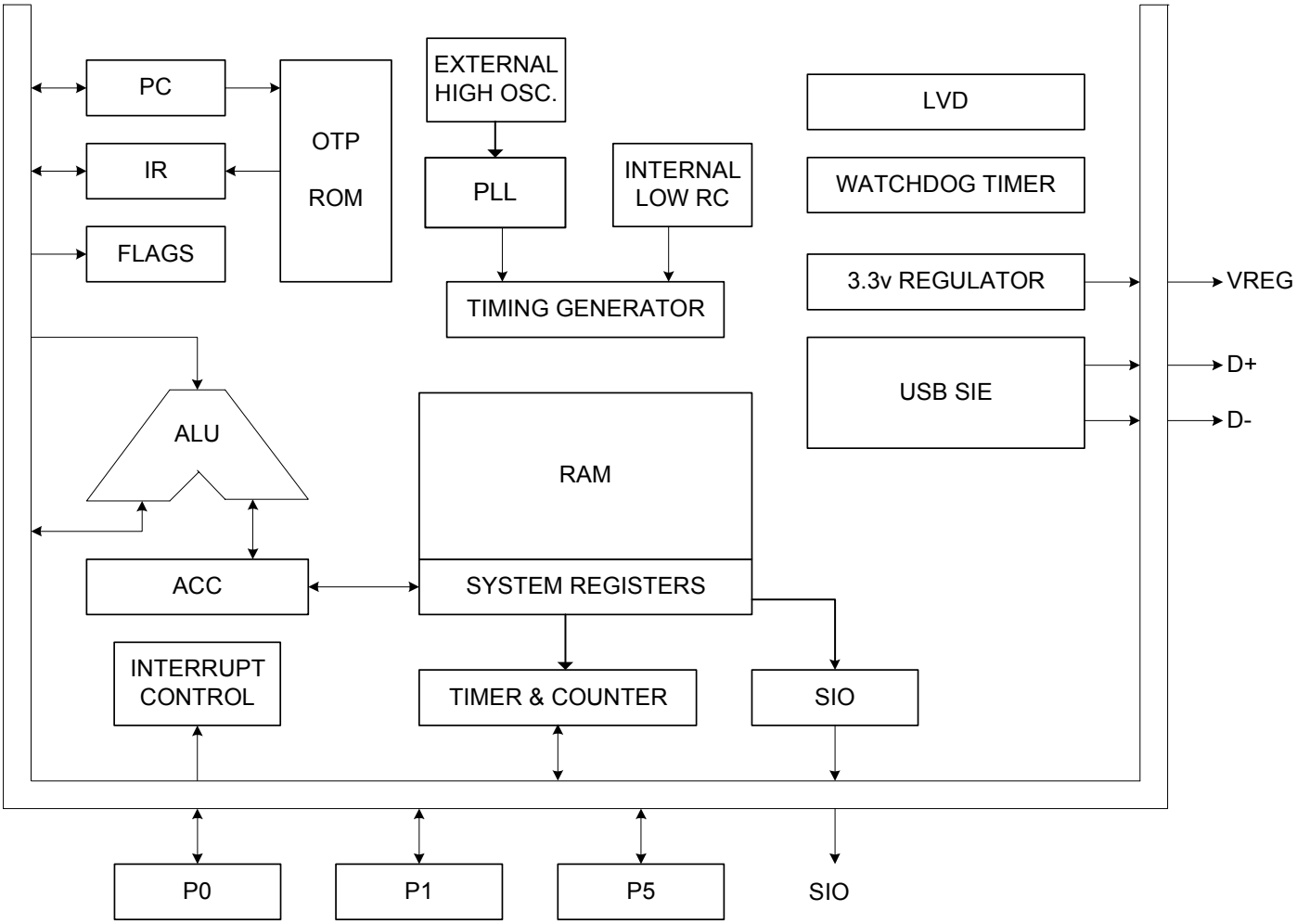
- **Four operating modes.**
  Normal mode: Both high and low clocks active.
  Slow mode: Low clock only.
  Sleep: Both high and low clocks stop.
  Green mode: Periodical wakeup by timer.

- **Package (Chip form support)**
  SK-DIP/P-DIP: 24/20/18
  SOP: 24/20/18
  SSOP: 24/20

☞ **Features Selection Table**

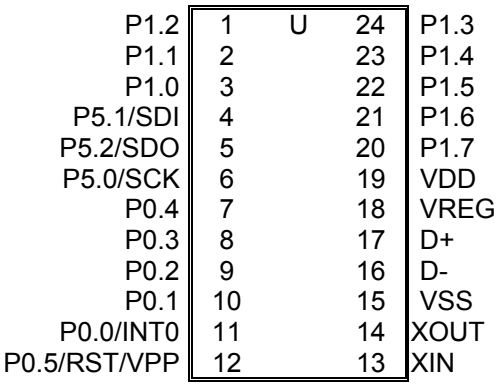
CHIP	ROM	RAM	STACK	TIMER		SIO	WAKE-UP PIN NO.	PACKAGE
				T0	TC0			
SN8P2213	4K*16	192*8	8	V	V	V	14	DIP24/SOP24/SSOP24
SN8P22121	4K*16	192*8	8	V	V	V	10	DIP20/SOP20/SSOP20
SN8P2212	4K*16	192*8	8	V	V	V	8	DIP18/SOP18

1.2 SYSTEM BLOCK DIAGRAM



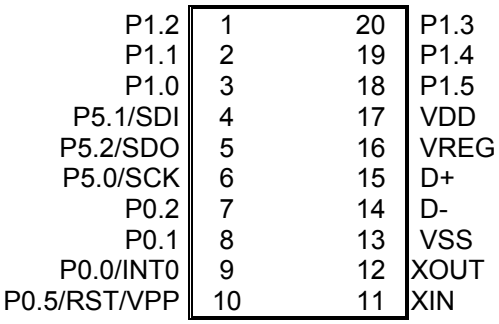
1.3 PIN ASSIGNMENT

SN8P2213K (SK-DIP 24 pins)
SN8P2213S (SOP 24 pins)
SN8P2213X (SSOP 24 pins)



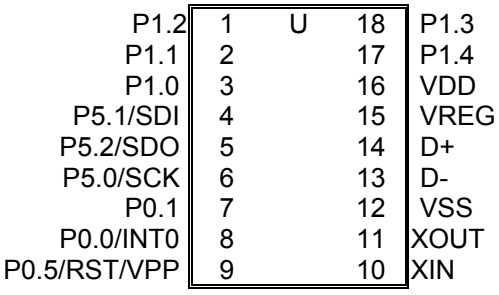
SN8P2213K
SN8P2213S
SN8P2213X

SN8P22121K (SK-DIP 20 pins)
SN8P22121S (SOP 20 pins)
SN8P22121X (SSOP 20 pins)



SN8P22121K
SN8P22121S
SN8P22121X

SN8P2212P (P-DIP 18 pins)
SN8P2212S (SOP 18 pins)



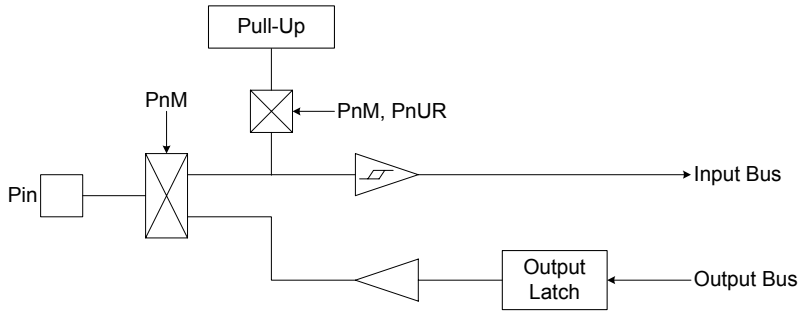
SN8P2212P
SN8P2212S

1.4 PIN DESCRIPTIONS

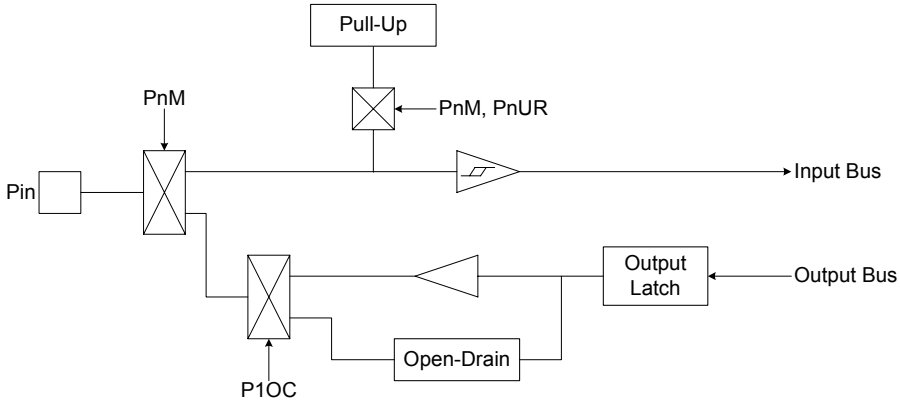
PIN NAME	TYPE	DESCRIPTION
VDD, VSS	P	Power supply input pins for digital circuit.
P0.0/INT0	I/O	P0.0: Port 0.0 bi-direction pin. Schmitt trigger structure and built-in pull-up resisters as input mode. Built wakeup function. INT0: External interrupt 0 input pin.
P0[4:0]	I/O	P0: Port 0 bi-direction pin. Schmitt trigger structure and built-in pull-up resisters as input mode. Built wakeup function.
P1.0	I/O	P1.0: Port 1.0 bi-direction pin. Schmitt trigger structure and built-in pull-up resisters as input mode. Open-Drain function controlled by “P1OC” register. Built wakeup function.
P1.1	I/O	P1.1: Port 1.1 bi-direction pin. Schmitt trigger structure and built-in pull-up resisters as input mode. Open-Drain function controlled by “P1OC” register. Built wakeup function.
XOUT	I/O	XOUT: Oscillator output pin while external oscillator enable.
XIN	I/O	XIN: Oscillator input pin while external oscillator enable.
P0.5/RST/VPP	I, P	RST is system external reset input pin under Ext_RST mode. Schmitt trigger structure, active “low”, normal stay to “high”. P0.5 is input only pin without pull-up resistor under P0.5 mode. Built wakeup function. OTP 12.3V power input pin in programming mode.
P1[7:2]	I/O	P1: Port 1 bi-direction pin. Schmitt trigger structure and built-in pull-up resisters as input mode.
P5.0/SCK	I/O	P5.0: Port 5.0 bi-direction pin. Schmitt trigger structure and built-in pull-up resisters as input mode. SCK: SIO output clock pin. Open-Drain function controlled by “P1OC” register.
P5.1/SDI	I/O	P5.1: Port 5.1 bi-direction pin. Schmitt trigger structure and built-in pull-up resisters as input mode. SDI: SIO data input pin.
P5.2/SDO	I/O	P5.2: Port 5.2 bi-direction pin. Schmitt trigger structure and built-in pull-up resisters as input mode. SCK: SIO data output pin. Open-Drain function controlled by “P1OC” register.
VREG	O	3.3V voltage output from USB 3.3V regulator.
D+, D-	I/O	USB differential data line.

1.5 PIN CIRCUIT DIAGRAMS

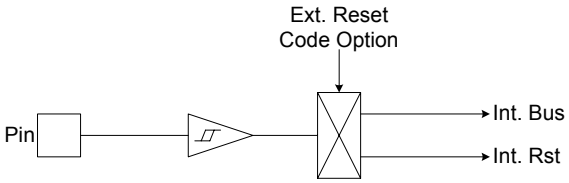
Port 0, 1, 5 structures:



Port 1.0, Port 1.1 structure:



Port 0.5 structure:

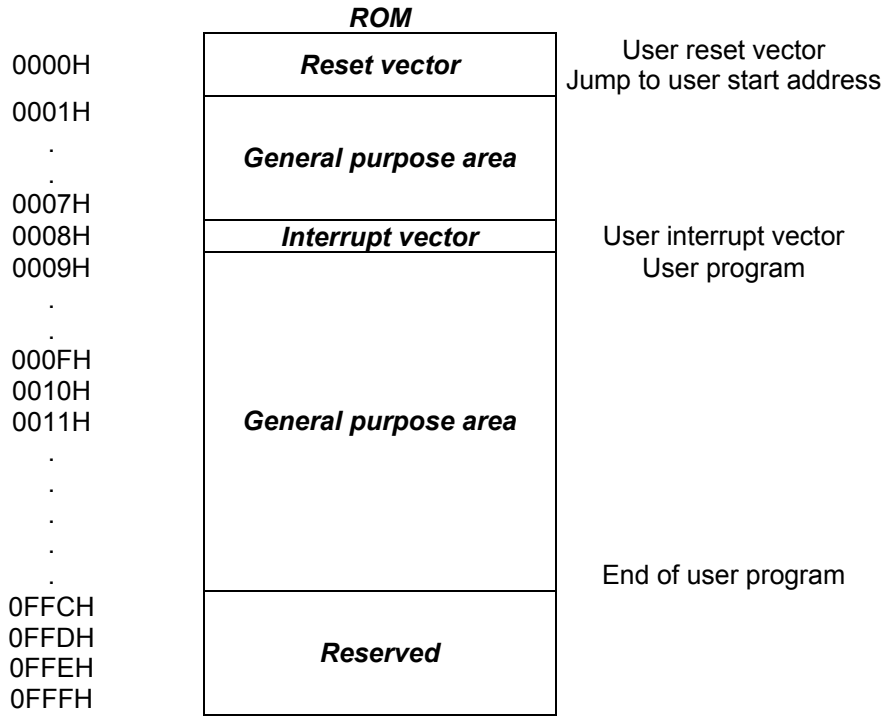


2 CENTRAL PROCESSOR UNIT (CPU)

2.1 MEMORY MAP

2.1.1 PROGRAM MEMORY (ROM)

☞ ***4K words ROM***



2.1.1.1 RESET VECTOR (0000H)

A one-word vector address area is used to execute system reset.

- ☞ **Power On Reset (NT0=1, NPD=0).**
- ☞ **Watchdog Reset (NT0=0, NPD=0).**
- ☞ **External Reset (NT0=1, NPD=1).**

After power on reset, external 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. It is easy to know reset status from NT0, NPD flags of PFLAG register. The following example shows the way to define the reset vector in the program memory.

➢ **Example: Defining Reset Vector**

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

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

        ENDP                ; End of program
```

2.1.1.2 INTERRUPT VECTOR (0008H)

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

✸ ***Note:"PUSH", "POP" instructions save and load ACC/PFLAG without (NT0, NPD). PUSH/POP buffer is a unique buffer and only one level.***

➢ **Example: Defining Interrupt Vector. The interrupt service routine is following ORG 8.**

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

                ORG     8               ; Interrupt vector.
                PUSH                    ; Save ACC and PFLAG register to buffers.
                …
                …
                POP                     ; Load ACC and PFLAG register from buffers.
                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: Defining Interrupt Vector. The interrupt service routine is following user program.**

```
.CODE
                ORG     0           ; 0000H
                JMP     START       ; Jump to user program address.
                …
                ORG     8           ; Interrupt vector.
                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.
                PUSH                ; Save ACC and PFLAG register to buffers.
                …
                …
                POP                 ; Load ACC and PFLAG register from buffers.
                RETI                ; End of interrupt service routine.
                …

                ENDP                ; End of program.
```

✸ ***Note: It is easy to understand 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 starts from the beginning.***
2. ***The address 0008H is interrupt vector.***
3. ***User's program is a loop routine for main purpose application.***

2.1.1.3 LOOK-UP TABLE DESCRIPTION

In the ROM's data lookup function, Y register is pointed to middle byte address (bit 8~bit 15) and Z register is pointed to low byte address (bit 0~bit 7) of ROM. After MOVC instruction executed, the low-byte data 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              ; Z is not overflow.
                INCMS   Y               ; Z overflow (FFH → 00), → Y=Y+1
                NOP                     ;
                                        ;
@@:             MOVC                    ; To lookup data, R = 51H, ACC = 05H.
                …                       ;
TABLE1:         DW      0035H           ; To define a word (16 bits) data.
                DW      5105H
                DW      2012H
                …
```

✸ ***Note: The Y register will not increase automatically when Z register crosses boundary from 0xFF to 0x00. Therefore, user must take care such situation to avoid loop-up table errors. If Z register overflows, Y register must be added one. The following INC_YZ macro shows a simple method to process Y and Z registers automatically.***

➢ **Example: INC_YZ macro.**

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

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

➢ **Example: Modify above example by "INC_YZ" macro.**

```
        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

        INC_YZ                  ; Increment the index address for next address.
                                ;
@@:     MOVC                    ; To lookup data, R = 51H, ACC = 05H.
        …                       ;
TABLE1: DW      0035H           ; To define a word (16 bits) data.
        DW      5105H
        DW      2012H
        …
```

The other example of loop-up table is to add Y or Z index register by accumulator. Please be careful if "carry" happen.

➢ **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
        …
```

2.1.1.4 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. If PCL is overflow after PCL+ACC, PCH adds one automatically. The new program counter (PC) points to a series jump instructions as a listing table. It is easy to make a multi-jump program depends on the value of the accumulator (A).

✸ ***Note: PCH only support PC up counting result and doesn't support PC down counting. When PCL is carry after PCL+ACC, PCH adds one automatically. If PCL borrow after PCL–ACC, PCH keeps value and not change.***

➢ **Example: Jump table.**

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

        B0ADD   PCL, A          ; PCL = PCL + ACC, PCH + 1 when PCL overflow occurs.
        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
```

SONIX provides a macro for safe jump table function. This macro will check the ROM boundary and move the jump table to the right position automatically. The side effect of this macro maybe wastes some ROM size.

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

```
@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       ; 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 across a ROM boundary (0x00FF~0x0100), the "@JMP_A" macro will adjust the jump table routine begin from next RAM boundary (0x0100).

➢ **Example: "@JMP_A" operation.**

; Before compiling program.

```
ROM address
        B0MOV     A, BUF0       ; "BUF0" is from 0 to 4.
        @JMP_A    5             ; The number of the jump table listing is five.
0X00FD  JMP       A0POINT       ; ACC = 0, jump to A0POINT
0X00FE  JMP       A1POINT       ; ACC = 1, jump to A1POINT
0X00FF  JMP       A2POINT       ; ACC = 2, jump to A2POINT
0X0100  JMP       A3POINT       ; ACC = 3, jump to A3POINT
0X0101  JMP       A4POINT       ; ACC = 4, jump to A4POINT
```

; After compiling program.

```
ROM address
        B0MOV     A, BUF0       ; "BUF0" is from 0 to 4.
        @JMP_A    5             ; The number of the jump table listing is five.
0X0100  JMP       A0POINT       ; ACC = 0, jump to A0POINT
0X0101  JMP       A1POINT       ; ACC = 1, jump to A1POINT
0X0102  JMP       A2POINT       ; ACC = 2, jump to A2POINT
0X0103  JMP       A3POINT       ; ACC = 3, jump to A3POINT
0X0104  JMP       A4POINT       ; ACC = 4, jump to A4POINT
```

2.1.1.5 CHECKSUM CALCULATION

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

➢ **Example: The demo program shows how to 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 00H
                CLR     Z                       ; Set Z to 00H
@@:
                MOVC
                B0BSET  FC                      ; Clear C flag
                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.

Y_ADD_1:
                INCMS   Y                       ; Increase Y
                NOP
                JMP     @B                      ; Jump to checksum calculate
CHECKSUM_END:
                …
                …
END_USER_CODE:                                  ; Label of program end
```

2.1.2 CODE OPTION TABLE

Code Option	Content	Function Description
High_Clk	6MHz	6MHz crystal /resonator for external high clock oscillator.
Watch_Dog	Always_On	Watchdog timer is always on enable even in power down and green mode.
	Enable	Enable watchdog timer. Watchdog timer stops in power down mode and green mode.
	Disable	Disable Watchdog function.
Fcpu	Fhosc/1	Instruction cycle is 12 MHz clock.
	Fhosc/2	Instruction cycle is 6 MHz clock.
	Fhosc/4	Instruction cycle is 3 MHz clock.
Reset_Pin	Reset	Enable External reset pin.
	P05	Enable P0.5 input only without pull-up resister.
Security	Enable	Enable ROM code Security function.
	Disable	Disable ROM code Security function.

✷ ***Note: Fcpu code option is only available for High Clock. Fcpu of slow mode is Flosc/4.***

2.1.3 DATA MEMORY (RAM)

☞ ***192 X 8-bit RAM***

	Address	RAM location	
BANK 0	000h “ “ “ “ “ 07Fh	***General purpose area***	BANK 0
	080h “ “ “ “ “	***System register***	80h~FFh of Bank 0 store system registers (128 bytes).
	0FFh	***End of bank 0 area***	
BANK1	100h “ “ “ “ “ 13Fh	***General purpose area***	BANK1

2.1.4 SYSTEM REGISTER

2.1.4.1 SYSTEM REGISTER TABLE

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
8	-	-	R	Z	Y	-	PFLAG	RBANK	-	-	-	-	-	-	-	-
9	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
A	UDA	USTATUS	UCTRL	UE0E	UE1E	UE2E	UDP0	UDR0	UDP1	UDR1	UPID	EP1OUT_CNT	EP1OUT_CNT			
B	-	-	-	-	SIOM	SIOR	SIOB	-	P0M	-	-	-	-	-	-	PEDGE
C	P1W	P1M	-	-	-	P5M	-	-	INTRQ	INTEN	OSCM	-	WDTR	TC0R	PCL	PCH
D	P0	P1	-	-	-	P5	-	-	T0M	T0C	TC0M	TC0C	-	-	-	STKP
E	P0UR	P1UR	-	-	-	P5UR	-	@YZ	-	P1OC	-	-	-	-	-	-
F	STK7L	STK7H	STK6L	STK6H	STK5L	STK5H	STK4L	STK4H	STK3L	STK3H	STK2L	STK2H	STK1L	STK1H	STK0L	STK0H

2.1.4.2 SYSTEM REGISTER DESCRIPTION

R = Working register and ROM look-up data buffer.
PFLAG = ROM page and special flag register.
UDA = USB control register.
UDP0 = USB FIFO 0 address pointer.
UDP1 = USB FIFO 1 address pointer.
USTATUS = USB status register.
EP1OUT_CNT = USB endpoint 1 OUT token data byte counter
SIOM = SIO mode control register.
SIOB = SIO's data buffer.
PnM = Port n input/output mode register.
INTRQ = Interrupt request register.
OSCM = Oscillator mode register.
TC0R = TC0 auto-reload data buffer.
Pn = Port n data buffer.
T0C = T0 counting register.
TC0C = TC0 counting register.
PnUR = Port n pull-up resister control register.
P1OC = Port 1 open-drain control register.
P1W = Port 1 wakeup control register

Y, Z = Working, @YZ and ROM addressing register.
RBANK = RAM bank selection register.
UE0R~UE2R = Endpoint 0~2 control registers.
UDR0 = USB FIFO 0 data buffer by UDP0 point to.
UDR1 = USB FIFO 1 data buffer by UDP1 point to.
UPID = USB bus control register.
EP2OUT_CNT = USB endpoint 2 OUT token data byte counter
UCTRL= USB control register
SIOR = SIO's clock reload buffer
PEDGE = P0.0 edge direction register.
INTEN = Interrupt enable register.
WDTR = Watchdog timer clear register.
PCH, PCL = Program counter.
T0M = T0 mode register.
TC0M = TC0 mode register.
STKP = Stack pointer buffer.
@YZ = RAM YZ indirect addressing index pointer.
STK0~STK7 = Stack 0 ~ stack 7 buffer.

2.1.4.3 BIT DEFINITION of SYSTEM REGISTER

Address	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	R/W	Remarks
082H	RBIT7	RBIT6	RBIT5	RBIT4	RBIT3	RBIT2	RBIT1	RBIT0	R/W	R
083H	ZBIT7	ZBIT6	ZBIT5	ZBIT4	ZBIT3	ZBIT2	ZBIT1	ZBIT0	R/W	Z
084H	YBIT7	YBIT6	YBIT5	YBIT4	YBIT3	YBIT2	YBIT1	YBIT0	R/W	Y
086H	NT0	NPD				C	DC	Z	R/W	PFLAG
087H								RBNKS0	R/W	RBANK
0A0H	UDE	UDA6	UDA5	UDA4	UDA3	UDA2	UDA1	UDA0	R/W	UDA
0A1H		BUS_RST	SUSPEND	EP0_SETUP	EP0_IN	EP0_OUT	EP1_ACK	EP2_ACK	R/W	USTATUS
0A2H	FFS2	FFS1	FFS0	UEP0OC4	UEP0OC3	UEP0OC2	UEP0OC1	UEP0OC0	R/W	UCTRL
0A3H		UE0M1	UE0M0		UE0C3	UE0C2	UE0C1	UE0C0	R/W	UE0E
0A4H	UE1E	UE1M1	UE1M0	UE1C4	UE1C3	UE1C2	UE1C1	UE1C0	R/W	UE1E
0A5H	UE2E	UE2M1	UE2M0	UE2C4	UE2C3	UE2C2	UE2C1	UE2C0	R/W	UE2E
0A6H				UDP04	UDP03	UDP02	UDP01	UDP00	R/W	UDP0
0A7H	UDR07	UDR06	UDR05	UDR04	UDR03	UDR02	UDR01	UDR00	R/W	UDR0
0A8H				UDP14	UDP13	UDP12	UDP11	UDP10	R/W	UDP1
0A9H	UDR17	UDR16	UDR15	UDR14	UDR13	UDR12	UDR11	UDR10	R/W	UDR1
0AAH						UBDE	DDP	DDN	R/W	UPID
0ABH				UEP1OC4	UEP1OC3	UEP1OC2	UEP1OC1	UEP1OC0	R/W	EP1OUT_CNT
0ACH				UEP2OC4	UEP2OC3	UEP2OC2	UEP2OC1	UEP2OC0	R/W	EP2OUT_CNT
0B4H	SENB	START	SRATE1	SRATE0	0	SCKMD	SEDGE	TXRX	R/W	SIOM
0B5H	SIOR7	SIOR6	SIOR5	SIOR4	SIOR3	SIOR2	SIOR1	SIOR0	W	SIOR
0B6H	SIOB7	SIOB6	SIOB5	SIOB4	SIOB3	SIOB2	SIOB1	SIOB0	R/W	SIOB
0B8H				P04M	P03M	P02M	P01M	P00M	R/W	P0M
0BFH				P00G1	P00G0				R/W	PEDGE
0C0H	P17W	P16W	P15W	P14W	P13W	P12W	P11W	P10W	R/W	P1W
0C1H	P17M	P16M	P15M	P14M	P13M	P12M	P11M	P10M	R/W	P1M
0C5H						P52M	P51M	P50M	R/W	P5M
0C8H		USBIRQ	TC0IRQ	T0IRQ	SIOIRQ		WAKEIRQ	P00IRQ	R/W	INTRQ
0C9H		USBIEN	TC0IEN	T0IEN	SIOIEN		WAKEIEN	P00IEN	R/W	INTEN
0CAH				CPUM1	CPUM0	CLKMD	STPHX		R/W	OSCM
0CCH	WDTR7	WDTR6	WDTR5	WDTR4	WDTR3	WDTR2	WDTR1	WDTR0	W	WDTR
0CDH										
0CEH	PC7	PC6	PC5	PC4	PC3	PC2	PC1	PC0	R/W	PCL
0CFH				PC12	PC11	PC10	PC9	PC8	R/W	PCH
0D0H			P05	P04	P03	P02	P01	P00	R/W	P0
0D1H	P17	P16	P15	P14	P13	P12	P11	P10	R/W	P1
0D5H						P52	P51	P50	R/W	P5
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	PWM0OUT	R/W	TC0M
0DBH	TC0C7	TC0C6	TC0C5	TC0C4	TC0C3	TC0C2	TC0C1	TC0C0	R/W	TC0C
0DFH	GIE					STKPB2	STKPB1	STKPB0	R/W	STKP
0E0H		P06R	P05R	P04R	P03R	P02R	P01R	P00R	W	P0UR
0E1H	P17R	P16R	P15R		P13R	P12R	P11R	P10R	W	P1UR
0E5H	P57R	P56R	P55R	P54R	P53R	P52R	P51R	P50R	W	P5UR
0E7H	@YZ7	@YZ6	@YZ5	@YZ4	@YZ3	@YZ2	@YZ1	@YZ0	R/W	@YZ
0E9H					P52OC	P50OC	P11OC	P10OC	W	P1OC
0F0H	S7PC7	S7PC6	S7PC5	S7PC4	S7PC3	S7PC2	S7PC1	S7PC0	R/W	STK7L
0F1H				S7PC12	S7PC11	S7PC10	S7PC9	S7PC8	R/W	STK7H
0F2H	S6PC7	S6PC6	S6PC5	S6PC4	S6PC3	S6PC2	S6PC1	S6PC0	R/W	STK6L
0F3H				S6PC12	S6PC11	S6PC10	S6PC9	S6PC8	R/W	STK6H
0F4H	S5PC7	S5PC6	S5PC5	S5PC4	S5PC3	S5PC2	S5PC1	S5PC0	R/W	STK5L
0F5H				S5PC12	S5PC11	S5PC10	S5PC9	S5PC8	R/W	STK5H
0F6H	S4PC7	S4PC6	S4PC5	S4PC4	S4PC3	S4PC2	S4PC1	S4PC0	R/W	STK4L
0F7H				S4PC12	S4PC11	S4PC10	S4PC9	S4PC8	R/W	STK4H
0F8H	S3PC7	S3PC6	S3PC5	S3PC4	S3PC3	S3PC2	S3PC1	S3PC0	R/W	STK3L

0F9H				S3PC12	S3PC11	S3PC10	S3PC9	S3PC8	R/W	STK3H
0FAH	S2PC7	S2PC6	S2PC5	S2PC4	S2PC3	S2PC2	S2PC1	S2PC0	R/W	STK2L
0FBH				S2PC12	S2PC11	S2PC10	S2PC9	S2PC8	R/W	STK2H
0FCH	S1PC7	S1PC6	S1PC5	S1PC4	S1PC3	S1PC2	S1PC1	S1PC0	R/W	STK1L
0FDH				S1PC12	S1PC11	S1PC10	S1PC9	S1PC8	R/W	STK1H
0FEH	S0PC7	S0PC6	S0PC5	S0PC4	S0PC3	S0PC2	S0PC1	S0PC0	R/W	STK0L
0FFH				S0PC12	S0PC11	S0PC10	S0PC9	S0PC8	R/W	STK0H

✷ ***Note:***

1. *To avoid system error, please be sure to put all the "0" and "1" as it indicates in the above table.*
2. *All of register names had been declared in SN8ASM assembler.*
3. *One-bit name had been declared in SN8ASM assembler with "F" prefix code.*
4. *"b0bset", "b0bclr", "bset", "bclr" instructions are only available to the "R/W" registers.*
5. *For detail description, please refer to the "System Register Quick Reference Table".*

2.1.4.4 ACCUMULATOR

The ACC is an 8-bit data register responsible for transferring or manipulating data between ALU and data memory. If the result of operating is zero (Z) or there is carry (C or DC) occurrence, then these flags will be set to PFLAG register. ACC is not in data memory (RAM), so ACC can't be access by "B0MOV" instruction during the instant addressing mode.

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

```
; Read ACC data and store in BUF data memory.
                MOV         BUF, A

; Write a immediate data into ACC.
                MOV         A, #0FH

; Write ACC data from BUF data memory.
                MOV         A, BUF
; or
                B0MOV       A, BUF
```

The system doesn't store ACC and PFLAG value when interrupt executed. ACC and PFLAG data must be saved to other data memories. "PUSH", "POP" save and load ACC, PFLAG data into buffers.

- **Example: Protect ACC and working registers.**

```
INT_SERVICE:
                PUSH                ; Save ACC and PFLAG to buffers.
                …           .
                …
                POP                 ; Load ACC and PFLAG from buffers.

                RETI                ; Exit interrupt service vector
```

2.1.4.5 PROGRAM FLAG

The PFLAG register contains the arithmetic status of ALU operation, system reset status and LVD detecting status. NT0, NPD bits indicate system reset status including power on reset, LVD reset, reset by external pin active and watchdog reset. C, DC, Z bits indicate the result status of ALU operation.

086H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
PFLAG	NT0	NPD	-	-	-	C	DC	Z
Read/Write	R/W	R/W	-	-	-	R/W	R/W	R/W
After reset	-	-	-	-	-	0	0	0

Bit [7:6] **NT0, NPD:** Reset status flag.

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

Bit 2 **C:** Carry flag
1 = Addition with carry, subtraction without borrowing, rotation with shifting out logic “1”, comparison result ≥ 0.
0 = Addition without carry, subtraction with borrowing signal, rotation with shifting out logic “0”, comparison result < 0.

Bit 1 **DC:** Decimal carry flag
1 = Addition with carry from low nibble, subtraction without borrow from high nibble.
0 = Addition without carry from low nibble, subtraction with borrow from high nibble.

Bit 0 **Z:** Zero flag
1 = The result of an arithmetic/logic/branch operation is zero.
0 = The result of an arithmetic/logic/branch operation is not zero.

✸ ***Note: Refer to instruction set table for detailed information of C, DC and Z flags.***

2.1.4.6 PROGRAM COUNTER

The program counter (PC) is a 13-bit binary counter separated into the high-byte 5 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 12.

	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	-	-	-	PC12	PC11	PC10	PC9	PC8	PC7	PC6	PC5	PC4	PC3	PC2	PC1	PC0
After reset	-	-	-	0	0	0	0	0	0	0	0	0	0	0	0	0
	PCH								PCL							

☞ **ONE ADDRESS SKIPPING**

There are nine instructions (CMPRS, INCS, INCMS, DECS, DECMS, BTS0, BTS1, B0BTS0, B0BTS1) with one address skipping function. If the result of these instructions is true, the PC will add 2 steps to skip next instruction.

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

```
            B0BTS1    FC          ; To skip, if Carry_flag = 1
            JMP       C0STEP      ; Else jump to C0STEP.
            …
            …
C0STEP:     NOP

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

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

```
            CMPRS     A, #12H     ; To skip, if ACC = 12H.
            JMP       C0STEP      ; Else jump to C0STEP.
            …
            …
C0STEP:     NOP
```

If the destination increased by 1, which results overflow of 0xFF to 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
```

If the destination decreased by 1, which results underflow of 0x00 to 0xFF, the PC will add 2 steps to skip next instruction.

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 around the multi-address by either JMP instruction or ADD M, A instruction (M = PCL) to activate multi-address jumping function. Program Counter supports **"ADD M,A"**, **"ADC M,A"** and **"B0ADD M,A"** instructions for carry to PCH when PCL overflow automatically. For jump table or others applications, users can calculate PC value by the three instructions and don't care PCL overflow problem.

✸ ***Note: PCH only support PC up counting result and doesn't support PC down counting. When PCL is carry after PCL+ACC, PCH adds one automatically. If PCL borrow after PCL–ACC, PCH keeps value and not change.***

➢ **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
                …
                …
```

2.1.4.7 Y, Z REGISTERS

The Y and Z registers are the 8-bit buffers. There are three major functions of these registers.

- can be used as general working registers
- can be used as RAM data pointers with @YZ register
- can be used as ROM data pointer with the MOVC instruction for look-up table

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

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

Example: Uses Y, Z register as the data pointer to access data in the RAM address 025H of bank0.

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

Example: Uses the Y, Z register as data pointer to clear the RAM data.

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

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

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

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

2.1.4.8 R REGISTERS

R register is an 8-bit buffer. There are two major functions of the register.

- Can be used as working register
- For store high-byte data of look-up table
  (MOVC instruction executed, the high-byte data of specified ROM address will be stored in R register and the low-byte data will be stored in ACC).

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

✷ ***Note: Please refer to the "LOOK-UP TABLE DESCRIPTION" about R register look-up table application.***

2.2 ADDRESSING MODE

2.2.1 IMMEDIATE ADDRESSING MODE

The immediate addressing mode uses an immediate data to set up the location in ACC or specific RAM.

- **Example: Move the immediate data 12H to ACC.**

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

- **Example: Move the immediate data 12H to R register.**

```
B0MOV   R, #12H     ; To set an immediate data 12H into R register.
```

✸ ***Note: In immediate addressing mode application, the specific RAM must be 0x80~0x87 working register.***

2.2.2 DIRECTLY ADDRESSING MODE

The directly addressing mode moves the content of RAM location in or out of ACC.

- **Example: Move 0x12 RAM location data into ACC.**

```
B0MOV   A, 12H      ; To get a content of RAM location 0x12 of bank 0 and save in
                    ACC.
```

- **Example: Move ACC data into 0x12 RAM location.**

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

2.2.3 INDIRECTLY ADDRESSING MODE

The indirectly addressing mode is to access the memory by the data pointer registers (Y/Z).

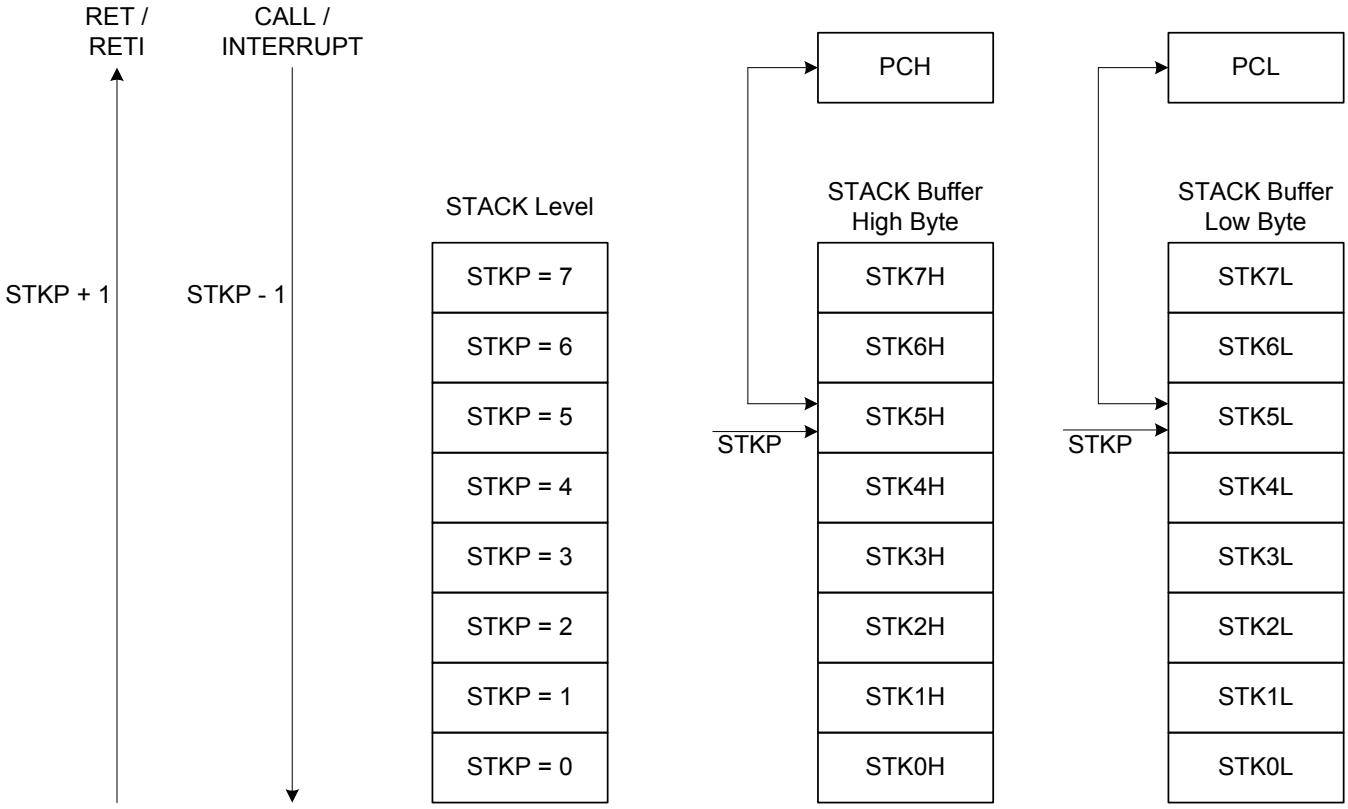
- **Example: Indirectly addressing mode with @YZ register.**

```
B0MOV   Y, #0       ; 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.
```

2.3 STACK OPERATION

2.3.1 OVERVIEW

The stack buffer has 8-level. These buffers are designed to push and pop up program counter's (PC) data when interrupt service routine and "CALL" instruction are executed. The STKP register is a pointer designed to point active level in order to push or pop up data from stack buffer. The STKnH and STKnL are the stack buffers to store program counter (PC) data.



2.3.2 STACK REGISTERS

The stack pointer (STKP) is a 3-bit register to store the address used to access the stack buffer, 13-bit data memory (STKnH and STKnL) set aside for temporary storage of stack addresses.

The two stack operations are writing to the top of the stack (push) and reading from the top of stack (pop). Push operation decrements the STKP and the pop operation increments each time. That makes the STKP always point to the top address of stack buffer and write the last program counter value (PC) into the stack buffer.

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

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

Bit[2:0] **STKPBn:** Stack pointer (n = 0 ~ 2)

Bit 7 **GIE:** Global interrupt control bit.
0 = Disable.
1 = Enable. Please refer to the interrupt chapter.

➢ **Example: Stack pointer (STKP) reset, we strongly recommended to clear the stack pointers in the beginning of the program.**

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

0F0H~0FFH	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
STKnH	-	-	-	SnPC12	SnPC11	SnPC10	SnPC9	SnPC8
Read/Write	-	-	-	R/W	R/W	R/W	R/W	R/W
After reset	-	-	-	0	0	0	0	0

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

STKn = STKnH , STKnL (n = 7 ~ 0)

2.3.3 STACK OPERATION EXAMPLE

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

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

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

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

3 RESET

3.1 OVERVIEW

The system would be reset in three conditions as following.

- Power on reset
- Watchdog reset
- Brown out reset
- External reset (only supports external reset pin enable situation)

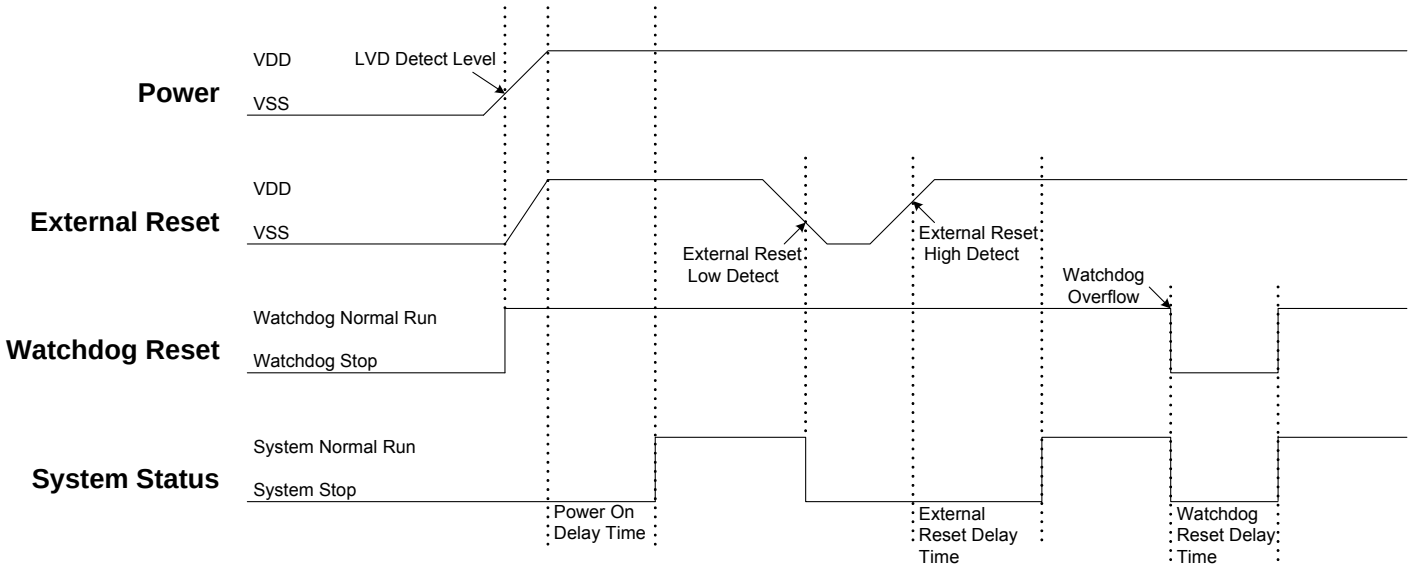
When any reset condition occurs, all system registers keep initial status, program stops and program counter is cleared. After reset status released, the system boots up and program starts to execute from ORG 0. The NT0, NPD flags indicate system reset status. The system can depend on NT0, NPD status and go to different paths by program.

086H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
PFLAG	NT0	NPD	-	-	-	C	DC	Z
Read/Write	R/W	R/W	-	-	-	R/W	R/W	R/W
After reset	-	-	-	-	-	0	0	0

Bit [7:6] **NT0, NPD:** Reset status flag.

NT0	NPD	Condition	Description
0	0	Watchdog reset	Watchdog timer overflow.
0	1	Reserved	-
1	0	Power on reset and LVD reset.	Power voltage is lower than LVD detecting level.
1	1	External reset	External reset pin detect low level status.

Finishing any reset sequence needs some time. The system provides complete procedures to make the power on reset successful. For different oscillator types, the reset time is different. That causes the VDD rise rate and start-up time of different oscillator is not fixed. RC type oscillator's start-up time is very short, but the crystal type is longer. Under client terminal application, users have to take care the power on reset time for the master terminal requirement. The reset timing diagram is as following.



3.2 POWER ON RESET

The power on reset depend no LVD operation for most power-up situations. The power supplying to system is a rising curve and needs some time to achieve the normal voltage. Power on reset sequence is as following.

- **Power-up:** System detects the power voltage up and waits for power stable.
- **External reset (only external reset pin enable):** System checks external reset pin status. If external reset pin is not high level, the system keeps reset status and waits external reset pin released.
- **System initialization:** All system registers is set as initial conditions and system is ready.
- **Oscillator warm up:** Oscillator operation is successfully and supply to system clock.
- **Program executing:** Power on sequence is finished and program executes from ORG 0.

3.3 WATCHDOG RESET

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

- **Watchdog timer status:** System checks watchdog timer overflow status. If watchdog timer overflow occurs, the system is reset.
- **System initialization:** All system registers is set as initial conditions and system is ready.
- **Oscillator warm up:** Oscillator operation is successfully and supply to system clock.
- **Program executing:** Power on sequence is finished and program executes from ORG 0.

Watchdog timer application note is as following.

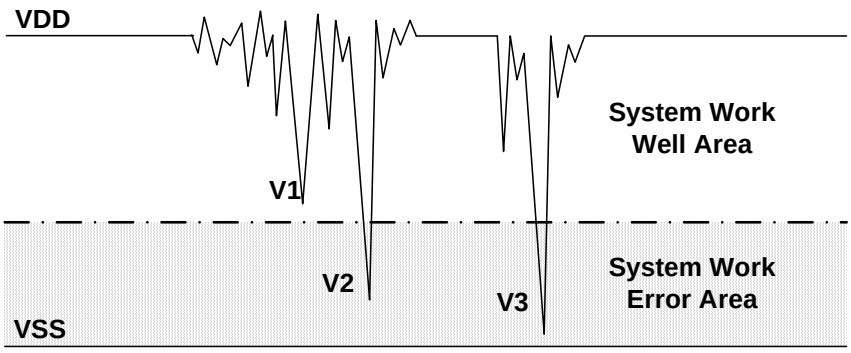
- Before clearing watchdog timer, check I/O status and check RAM contents can improve system error.
- Don't clear watchdog timer in interrupt vector and interrupt service routine. That can improve main routine fail.
- Clearing watchdog timer program is only at one part of the program. This way is the best structure to enhance the watchdog timer function.

✷ ***Note: Please refer to the "WATCHDOG TIMER" about watchdog timer detail information.***

3.4 BROWN OUT RESET

3.4.1 BROWN OUT DESCRIPTION

The brown out reset is a power dropping condition. The power drops from normal voltage to low voltage by external factors (e.g. EFT interference or external loading changed). The brown out reset would make the system not work well or executing program error.



Brown Out Reset Diagram

The power dropping might through the voltage range that's the system dead-band. The dead-band means the power range can't offer the system minimum operation power requirement. The above diagram is a typical brown out reset diagram. There is a serious noise under the VDD, and VDD voltage drops very deep. There is a dotted line to separate the system working area. The above area is the system work well area. The below area is the system work error area called dead-band. V1 doesn't touch the below area and not effect the system operation. But the V2 and V3 is under the below area and may induce the system error occurrence. Let system under dead-band includes some conditions.

DC application:

The power source of DC application is usually using battery. When low battery condition and MCU drive any loading, the power drops and keeps in dead-band. Under the situation, the power won't drop deeper and not touch the system reset voltage. That makes the system under dead-band.

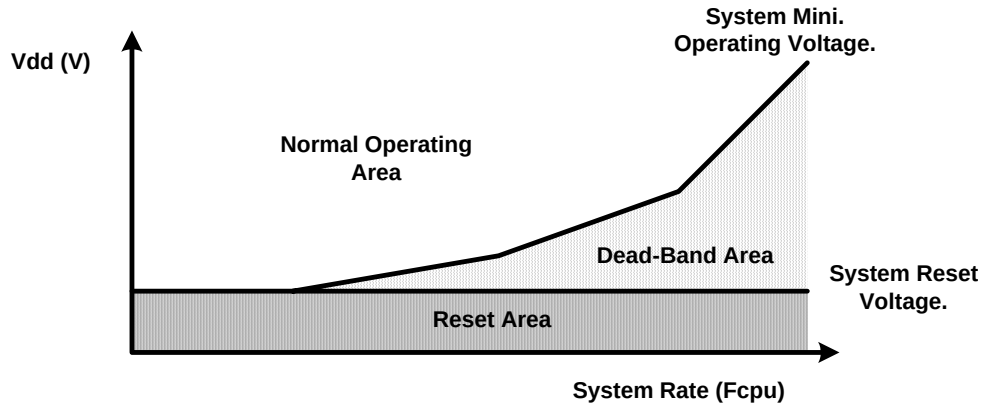
AC application:

In AC power application, the DC power is regulated from AC power source. This kind of power usually couples with AC noise that makes the DC power dirty. Or the external loading is very heavy, e.g. driving motor. The loading operating induces noise and overlaps with the DC power. VDD drops by the noise, and the system works under unstable power situation.

The power on duration and power down duration are longer in AC application. The system power on sequence protects the power on successful, but the power down situation is like DC low battery condition. When turn off the AC power, the VDD drops slowly and through the dead-band for a while.

3.4.2 THE SYSTEM OPERATING VOLTAGE DECSRIPTION

To improve the brown out reset needs to know the system minimum operating voltage which is depend on the system executing rate and power level. Different system executing rates have different system minimum operating voltage. The electrical characteristic section shows the system voltage to executing rate relationship.



Normally the system operation voltage area is higher than the system reset voltage to VDD, and the reset voltage is decided by LVD detect level. The system minimum operating voltage rises when the system executing rate upper even higher than system reset voltage. The dead-band definition is the system minimum operating voltage above the system reset voltage.

3.4.3 BROWN OUT RESET IMPROVEMENT

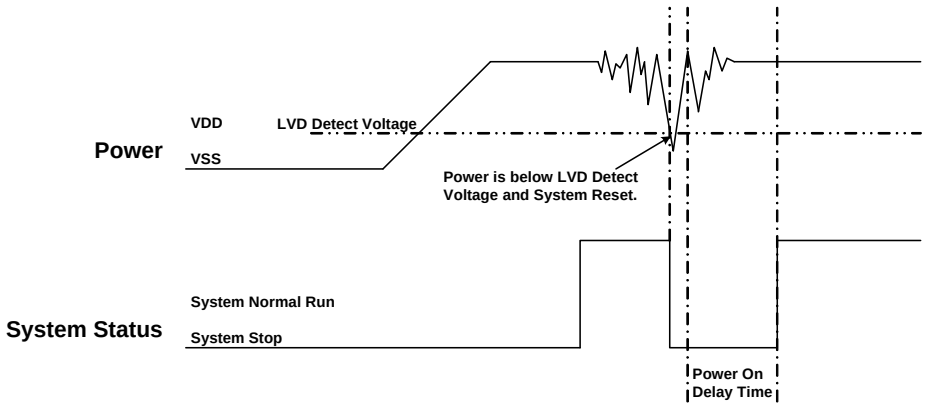
How to improve the brown reset condition? There are some methods to improve brown out reset as following.

- **LVD reset**
- **Watchdog reset**
- **Reduce the system executing rate**
- **External reset circuit. (Zener diode reset circuit, Voltage bias reset circuit, External reset IC)**

✷ ***Note:***

1. ***The " Zener diode reset circuit", "Voltage bias reset circuit" and "External reset IC" can completely improve the brown out reset, DC low battery and AC slow power down conditions.***
2. ***For AC power application and enhance EFT performance, the system clock is 4MHz/4 (1 mips) and use external reset (" Zener diode reset circuit", "Voltage bias reset circuit", "External reset IC"). The structure can improve noise effective and get good EFT characteristic.***

LVD reset:



The LVD (low voltage detector) is built-in Sonix 8-bit MCU to be brown out reset protection. When the VDD drops and is below LVD detect voltage, the LVD would be triggered, and the system is reset. The LVD detect level is different by each MCU. The LVD voltage level is a point of voltage and not easy to cover all dead-band range. Using LVD to improve brown out reset is depend on application requirement and environment. If the power variation is very deep, violent and trigger the LVD, the LVD can be the protection. If the power variation can touch the LVD detect level and make system work error, the LVD can't be the protection and need to other reset methods. More detail LVD information is in the electrical characteristic section.

Watchdog reset:

The watchdog timer is a protection to make sure the system executes well. Normally the watchdog timer would be clear at one point of program. Don't clear the watchdog timer in several addresses. The system executes normally and the watchdog won't reset system. When the system is under dead-band and the execution error, the watchdog timer can't be clear by program. The watchdog is continuously counting until overflow occurrence. The overflow signal of watchdog timer triggers the system to reset, and the system return to normal mode after reset sequence. This method also can improve brown out reset condition and make sure the system to return normal mode.
If the system reset by watchdog and the power is still in dead-band, the system reset sequence won't be successful and the system stays in reset status until the power return to normal range.

Reduce the system executing rate:

If the system rate is fast and the dead-band exists, to reduce the system executing rate can improve the dead-band. The lower system rate is with lower minimum operating voltage. Select the power voltage that's no dead-band issue and find out the mapping system rate. Adjust the system rate to the value and the system exits the dead-band issue. This way needs to modify whole program timing to fit the application requirement.

External reset circuit:

The external reset methods also can improve brown out reset and is the complete solution. There are three external reset circuits to improve brown out reset including "Zener diode reset circuit", "Voltage bias reset circuit" and "External reset IC". These three reset structures use external reset signal and control to make sure the MCU be reset under power dropping and under dead-band. The external reset information is described in the next section.

3.5 EXTERNAL RESET

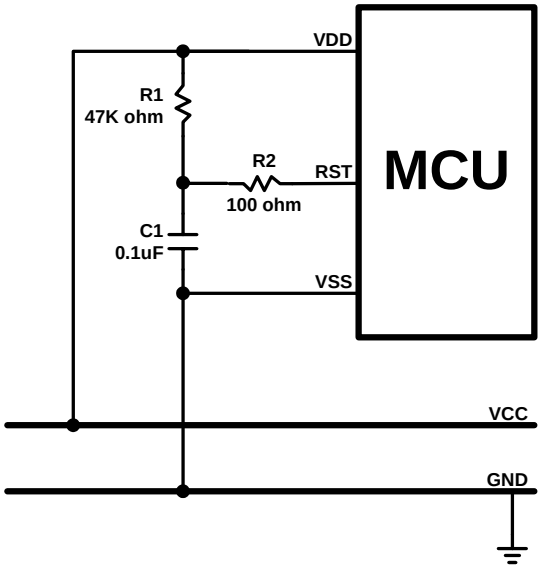
External reset function is controlled by “Reset_Pin” code option. Set the code option as “Reset” option to enable external reset function. External reset pin is Schmitt Trigger structure and low level active. The system is running when reset pin is high level voltage input. The reset pin receives the low voltage and the system is reset. The external reset operation actives in power on and normal running mode. During system power-up, the external reset pin must be high level input, or the system keeps in reset status. External reset sequence is as following.

- **External reset (only external reset pin enable):** System checks external reset pin status. If external reset pin is not high level, the system keeps reset status and waits external reset pin released.
- **System initialization:** All system registers is set as initial conditions and system is ready.
- **Oscillator warm up:** Oscillator operation is successfully and supply to system clock.
- **Program executing:** Power on sequence is finished and program executes from ORG 0.

The external reset can reset the system during power on duration, and good external reset circuit can protect the system to avoid working at unusual power condition, e.g. brown out reset in AC power application…

3.6 EXTERNAL RESET CIRCUIT

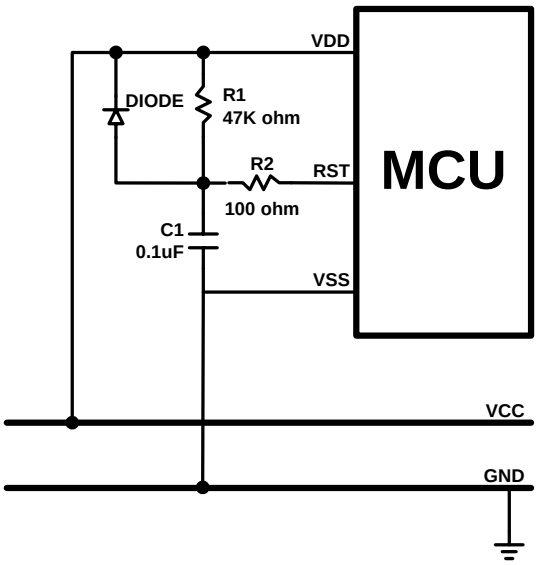
3.6.1 Simply RC Reset Circuit



This is the basic reset circuit, and only includes R1 and C1. The RC circuit operation makes a slow rising signal into reset pin as power up. The reset signal is slower than VDD power up timing, and system occurs a power on signal from the timing difference.

✷ ***Note: The reset circuit is no any protection against unusual power or brown out reset.***

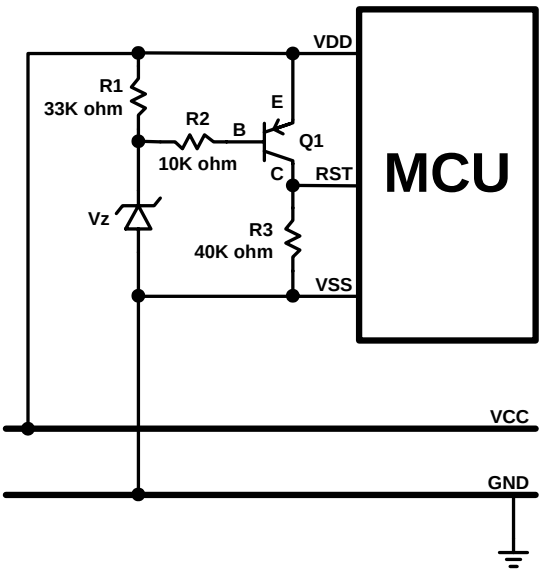
3.6.2 Diode & RC Reset Circuit



This is the better reset circuit. The R1 and C1 circuit operation is like the simply reset circuit to make a power on signal. The reset circuit has a simply protection against unusual power. The diode offers a power positive path to conduct higher power to VDD. It is can make reset pin voltage level to synchronize with VDD voltage. The structure can improve slight brown out reset condition.

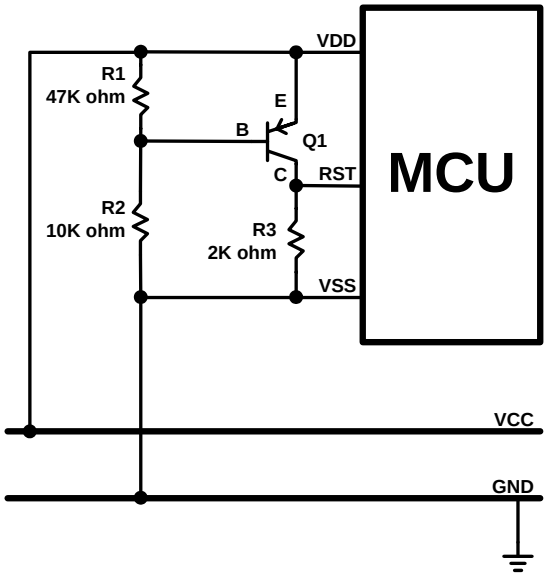
✷ ***Note: The R2 100 ohm resistor of "Simply reset circuit" and "Diode & RC reset circuit" is necessary to limit any current flowing into reset pin from external capacitor C in the event of reset pin breakdown due to Electrostatic Discharge (ESD) or Electrical Over-stress (EOS).***

3.6.3 Zener Diode Reset Circuit



The zener diode reset circuit is a simple low voltage detector and can **improve brown out reset condition completely**. Use zener voltage to be the active level. When VDD voltage level is above "Vz + 0.7V", the C terminal of the PNP transistor outputs high voltage and MCU operates normally. When VDD is below "Vz + 0.7V", the C terminal of the PNP transistor outputs low voltage and MCU is in reset mode. Decide the reset detect voltage by zener specification. Select the right zener voltage to conform the application.

3.6.4 Voltage Bias Reset Circuit

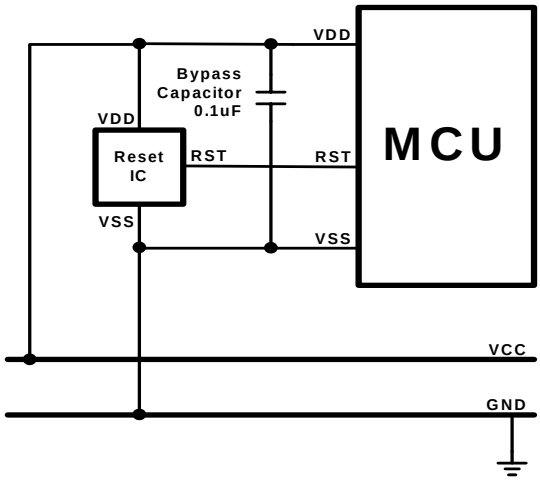


The voltage bias reset circuit is a low cost voltage detector and can **improve brown out reset condition completely**. The operating voltage is not accurate as zener diode reset circuit. Use R1, R2 bias voltage to be the active level. When VDD voltage level is above or equal to "0.7V x (R1 + R2) / R1", the C terminal of the PNP transistor outputs high voltage and MCU operates normally. When VDD is below "0.7V x (R1 + R2) / R1", the C terminal of the PNP transistor outputs low voltage and MCU is in reset mode.

Decide the reset detect voltage by R1, R2 resistances. Select the right R1, R2 value to conform the application. In the circuit diagram condition, the MCU's reset pin level varies with VDD voltage variation, and the differential voltage is 0.7V. If the VDD drops and the voltage lower than reset pin detect level, the system would be reset. If want to make the reset active earlier, set the R2 > R1 and the cap between VDD and C terminal voltage is larger than 0.7V. The external reset circuit is with a stable current through R1 and R2. For power consumption issue application, e.g. DC power system, the current must be considered to whole system power consumption.

✷ ***Note: Under unstable power condition as brown out reset, "Zener diode rest circuit" and "Voltage bias reset circuit" can protects system no any error occurrence as power dropping. When power drops below the reset detect voltage, the system reset would be triggered, and then system executes reset sequence. That makes sure the system work well under unstable power situation.***

3.6.5 External Reset IC



The external reset circuit also use external reset IC to enhance MCU reset performance. This is a high cost and good effect solution. By different application and system requirement to select suitable reset IC. The reset circuit can improve all power variation

4 SYSTEM CLOCK

4.1 OVERVIEW

The micro-controller is a dual clock system. There are high-speed clock and low-speed clock. The high-speed clock is generated from the external oscillator circuit. The low-speed clock is generated from on-chip low-speed RC oscillator circuit (ILRC 16KHz @3V, 32KHz @5V).

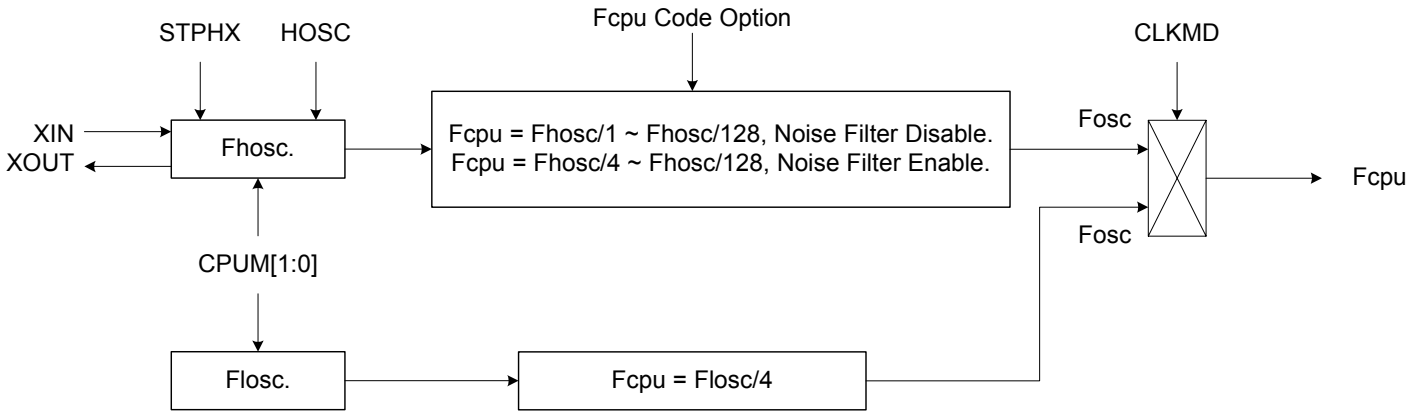
Both the high-speed clock and the low-speed clock can be system clock (Fosc). The system clock in slow mode is divided by 4 to be the instruction cycle (Fcpu).

☞ **Normal Mode (High Clock):** **Fcpu = Fhosc / N**, N = 1 ~ 4, Select N by Fcpu code option.

☞ **Slow Mode (Low Clock):** **Fcpu = Flosc/4.**

SONIX provides a **"Noise Filter"** controlled by code option. In high noisy situation, the noise filter can isolate noise outside and protect system works well. The minimum Fcpu of high clock is limited at **Fhosc/4** when noise filter enable.

4.2 CLOCK BLOCK DIAGRAM



- HOSC: High_Clk code option.
- Fhosc: External high-speed clock.
- Flosc: Internal low-speed RC clock (about 16KHz@3V, 32KHz@5V).
- Fosc: System clock source.
- Fcpu: Instruction cycle.

4.3 OSCM REGISTER

The OSCM register is an oscillator control register. It controls oscillator status, system mode.

0CAH	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
OSCM	0	0	0	CPUM1	CPUM0	CLKMD	STPHX	0
Read/Write	-	-	-	R/W	R/W	R/W	R/W	-
After reset	-	-	-	0	0	0	0	-

Bit 1 **STPHX:** External high-speed oscillator control bit.
0 = External high-speed oscillator free run.
1 = External high-speed oscillator free run stop. Internal low-speed RC oscillator is still running.

Bit 2 **CLKMD:** System high/Low clock mode control bit.
0 = Normal (dual) mode. System clock is high clock.
1 = Slow mode. System clock is internal low clock.

Bit[4:3] **CPUM[1:0]:** CPU operating mode control bits.
00 = normal.
01 = sleep (power down) mode.
10 = green mode.
11 = reserved.

➢ **Example: Stop high-speed oscillator**

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

Example: When entering the power down mode (sleep mode), both high-speed oscillator and internal low-speed oscillator will be stopped.

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

4.4 SYSTEM HIGH CLOCK

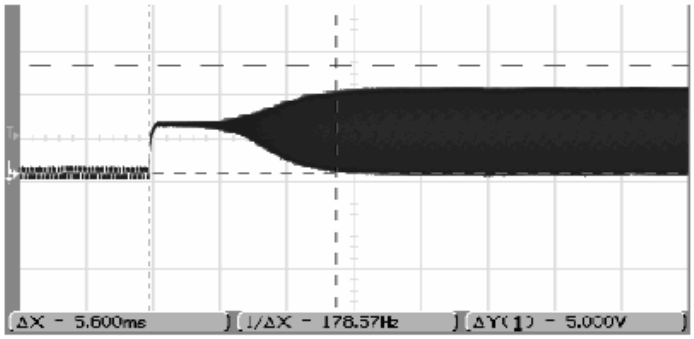
The system high clock is from external oscillator. The high clock type is controlled by “High_Clk” code option.

High_Clk Code Option	Description
6MHz	The high clock is external high speed oscillator. The typical frequency is 6MHz.

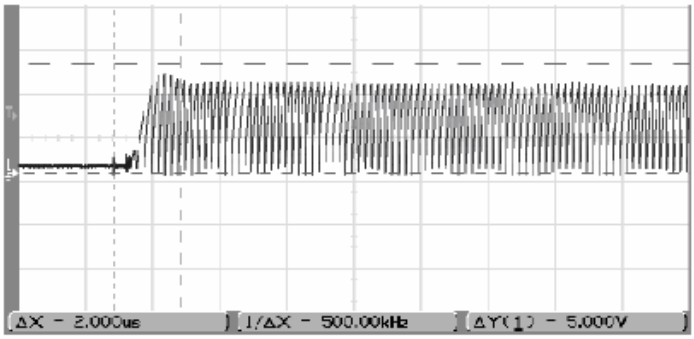
4.4.1 EXTERNAL HIGH CLOCK

External high clock includes three modules (Crystal/Ceramic, RC and external clock signal). The high clock oscillator module is controlled by High_Clk code option. The start up time of crystal/ceramic and RC type oscillator is different. RC type oscillator’s start-up time is very short, but the crystal’s is longer. The oscillator start-up time decides reset time length.

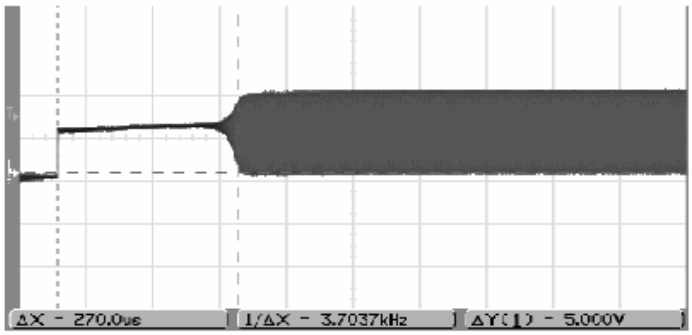
4MHz Crystal



RC

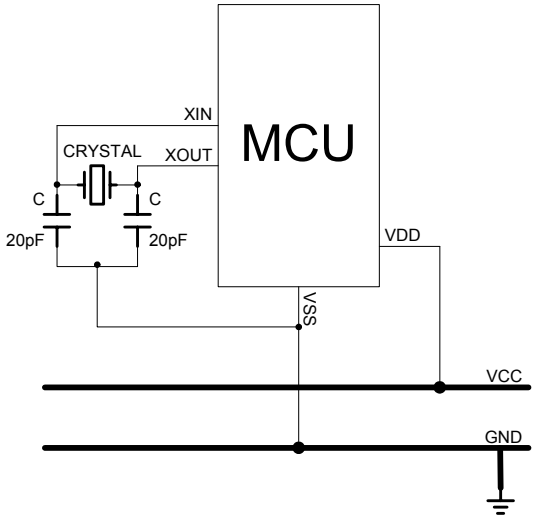


4MHz Ceramic



4.4.1.1 CRYSTAL/CERAMIC

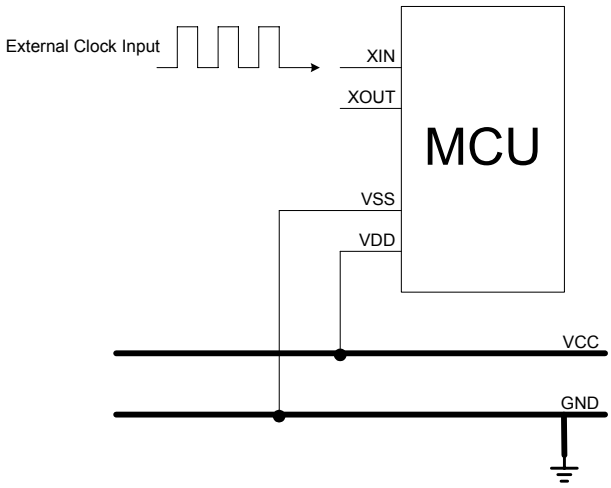
Crystal/Ceramic devices are driven by XIN, XOUT pins.



✸ ***Note: Connect the Crystal/Ceramic and C as near as possible to the XIN/XOUT/VSS pins of micro-controller.***

4.1.1.2 EXTERNAL CLOCK SIGNAL

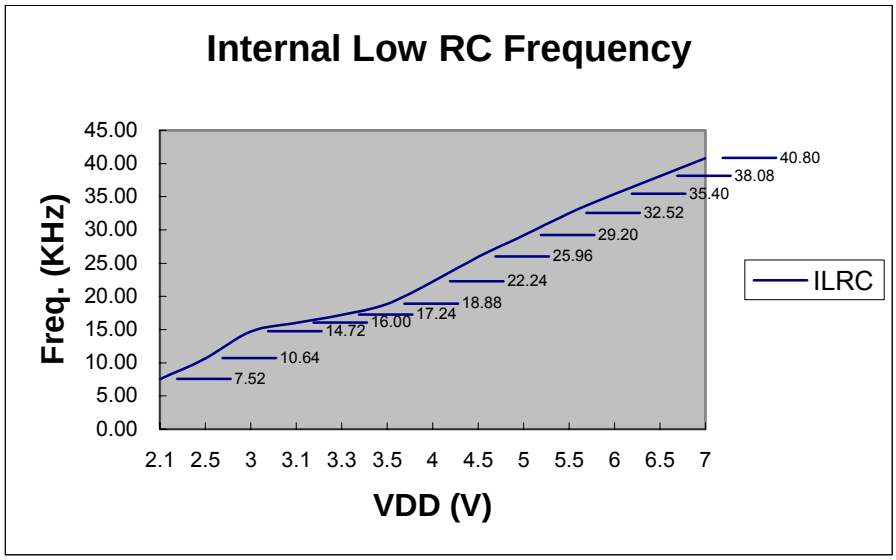
Selecting external clock signal input to be system clock is by RC option of High_Clk code option. The external clock signal is input from XIN pin. XOUT pin is general purpose I/O pin.



✸ ***Note: The GND of external oscillator circuit must be as near as possible to VSS pin of micro-controller.***

4.2 SYSTEM LOW CLOCK

The system low clock source is the internal low-speed oscillator built in the micro-controller. The low-speed oscillator uses RC type oscillator circuit. The frequency is affected by the voltage and temperature of the system. In common condition, the frequency of the RC oscillator is about 16KHz at 3V and 32KHz at 5V. The relation between the RC frequency and voltage is as the following figure.



The internal low RC supports watchdog clock source and system slow mode controlled by CLKMD.

☞ ***Flosc = Internal low RC oscillator (about 16KHz @3V, 32KHz @5V).***

☞ ***Slow mode Fcpu = Flosc / 4***

There are two conditions to stop internal low RC. One is power down mode, and the other is green mode of 32K mode and watchdog disable. If system is in 32K mode and watchdog disable, only 32K oscillator actives and system is under low power consumption.

➢ **Example: Stop internal low-speed oscillator by power down mode.**

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

✸ ***Note: The internal low-speed clock can't be turned off individually. It is controlled by CPUM0, CPUM1 (32K, watchdog disable) bits of OSCM register.***

4.2.1 SYSTEM CLOCK MEASUREMENT

Under design period, the users can measure system clock speed by software instruction cycle (Fcpu). This way is useful in RC mode.

Example: Fcpu instruction cycle of external oscillator.

```
        B0BSET    P0M.0       ; Set P0.0 to be output mode for outputting Fcpu toggle signal.
@@:
        B0BSET    P0.0        ; Output Fcpu toggle signal in low-speed clock mode.
        B0BCLR    P0.0        ; Measure the Fcpu frequency by oscilloscope.
        JMP       @B
```

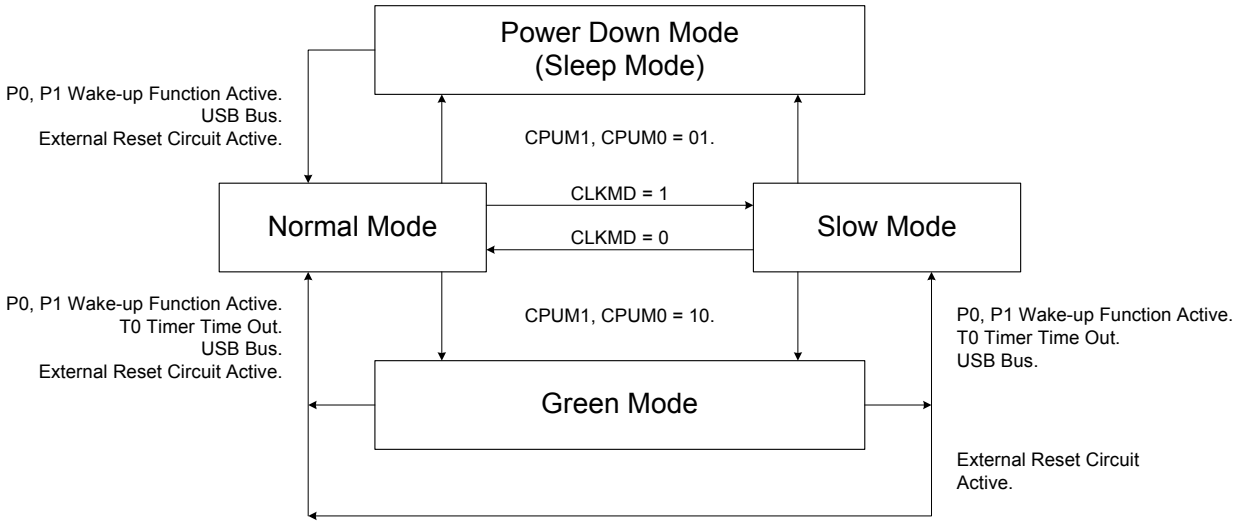
✸ ***Note: Do not measure the RC frequency directly from XIN; the probe impendence will affect the RC frequency.***

5 SYSTEM OPERATION MODE

5.1 OVERVIEW

The chip is featured with low power consumption by switching around four different modes as following.

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



System Mode Switching Diagram

Operating mode description

MODE	NORMAL	SLOW	GREEN	POWER DOWN (SLEEP)	REMARK
EHOSC	Running	By STPHX	By STPHX	Stop	
ILRC	Running	Running	Running	Stop	
CPU instruction	Executing	Executing	Stop	Stop	
T0 timer	*Active	*Active	*Active	Inactive	* Active if T0ENB=1
TC0 timer	*Active	*Active	Inactive	Inactive	* Active if TC0ENB=1
Watchdog timer	By Watch_Dog Code option	By Watch_Dog Code option	By Watch_Dog Code option	By Watch_Dog Code option	Refer to code option description
Internal interrupt	All active	All active	T0	All inactive	
External interrupt	All active	All active	All active	All inactive	
Wakeup source	-	-	P0, P1, T0 Reset	P0, P1, Reset	

- **EHOSC:** External high clock
- **ILRC:** Internal low clock (16K RC oscillator at 3V, 32K at 5V)

5.2 SYSTEM MODE SWITCHING EXAMPLE

➢ **Example: Switch normal/slow mode to power down (sleep) mode.**

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

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

➢ **Example: Switch normal mode to slow mode.**

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

➢ **Example: Switch slow mode to normal mode (The external high-speed oscillator is still running).**

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

Example: 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.

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

Example: Switch normal/slow mode to green mode.

```
        B0BSET      FCPUM1          ; Set CPUM1 = 1.
```

✸ ***Note: If T0 timer wakeup function is disabled in the green mode, only the wakeup pin and reset pin can wakeup the system backs to the previous operation mode.***

Example: Switch normal/slow mode to green mode and enable T0 wake-up function.

```
; Set T0 timer wakeup function.
        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: During the green mode with T0 wake-up function, the wakeup pin and T0 wakeup the system back to the last mode. T0 wake-up period is controlled by program.***

5.3 WAKEUP

5.3.1 OVERVIEW

Under power down mode (sleep mode) or green mode, program doesn't execute. The wakeup trigger can wake the system up to normal mode or slow mode. The wakeup trigger sources are external trigger (P0, P1 level change), internal trigger (T0 timer overflow) and USB bus toggle.

- Power down mode is waked up to normal mode. The wakeup trigger is only external trigger (P0, P1 level change and USB bus toggle)
- Green mode is waked up to last mode (normal mode or slow mode). The wakeup triggers are external trigger (P0, P1 level change), internal trigger (T0 timer overflow) and USB bus toggle.

5.3.2 WAKEUP TIME

When the system is in power down mode (sleep mode), the high clock oscillator stops. When waked up from power down mode, MCU waits for 4 internal 6MHz clock or 2048 external 6MHz clocks as the wakeup time to stable the oscillator circuit. After the wakeup time, the system goes into the normal mode.

✷ ***Note: Wakeup from green mode is no wakeup time because the clock doesn't stop in green mode.***

The value of the wakeup time is as the following.

"6M_X'tal" mode:

The Wakeup time = 1/Fosc * 2048 (sec) + high clock start-up time

✷ ***Note: The high clock start-up time is depended on the VDD and oscillator type of high clock.***

Example: In 6M_X'tal mode and power down mode (sleep mode), the system is waked up. After the wakeup time, the system goes into normal mode. The wakeup time is as the following.

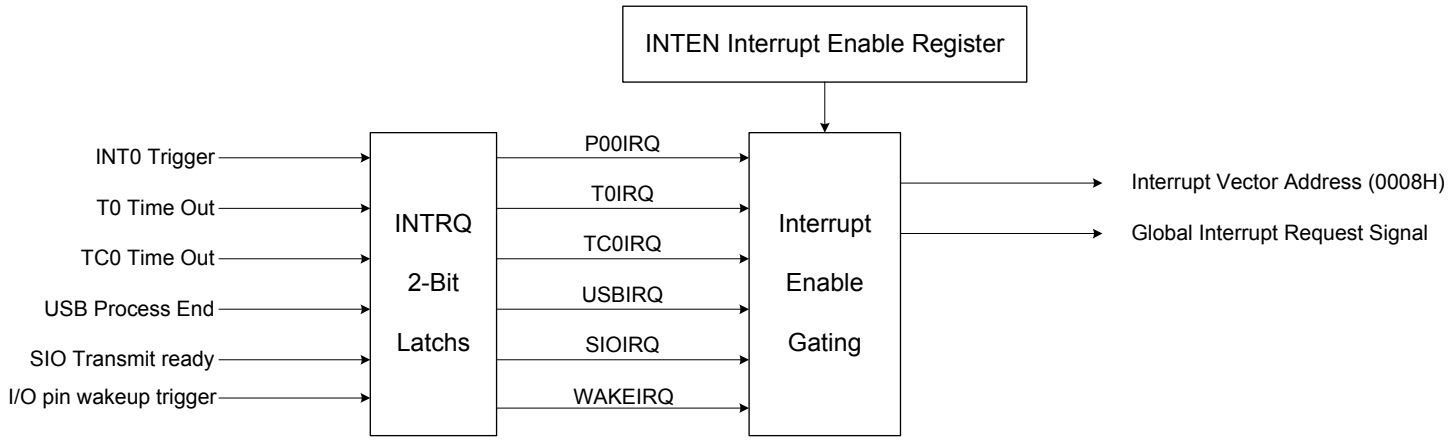
The wakeup time = 1/Fosc * 2048 = 0.341 ms (Fosc = 6MHz)

The total wakeup time = 0.341 ms + oscillator start-up time

6 INTERRUPT

6.1 OVERVIEW

This MCU provides 5 interrupt sources, including 4 internal interrupt (T0/TC0/USB/SIO) and one external interrupt (INT0). The external interrupt can wakeup the chip while the system is switched from power down mode to high-speed normal mode. Once interrupt service is executed, the GIE bit in STKP register will clear to “0” for stopping other interrupt request. On the contrast, when interrupt service exits, the GIE bit will set to “1” to accept the next interrupts’ request. All of the interrupt request signals are stored in INTRQ register.



✸ ***Note: The GIE bit must enable during all interrupt operation.***

6.2 INTEN INTERRUPT ENABLE REGISTER

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

0C9H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
INTEN	-	USBIEN	TC0IEN	T0IEN	SIOIEN		WAKEIEN	P00IEN
Read/Write	-	R/W	R/W	R/W	R/W		R/W	R/W
After reset	-	0	0	0	0		0	0

Bit 0 **P00IEN:** External P0.0 interrupt (INT0) control bit.
0 = Disable INT0 interrupt function.
1 = Enable INT0 interrupt function.

Bit 1 **WAKEIEN:** I/O PORT0 & PORT 1 WAKEUP interrupt control bit.
0 = Disable WAKEUP interrupt function.
1 = Enable WAKEUP interrupt function.

Bit 3 **SIOIEN:** SIO interrupt control bit.
0 = Disable SIO interrupt function.
1 = Enable SIO interrupt function.

Bit 4 **T0IEN:** T0 timer interrupt control bit.
0 = Disable T0 interrupt function.
1 = Enable T0 interrupt function.

Bit 5 **TC0IEN:** TC0 timer interrupt control bit.
0 = Disable TC0 interrupt function.
1 = Enable TC0 interrupt function.

Bit 6 **USBIEN:** USB interrupt control bit.
0 = Disable USB interrupt function.
1 = Enable USB interrupt function.

6.3 INTRQ INTERRUPT REQUEST REGISTER

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

0C8H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
INTRQ	-	USBIRQ	TC0IRQ	T0IRQ	SIOIRQ		WAKEIRQ	P00IRQ
Read/Write	-	R/W	R/W	R/W	R/W		R/W	R/W
After reset	-	0	0	0	0		0	0

Bit 0 **P00IRQ:** External P0.0 interrupt (INT0) request flag.
0 = None INT0 interrupt request.
1 = INT0 interrupt request.

Bit 1 **WAKEIRQ:** I/O PORT0 & PORT1 WAKEUP interrupt request flag.
0 = None WAKEUP interrupt request.
1 = WAKEUP interrupt request.

Bit 3 **SIOIRQ:** SIO interrupt request flag.
0 = None SIO interrupt request.
1 = SIO interrupt request.

Bit 4 **T0IRQ:** T0 timer interrupt request flag.
0 = None T0 interrupt request.
1 = T0 interrupt request.

Bit 5 **TC0IRQ:** TC0 timer interrupt request flag.
0 = None TC0 interrupt request.
1 = TC0 interrupt request.

Bit 6 **USBIRQ:** USB timer interrupt request flag.
0 = None USB interrupt request.
1 = USB interrupt request.

6.4 GIE GLOBAL INTERRUPT OPERATION

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

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

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

Example: Set global interrupt control bit (GIE).

```
B0BSET      FGIE        ; Enable GIE
```

✷ ***Note: The GIE bit must enable during all interrupt operation.***

6.5 PUSH, POP ROUTINE

When any interrupt occurs, system will jump to ORG 8 and execute interrupt service routine. It is necessary to save ACC, PFLAG data. The chip includes “PUSH”, “POP” for in/out interrupt service routine. The two instructions save and load **ACC**, **PFLAG** data into buffers and avoid main routine error after interrupt service routine finishing.

✸ ***Note: ”PUSH”, “POP” instructions save and load ACC/PFLAG without (NT0, NPD). PUSH/POP buffer is an unique buffer and only one level.***

➢ **Example: Store ACC and PAFLG data by PUSH, POP instructions when interrupt service routine executed.**

```
                ORG     0
                JMP     START

                ORG     8
                JMP     INT_SERVICE

                ORG     10H
START:
                …

INT_SERVICE:
                PUSH                    ; Save ACC and PFLAG to buffers.
                …
                …
                POP                     ; Load ACC and PFLAG from buffers.

                RETI                    ; Exit interrupt service vector
                …
                ENDP
```

6.6 INT0 (P0.0) INTERRUPT OPERATION

When the INT0 trigger occurs, the P00IRQ will be set to "1" no matter the P00IEN is enable or disable. If the P00IEN = 1 and the trigger event P00IRQ is also set to be "1". As the result, the system will execute the interrupt vector (ORG 8). If the P00IEN = 0 and the trigger event P00IRQ is still set to be "1". Moreover, the system won't execute interrupt vector even when the P00IRQ is set to be "1". Users need to be cautious with the operation under multi-interrupt situation.

If the interrupt trigger direction is identical with wake-up trigger direction, the INT0 interrupt request flag (INT0IRQ) is latched while system wake-up from power down mode or green mode by P0.0 wake-up trigger. System inserts to interrupt vector (ORG 8) after wake-up immediately.

✸ ***Note: INT0 interrupt request can be latched by P0.0 wake-up trigger.***

✸ ***Note: The interrupt trigger direction of P0.0 is control by PEDGE register.***

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

Bit[4:3] **P00G[1:0]:** P0.0 interrupt trigger edge control bits.
00 = reserved.
01 = rising edge.
10 = falling edge.
11 = rising/falling bi-direction (Level change trigger).

Example: Setup INT0 interrupt request and bi-direction edge trigger.

```
MOV     A, #18H
B0MOV   PEDGE, A        ; Set INT0 interrupt trigger as bi-direction edge.

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:
            …                           ; Push routine to save ACC and PFLAG to buffers.

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

            B0BCLR      FP00IRQ         ; Reset P00IRQ
            …                           ; INT0 interrupt service routine
            …
EXIT_INT:
            …                           ; Pop routine to load ACC and PFLAG from buffers.

            RETI                        ; Exit interrupt vector
```

6.7 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:

        …                           ; Push routine to save ACC and PFLAG to buffers.

        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 routine to load ACC and PFLAG from buffers.

        RETI                        ; Exit interrupt vector
```

6.8 TC0 INTERRUPT OPERATION

When the TC0C counter overflows, the TC0IRQ will be set to "1" no matter the TC0IEN is enable or disable. If the TC0IEN and the trigger event TC0IRQ is set to be "1". As the result, the system will execute the interrupt vector. If the TC0IEN = 0, the trigger event TC0IRQ is still set to be "1". Moreover, the system won't execute interrupt vector even when the TC0IEN is set to be "1". Users need to be cautious with 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:

        …                           ; Push routine to save ACC and PFLAG to buffers.

        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 routine to load ACC and PFLAG from buffers.

        RETI                        ; Exit interrupt vector
```

6.9 USB INTERRUPT OPERATION

When the USB process finished, the USBIRQ will be set to "1" no matter the USBIEN is enable or disable. If the USBIEN and the trigger event USBIRQ is set to be "1". As the result, the system will execute the interrupt vector. If the USBIEN = 0, the trigger event USBIRQ is still set to be "1". Moreover, the system won't execute interrupt vector. Users need to be cautious with the operation under multi-interrupt situation.

➢ **Example: USB interrupt request setup.**

```
        B0BCLR      FUSBIEN     ; Disable USB interrupt service
        B0BCLR      FUSBIRQ     ; Clear USB interrupt request flag
        B0BSET      FUSBIEN     ; Enable USB interrupt service

        …                       ; USB initialize.
        …                       ; USB operation.

        B0BSET      FGIE        ; Enable GIE
```

Example: USB interrupt service routine.

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

        PUSH                    ; Push routine to save ACC and PFLAG to buffers.

        B0BTS1      FUSBIRQ     ; Check USBIRQ
        JMP         EXIT_INT    ; USBIRQ = 0, exit interrupt vector

        B0BCLR      FUSBIRQ     ; Reset USBIRQ

        …                       ; USB interrupt service routine
        …
EXIT_INT:
        POP                     ; Pop routine to load ACC and PFLAG from buffers.

        RETI                    ; Exit interrupt vector
```

6.10 WAKEUP INTERRUPT OPERATION

When the I/O port 1 or I/O port 0 wakeup the MCU from the sleep mode, the WAKEIRQ will be set to "1" no matter the WAKEIEN is enable or disable. If the WAKEIEN and the trigger event WAKEIRQ is set to be "1". As the result, the system will execute the interrupt vector. If the WAKEIEN = 0, the trigger event WAKEIRQ is still set to be "1". Moreover, the system won't execute interrupt vector. Users need to be cautious with the operation under multi-interrupt situation.

➢ **Example: WAKE interrupt request setup.**

```
        B0BCLR      FWAKEIEN        ; Disable WAKE interrupt service
        B0BCLR      FWAKEIRQ        ; Clear WAKE interrupt request flag
        B0BSET      FWAKEIEN        ; Enable WAKE interrupt service

        …                           ; Pin WAKEUP initialize.
        …                           ; Pin WAKEUP operation.

        B0BSET      FGIE            ; Enable GIE
```

Example: WAKE interrupt service routine.

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

        PUSH                        ; Push routine to save ACC and PFLAG to buffers.

        B0BTS1      FWAKEIRQ        ; Check WAKEIRQ
        JMP         EXIT_INT        ; WAKEIRQ = 0, exit interrupt vector

        B0BCLR      FWAKEIRQ        ; Reset WAKEIRQ

        …                           ; WAKE interrupt service routine
        …
EXIT_INT:
        POP                         ; Pop routine to load ACC and PFLAG from buffers.

        RETI                        ; Exit interrupt vector
```

6.11 SIO INTERRUPT OPERATION

When the SIO converting successfully, the SIOIRQ will be set to "1" no matter the SIOIEN is enable or disable. If the SIOIEN and the trigger event SIOIRQ is set to be "1". As the result, the system will execute the interrupt vector. If the SIOIEN = 0, the trigger event SIOIRQ is still set to be "1". Moreover, the system won't execute interrupt vector even when the SIOIEN is set to be "1". Users need to be cautious with 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:
        …                       ; Push routine to save ACC and PFLAG to buffers.

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

        B0BCLR      FSIOIRQ     ; Reset SIOIRQ
        …                       ; SIO interrupt service routine
        …
EXIT_INT:
        …                       ; Pop routine to load ACC and PFLAG from buffers.

        RETI                    ; Exit interrupt vector
```

6.12 MULTI-INTERRUPT OPERATION

Under certain condition, the software designer uses more than one interrupt requests. Processing multi-interrupt request requires setting the priority of the interrupt requests. The IRQ flags of interrupts are controlled by the interrupt event. Nevertheless, the IRQ flag "1" doesn't mean the system will execute the interrupt vector. In addition, which means the IRQ flags can be set "1" by the events without enable the interrupt. Once the event occurs, 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 controlled by PEDGE
T0IRQ	T0C overflow
TC0IRQ	TC0C overflow
USBIRQ	USB process finished
WAEKIRQ	I/O port0 & port1 wakeup MCU
SIOIRQ	SIO process finished

For multi-interrupt conditions, two things need to be taking care of. One is to set the priority for these interrupt requests. Two is using IEN and IRQ flags to decide which interrupt to be executed. Users have to check interrupt control bit and interrupt request flag in interrupt routine.

➢ **Example: Check the interrupt request under multi-interrupt operation**

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

        …                       ; Push routine to save ACC and PFLAG to buffers.

INTP00CHK:                      ; Check INT0 interrupt request
        B0BTS1  FP00IEN         ; Check P00IEN
        JMP     INTT0CHK        ; Jump check to next interrupt
        B0BTS0  FP00IRQ         ; Check P00IRQ
        JMP     INTP00
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
INTUSBCHK:                      ; Check USB interrupt request
        B0BTS1  FUSBIEN         ; Check USBIEN
        JMP     INTWAKECHK      ; Jump check to next interrupt
        B0BTS0  FUSBIRQ         ; Check USBIRQ
        JMP     INTUSB          ; Jump to USB interrupt service routine
INTWAKECHK:                     ; Check USB interrupt request
        B0BTS1  FWAKEIEN        ; Check WAKEIEN
        JMP     INTSIOCHK       ; Jump check to next interrupt
        B0BTS0  FWAKEIRQ        ; Check WAKEIRQ
        JMP     INTWAKEUP       ; Jump to WAKEUP interrupt service routine
INTSIOCHK:                      ; Check SIO interrupt request
        B0BTS1  FSIOIEN         ; Check SIOIEN
        JMP     INT_EXIT        ; Jump check to next interrupt
        B0BTS0  FSIOIRQ         ; Check SIOIRQ
        JMP     INTSIO          ; Jump to SIO interrupt service routine
INT_EXIT:
        …                       ; Pop routine to load ACC and PFLAG from buffers.

        RETI                    ; Exit interrupt vector
```

7 I/O PORT

7.1 I/O PORT MODE

The port direction is programmed by PnM register. All I/O ports can select input or output direction.

0B8H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
P0M	-	-	P05M	P04M	P03M	P02M	P01M	P00M
Read/Write	-	-	R	R/W	R/W	R/W	R/W	R/W
After reset	-	-	0	0	0	0	0	0

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

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

Bit[7:0] **PnM[7:0]:** Pn mode control bits. (n = 0~5).
0 = Pn is input mode.
1 = Pn is output mode.

✷ ***Note:***
1. ***Users can program them by bit control instructions (B0BSET, B0BCLR).***
2. ***P0.5 is input only pin, so there is no P0.5 mode control bit.***

➢ **Example: I/O mode selecting**

```
CLR     P0M             ; Set all ports to be input mode.
CLR     P1M
CLR     P5M

MOV     A, #0FFH        ; Set all ports to be output mode.
B0MOV   P0M, A
B0MOV   P1M, A
B0MOV   P5M, A

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

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

7.2 I/O PULL UP REGISTER

0E0H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
P0UR	-			P04R	P03R	P02R	P01R	P00R
Read/Write	-			W	W	W	W	W
After reset	-			0	0	0	0	0

0E1H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
P1UR	P17R	P16R	P15R	P16R	P13R	P12R	P11R	P10R
Read/Write	W	W	W	W	W	W	W	W
After reset	0	0	0	0	0	0	0	0

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

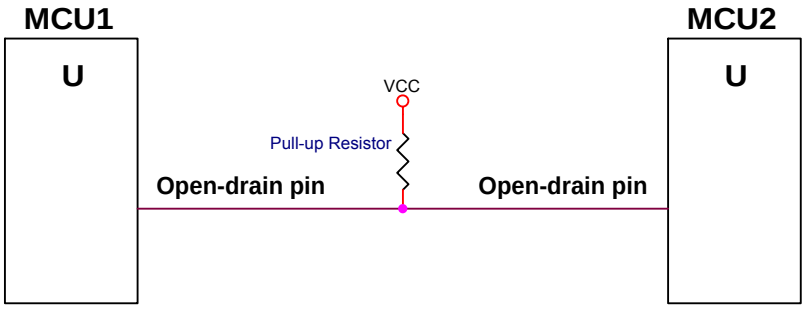
✸ ***Note: P0.5 is input only pin without pull-up resister, so there is no P0.5 pull-up resistor control bit.***
✸ ***Note: When set P0.5 to input mode, please add the series external 100 ohm on it.***

➢ **Example: I/O Pull up Register**

```
MOV     A, #0FFH        ; Enable Port0, 1, 5 Pull-up register,
B0MOV   P0UR, A         ;
B0MOV   P1UR, A
B0MOV   P5UR, A
```

7.3 I/O OPEN-DRAIN REGISTER

P1.0/P1.1 is built-in open-drain function. P1.0/P1.1 must be set as output mode when enable P1.0/P1.1 open-drain function. Open-drain external circuit is as following.



The pull-up resistor is necessary. Open-drain output high is driven by pull-up resistor. Output low is sunken by MCU's pin.

✸ ***Note: P1.0/P1.1 open-drain function can be 2nd PS/2 interface on chip. More detail information refer to PS/2 chapter.***

0E9H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
P1OC	-	-	-	-	P52OC	P50OC	P11OC	P10OC
Read/Write	-	-	-	-	W	W	W	W
After reset	-	-	-	-	0	0	0	0

Bit [1:0] **P1nOC:** Port 1 open-drain control bit
0 = Disable open-drain mode
1 = Enable open-drain mode

Bit [3:2] **P5nOC:** Port 5 open-drain control bit
0 = Disable open-drain mode
1 = Enable open-drain mode

➢ **Example: Enable P1.0 to open-drain mode and output high.**

```
B0BSET      P1.0        ; Set P1.0 buffer high.

B0BSET      P10M        ; Enable P1.0 output mode.
MOV         A, #01H     ; Enable P1.0 open-drain function.
B0MOV       P1OC, A
```

✸ ***Note: P1OC is a write only register. Setting P10OC must be used "MOV" instructions.***

➢ **Example: Disable P1.0 to open-drain mode and output low.**

```
MOV         A, #0xE     ; Disable P1.0 open-drain function.
B0MOV       P1OC, A
```

✸ ***Note: After disable P1.0 open-drain function, P1.0 mode returns to last I/O mode.***

7.4 I/O PORT DATA REGISTER

0D0H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
P0	-		P05	P04	P03	P02	P01	P00
Read/Write	-		R	R/W	R/W	R/W	R/W	R/W
After reset	-		0	0	0	0	0	0

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

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

✷ ***Note: The P05 keeps "1" when external reset enable by code option.***

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

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

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

```
MOV     A, #0FFH    ; Write data FFH to all Port.
B0MOV   P0, A
B0MOV   P1, A
B0MOV   P5, A
```

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

```
B0BSET  P1.3        ; Set P1.3 and P5.5 to be "1".
B0BSET  P5.5

B0BCLR  P1.3        ; Set P1.3 and P5.5 to be "0".
B0BCLR  P5.5
```

7.5 I/O PORT1 WAKEUP CONTROL REGISTER

0D0H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
P1W	P17W	P16W	P15W	P14W	P13W	P12W	P11W	P10W
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
After reset	0	0	0	0	0	0	0	0

Bit [7:0] **P1nW:** Port 1 wakeup function control bit
0 = Disable port 1 wakeup function
1 = Enable port 1 wakeup function

8 TIMERS

8.1 WATCHDOG TIMER

The watchdog timer (WDT) is a binary up counter designed for monitoring program execution. If the program goes into the unknown status by noise interference, WDT overflow signal raises and resets MCU. Watchdog clock controlled by code option and the clock source is internal low-speed oscillator (16KHz @3V, 32KHz @5V).

Watchdog overflow time = 8192 / Internal Low-Speed oscillator (sec).

VDD	*Internal Low RC Freq.*	*Watchdog Overflow Time*
3V	16KHz	512ms
5V	32KHz	256ms

✷ ***Note: If watchdog is "Always_On" mode, it keeps running event under power down mode or green mode.***

Watchdog clear is controlled by WDTR register. Moving **0x5A** data into WDTR is to reset watchdog timer.

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

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

```
Main:
        MOV     A,#5AH      ; Clear the watchdog timer.
        B0MOV   WDTR,A
        …
        CALL    SUB1
        CALL    SUB2
        …
        …
        …
        JMP     MAIN
```

Watchdog timer application note is as following.

- Before clearing watchdog timer, check I/O status and check RAM contents can improve system error.

- Don't clear watchdog timer in interrupt vector and interrupt service routine. That can improve main routine fail.
- Clearing watchdog timer program is only at one part of the program. This way is the best structure to enhance the watchdog timer function.

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

```
Main:
                …                               ; Check I/O.
                …                               ; Check RAM
Err:            JMP $                           ; I/O or RAM error. Program jump here and don't
                                                ; clear watchdog. Wait watchdog timer overflow to reset IC.

Correct:                                        ; I/O and RAM are correct. Clear watchdog timer and
                                                ; execute program.
                B0BSET          FWDRST          ; Only one clearing watchdog timer of whole program.
                …
                CALL            SUB1
                CALL            SUB2
                …
                …
                …
                JMP             MAIN
```

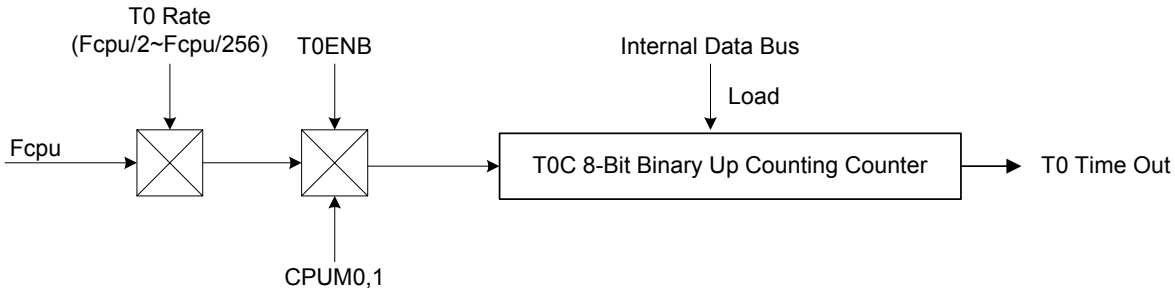
8.2 TIMER 0 (T0)

8.2.1 OVERVIEW

The T0 is an 8-bit binary up timer and event counter. If T0 timer occurs 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 purpose of the T0 timer is as following.

- **8-bit programmable up counting timer:** Generates interrupts at specific time intervals based on the selected clock frequency.
- **Green mode wakeup function:** T0 can be green mode wake-up time as T0ENB = 1. System will be wake-up by T0 time out.



8.2.2 T0M MODE REGISTER

0D8H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
T0M	T0ENB	T0rate2	T0rate1	T0rate0	-	-	-	
Read/Write	R/W	R/W	R/W	R/W	-	-	-	
After reset	0	0	0	0	-	-	-	

Bit [6:4] **T0RATE[2:0]:** T0 internal clock select bits.
000 = fcpu/256.
001 = fcpu/128.
…
110 = fcpu/4.
111 = fcpu/2.

Bit 7 **T0ENB:** T0 counter control bit.
0 = Disable T0 timer.
1 = Enable T0 timer.

8.2.3 T0C COUNTING REGISTER

T0C is an 8-bit counter register for T0 interval time control.

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

The equation of T0C initial value is as following.

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

Example: To set 1ms interval time for T0 interrupt. High clock is 12MHz. Fcpu=Fosc/2. Select T0RATE=010 (Fcpu/64).

*T0C initial value = 256 - (T0 interrupt interval time * input clock)*
*= 256 - (1ms * 6MHz / 1 / 64)*
*= 256 - (10^{-3} * 6 * 10^{6} / 1 / 64)*
= 162
= A2H

The basic timer table interval time of T0.

T0RATE	T0CLOCK	High speed mode (Fcpu = 12MHz / 2)	
		Max overflow interval	One step = max/256
000	Fcpu/256	10.923 ms	42.67 us
001	Fcpu/128	5.461 ms	21.33 us
010	Fcpu/64	2.731 ms	10.67 us
011	Fcpu/32	1.365 ms	5.33 us
100	Fcpu/16	0.683 ms	2.67 us
101	Fcpu/8	0.341 ms	1.33 us
110	Fcpu/4	0.171 ms	0.67 us
111	Fcpu/2	0.085 ms	0.33 us

8.2.4 T0 TIMER OPERATION SEQUENCE

T0 timer operation sequence of setup T0 timer is as following.

☞ **Stop T0 timer counting, disable T0 interrupt function and clear T0 interrupt request flag.**

```
B0BCLR      FT0ENB          ; T0 timer.
B0BCLR      FT0IEN          ; T0 interrupt function is disabled.
B0BCLR      FT0IRQ          ; T0 interrupt request flag is cleared.
```

☞ **Set T0 timer rate.**

```
MOV         A, #0xxx0000b   ;The T0 rate control bits exist in bit4~bit6 of T0M. The
                            ; value is from x000xxxxb~x111xxxxb.
B0MOV       T0M,A           ; T0 timer is disabled.
```

☞ **Set T0 interrupt interval time.**

```
MOV         A,#7FH
B0MOV       T0C,A           ; Set T0C value.
```

☞ **Set T0 timer function mode.**

```
B0BSET      FT0IEN          ; Enable T0 interrupt function.
```

☞ **Enable T0 timer.**

```
B0BSET      FT0ENB          ; Enable T0 timer.
```

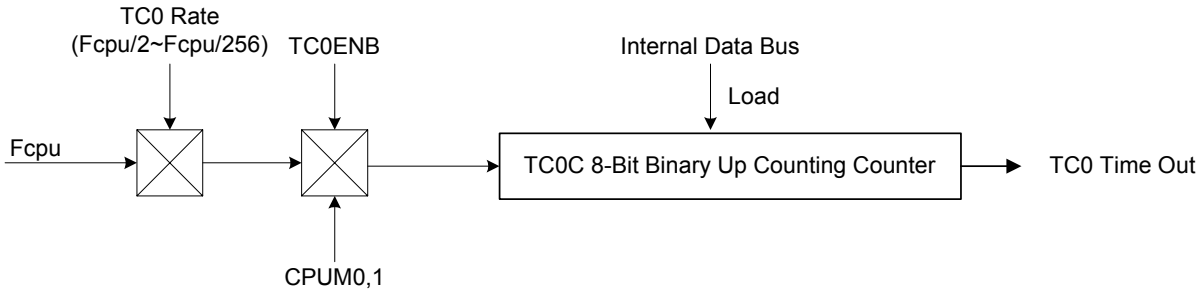
8.3 TIMER C0 (TC0)

8.3.1 OVERVIEW

The TC0 is an 8-bit binary up timer and event counter. If TC0 timer occurs an overflow (from FFH to 00H), it will continue counting and issue a time-out signal to trigger TC0 interrupt to request interrupt service.

The main purpose of the TC0 timer is as following.

- ☞ **8-bit programmable up counting timer:** Generates interrupts at specific time intervals based on the selected clock frequency.
- ☞ **Green mode wakeup function:** TC0 can be green mode wake-up time as TC0ENB = 1. System will be wake-up by TC0 time out.



8.3.2 TC0M MODE REGISTER

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

Bit [6:4] **TC0RATE[2:0]:** TC0 internal clock select bits.
000 = fcpu/256.
001 = fcpu/128.
…
110 = fcpu/4.
111 = fcpu/2.

Bit 7 **TC0ENB:** TC0 counter control bit.
0 = Disable TC0 timer.
1 = Enable TC0 timer.

8.3.3 TC0C COUNTING REGISTER

TC0C is an 8-bit counter register for TC0 interval time control.

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

The equation of TC0C initial value is as following.

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

Example: To set 1ms interval time for TC0 interrupt. High clock is 12MHz. Fcpu=Fosc/2. Select TC0RATE=010 (Fcpu/64).

*TC0C initial value = 256 - (TC0 interrupt interval time * input clock)*
*= 256 - (1ms * 6MHz / 1 / 64)*
*= 256 - (10^{-3} * 6 * 10^{6} / 1 / 64)*
= 162
= A2H

The basic timer table interval time of TC0.

TC0RATE	TC0CLOCK	High speed mode (Fcpu = 12MHz / 2)	
		Max overflow interval	One step = max/256
000	Fcpu/256	10.923 ms	42.67 us
001	Fcpu/128	5.461 ms	21.33 us
010	Fcpu/64	2.731 ms	10.67 us
011	Fcpu/32	1.365 ms	5.33 us
100	Fcpu/16	0.683 ms	2.67 us
101	Fcpu/8	0.341 ms	1.33 us
110	Fcpu/4	0.171 ms	0.67 us
111	Fcpu/2	0.085 ms	0.33 us

8.3.4 TC0 TIMER OPERATION SEQUENCE

TC0 timer operation sequence of setup TC0 timer is as following.

☞ **Stop TC0 timer counting, disable TC0 interrupt function and clear TC0 interrupt request flag.**

```
B0BCLR    FTC0ENB     ; TC0 timer.
B0BCLR    FTC0IEN     ; TC0 interrupt function is disabled.
B0BCLR    FTC0IRQ     ; TC0 interrupt request flag is cleared.
```

☞ **Set TC0 timer rate.**

```
MOV       A, #0xxx0000b   ;The TC0 rate control bits exist in bit4~bit6 of TC0M. The
                          ; value is from x000xxxxb~x111xxxxb.
B0MOV     TC0M,A          ; TC0 timer is disabled.
```

☞ **Set TC0 interrupt interval time.**

```
MOV     A,#7FH
B0MOV   TC0C,A          ; Set TC0C value.
```

☞ **Set TC0 timer function mode.**

```
B0BSET  FTC0IEN         ; Enable TC0 interrupt function.
```

☞ **Enable TC0 timer.**

```
B0BSET  FTC0ENB         ; Enable TC0 timer.
```

9 UNIVERSAL SERIAL BUS (USB)

9.1 OVERVIEW

The USB is the answer to connectivity for the PC architecture. A fast, bi-directional, isochronous, low-cost, dynamically attachable serial interface is consistent with the requirements of the PC platform of today and tomorrow. The SONIX USB microcontrollers are optimized for human-interface computer peripherals such as a mouse, joystick, and game pad.

USB Specification Compliance

— Conforms to USB specifications, Version 2.0.

— Conforms to USB HID Specification, Version 1.11.

— Supports 1 Full-speed USB device address.

— Supports 1 control endpoint and 2 interrupt endpoints.

— Integrated USB transceiver.

— 5V to 3.3V regulator output for D+ resistor pull up.

9.2 USB MACHINE

The USB machine allows the microcontroller to communicate with the USB host. The hardware handles the following USB bus activity independently of the microcontroller.

The USB machine will do:

- Translate the encoded received data and format the data to be transmitted on the bus.
- CRC checking and generation by hardware. If CRC is not correct, hardware will not send any response to USB host.
- Send and update the data toggle bit (Data1/0) automatically by hardware.
- Send appropriate ACK/NAK/STALL handshakes.
- SETUP, IN, or OUT Token type identification. Set the appropriate bit once a valid token is received.
- Place valid received data in the appropriate endpoint FIFOs.
- Bit stuffing/unstuffing.
- Address checking. Ignore the transactions not addressed to the device.
- Endpoint checking. Check the endpoint's request from USB host, and set the appropriate bit of registers.

Firmware is required to handle the rest of the following tasks:

- Coordinate enumeration by decoding USB device requests.
- Fill and empty the FIFOs.
- Suspend/Resume coordination.
- Remote wake up function.
- Determine the right interrupt request of USB communication.

9.3 USB INTERRUPT

The USB function will accept the USB host command and generate the relative interrupts, and the program counter will go to 0x08 vector. Firmware is required to check the USB status bit to realize what request comes from the USB host.

The USB function interrupt is generated when:

- The endpoint 0 is set to accept a SETUP token.
- The device receives an ACK handshake after a successful read transaction (IN) from the host.
- If the endpoint is in ACK OUT modes, an interrupt is generated when data is received.
- The USB host send USB suspend request to the device.
- USB bus reset event occurs.
- The USB endpoints interrupt after a USB transaction complete is on the bus.
- The USB resume when the USB bus is placed in the suspend state.

The following examples show how to avoid the error of reading or writing the endpoint FIFOs and to do the right USB request routine according to the flag.

Example: Save the UDP0, UDP1, ACC and Status flag when interrupt request occurs. To avoid the error when read or write data in the endpoints FIFOs.

```
ORG 0x8
PUSH                            ; Save ACC and status flag
mov a, UDP0                     ; Save the UDP0 register value to UDP0_TEMP
mov UDP0_TEMP, a
mov a, UDP1                     ; Save the UDP1 register value to UDP1_TEMP
mov UDP1_TEMP, a
…
…
mov a, UDP0_TEMP
mov UDP0, a                     ; Load the UDP0_TEMP register value to UDP0
mov a, UDP1_TEMP
mov UDP1, a                     ; Load the UDP1_TEMP register value to UDP1
POP                             ; Load the ACC and status flag
RETI
```

Example: Defining USB Interrupt Request. The interrupt service routine is following ORG 8.

```
ORG 0x8
b0bts0    Ustatus.6             ; check USB bus reset request
jmp  _USB_Bus_Reset             ; Jump to USB bus reset routine
b0bts0    Ustatus.5             ; check USB suspend request
jmp  _USB_suspend               ; Jump to USB suspend routine
b0bts0    Ustatus.4             ; check EP0 SETUP Token
jmp  _EP0_setup                 ; Jump to EP0 SETUP Token routine.
```

```
b0bts0    Ustatus.3          ; check EP0 IN Token
jmp  _EP0_in                 ; Jump to EP0 IN routine.
b0bts0    Ustatus.2          ; check EP0 OUT Token.
jmp  _EP0_out                ; Jump to EP0 OUT routine.
b0bts0    Ustatus.1          ; check EP1's transaction is completed
jmp  _EP1_ACK                ; Jump to EP1's transaction routine.
b0bts0    Ustatus.0          ; check EP2's transaction is completed
jmp  _EP2_ACK                ; Jump to EP2's transaction routine.
RETI
```

9.4 USB ENUMERATION

A typical USB enumeration sequence is shown below.

1. The host computer sends a SETUP packet followed by a DATA packet to USB address 0 requesting the Device descriptor.
2. Firmware decodes the request and retrieves its Device descriptor from the program memory tables.
3. The host computer performs a control read sequence and Firmware responds by sending the Device descriptor over the USB bus, via the on-chip FIFO.
4. After receiving the descriptor, the host sends a SETUP packet followed by a DATA packet to address 0 assigning a new USB address to the device.
5. Firmware stores the new address in its USB Device Address Register after the no-data control sequence completes.
6. The host sends a request for the Device descriptor using the new USB address.
7. Firmware decodes the request and retrieves the Device descriptor from program memory tables.
8. The host performs a control read sequence and Firmware responds by sending its Device descriptor over the USB bus.
9. The host generates control reads from the device to request the Configuration and Report descriptors.
10. Once the device receives a Set Configuration request, its functions may now be used.
11. Firmware should take appropriate action for Endpoint 1~2 transactions, which may occur from this point.

9.5 USB REGISTERS

9.5.1 USB DEVICE ADDRESS REGISTER

The USB Device Address Register (UDA) contains a 7-bit USB device address and one bit to enable the USB function. This register is cleared during a reset, setting the USB device address to zero and disable the USB function.

0A0H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
UDA	UDE	UDA6	UDA5	UDA4	UDA3	UDA2	UDA1	UDA0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
After reset	0	0	0	0	0	0	0	0

Bit [6:0] **UDA [6:0]:** These bits must be set by firmware during the USB enumeration process (i.e., SetAddress) to the non-zero address assigned by the USB host.

Bit 7 **UDE: Device Function Enable.** This bit must be enabled by firmware to enable the USB device function. After the bit is set, the D- will pull up automatically to indicate the low speed device to the USB host.

0 = Disable USB device function.

1 = Enable USB device function.

9.5.2 USB STATUS REGISTER

The USB status register indicates the status of USB.

0A1H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
USTATUS		BUS_RST	SUSPEND	EP0_SETUP	EP0_IN	EP0_OUT	EP1 ACK	EP2 ACK
Read/Write		R	R	R/W	R/W	R/W	R/W	R/W
After reset		0	0	0	0	0	0	0

Bit 0 **EP2_ACK :** Endpoint 2's ACK Transaction. The bit is set whenever the endpoint 2 that completes with an ACK packet.

0 = The endpoint 2's transaction doesn't complete with an ACK.

1 = The endpoint 2's transaction complete with an ACK.

Bit 1 **EP1_ACK :** Endpoint 1's ACK Transaction. The bit is set whenever the endpoint 1 that completes with an ACK packet.

0 = The endpoint 1's transaction doesn't complete with an ACK.

1 = The endpoint 1's transaction complete with an ACK.

Bit 2 **EP0_OUT :** Endpoint 0 OUT Token Received.

0 = Endpoint 0 has no OUT token received.

1 = A valid OUT packet has been received. The bit is set to 1 after the last received packet in an OUT transaction.

Bit 3 **EP0_IN :** Endpoint 0 IN Token Received.

0 = Endpoint 0 has no IN token received.

1 = A valid IN packet has been received. The bit is set to 1 after the last received packet in an IN transaction.

Bit 4 **EP0_SETUP :** Endpoint 0 SETUP Token Received.

0 = Endpoint 0 has no SETUP token received.

1 = A valid SETUP packet has been received. The bit is set to 1 after the last received packet in an SETUP transaction. While the bit is set to 1, the HOST can not write any data in to EP0 FIFO. This prevents SIE from overwriting an incoming SETUP transaction before firmware has a chance to read the SETUP data.

Bit 5 **SUSPEND:** indicate USB suspend status.

0 = Non-suspend status. When MCU wakeup from sleep mode by USB resume wakeup request, the bit will changes from 1 to 0 automatically.

1 = Set to 1 by hardware when USB suspend request.

Bit 6 **BUS_RST:** USB bus reset.

0 = Non-USB bus reset.

1 = Set to 1 by hardware when USB bus reset request.

9.5.3 USB DATA COUNT REGISTER

The USB FIFO Selection bits and USB EP0 OUT token data counter.

0A2H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
UCTRL	FFS2	FFS1	FFS0	UEP0OC4	UEP0OC3	UEP0OC2	UEP0OC1	UEP0OC0
Read/Write	R	R	R	R/W	R/W	R/W	R/W	R/W
After reset	0	0	0	0	0	0	0	0

Bit [4:0] **UEP0C [4:0]:** USB endpoint 0 OUT token data counter.

Bit 5 **FFS0:** endpoint 0 FIFO selection control bit.

FFS0	Endpoint 0 OUT data	Endpoint 0 IN data
0	FIFO 1	FIFO 0
1	FIFO 0	FIFO 1

Bit 6 **FFS1:** endpoint 1 FIFO selection control bit.

FFS0	Endpoint 1 OUT data	Endpoint 1 IN data
0	FIFO 1	FIFO 0
1	FIFO 0	FIFO 1

Bit 7 **FFS2:** endpoint 2 FIFO selection control bit.

FFS0	Endpoint 2 OUT data	Endpoint 2 IN data
0	FIFO 1	FIFO 0
1	FIFO 0	FIFO 1

9.5.4 USB ENDPOINT 0 ENABLE REGISTER

An endpoint 0 (EP0) is used to initialize and control the USB device. EP0 is bi-directional (Control pipe), as the device, can both receive and transmit data, which provides to access the device configuration information and allows generic USB status and control accesses.

0A3H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
UE0R	-	UE0M1	UE0M0	-	UE0C3	UE0C	UE0C1	UE0C0
Read/Write	-	R/W	R/W	-	R/W	R/W	R/W	R/W
After reset	-	0	0	-	0	0	0	0

Bit [3:0] **UE0C [3:0]:** Indicate the number of data bytes in a transaction: For IN transactions, firmware loads the count with the number of bytes to be transmitted to the host from the endpoint 0 FIFO.

Bit [6:5] **UE0M [1:0]:** The endpoint 0 modes determine how the SIE responds to USB traffic that the host sends to the endpoint 0. For example, if the endpoint 0's mode bit is set to 00 that is NAK IN/OUT mode as shown in *Table,* The USB SIE will send NAK handshakes in response to any IN/OUT token set to the endpoint 0.

USB endpoint 0's mode table

UE0M1	UE0M0	IN/OUT Token Handshake
0	0	NAK
0	1	ACK
1	0	STALL
1	1	STALL

9.5.5 USB ENDPOINT 1 ENABLE REGISTER

The communication with the USB host using endpoint 1, endpoint 1's FIFO is implemented as 16 bytes of dedicated RAM. The endponit1 is an interrupt endpoint.

0A4H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
UE1R	UE1E	UE1M1	UE1M0	UE1C4	UE1C3	UE1C	UE1C1	UE1C0
Read/Write	R/W	R	R/W	R/W	R/W	R/W	R/W	R/W
After reset	0	0	0	0	0	0	0	0

Bit [4:0] **UE1C [4:0]:** Indicate the number of data bytes in a transaction: For IN transactions, firmware loads the count with the number of bytes to be transmitted to the host from the endpoint 1 FIFO.

Bit [6:5] **UE1M [1:0]:** The endpoint 1 modes determine how the SIE responds to USB traffic that the host sends to the endpoint 1. For example, if the endpoint 1's mode bit is set to 00 that is NAK IN/OUT mode as shown in *Table,* The USB SIE will send NAK handshakes in response to any IN/OUT token set to the endpoint 1.

USB endpoint 1's mode table

UE1M1	UE1M0	IN/OUT Token Handshake
0	0	NAK
0	1	ACK
1	0	STALL
1	1	STALL

Bit 7 **UE1E:** USB endpoint 1 function enable bit.

0 = disable USB endpoint 1 function.

1 = enable USB endpoint 1 function.

The following examples show how to do the right USB endpoints request routine according to the flag.

Example: Check the Endpoint 1's IN/OUT request

```
_EP1_CHECK:
        B0BTS0   USTATUS.1                ;Check if Endpoint 1 transaction is completed.
        CALL     _EP1_FUNCTION
        JMP      _EP1_CHECK

_EP1_FUNCTION:
WRITE_EP1:
        EP1_WR_RAM_addr_set #0x8          ;SET Endpoint 1 FIFO address
        EP1_WR_RAM_data  EP1_DATA         ;Write EP1_DATA to USB FIFO.
        EP1_WR_RAM_addr_add #0x1          ;FIFO address + 1
        EP1_WR_RAM_data  MOUSE_X_AXIS     ;Write MOUSE_X_AXIS to FIFO.
        …
        …
        MOV      a, #0xa8                 ;1.keep enable ep1
                                          ;2. Send 8 byte to USB host
                                          ;3. Ready to transfer, the ACK handshake
        B0MOV    UE1E, a
        RET
```

Example: Check the Endpoint 1's OUT request

```
_EP1_CHECK:
        B0BSET   UE1E.5                   ;Ready to receive data from USB host (OUT token)
        B0BTS0   USTATUS.1                ;Check if Endpoint 1 transaction is completed.
        CALL     _EP1_FUNCTION
        JMP      _EP1_CHECK

_EP1_FUNCTION:
READ_EP1:
        EP1_RD_RAM_addr_set #0x8          ;SET Endpoint 1 FIFO address
        EP1_RD_RAM_data                   ;Data move to ACC
        MOV      FIFO_DATA_0, A           ;Data move to FIFO_DATA_0
        EP1_RD_RAM_addr_add #0x1          ;FIFO address + 1
        EP1_RD_RAM_data                   ;Data move to ACC
        RET
```

9.5.6 USB ENDPOINT 2 ENABLE REGISTER

The communication with the USB host using endpoint 2, endpoint 2's FIFO is implemented as 16 bytes of dedicated RAM. The endpoint 2 is an interrupt endpoint.

0A5H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
UE2R	UE2E	UE2M1	UE2M0	UE2C4	UE2C3	UE2C	UE2C1	UE2C0
Read/Write	R/W	R	R/W	R/W	R/W	R/W	R/W	R/W
After reset	0	0	0	0	0	0	0	0

Bit [4:0] **UE2C [4:0]:** Indicate the number of data bytes in a transaction: For IN transactions, firmware loads the count with the number of bytes to be transmitted to the host from the endpoint 2 FIFO.

Bit [6:5] **UE2M [1:0]:** The endpoint 2 modes determine how the SIE responds to USB traffic that the host sends to the endpoint 2. For example, if the endpoint 2's mode bit is set to 00 that is NAK IN/OUT mode as shown in *Table,* The USB SIE will send NAK handshakes in response to any IN/OUT token set to the endpoint 2.

USB endpoint 2's mode table

UE2M1	UE2M0	IN/OUT Token Handshake
0	0	NAK
0	1	ACK
1	0	STALL
1	1	STALL

Bit 7 **UE2E:** USB endpoint 2 function enable bit.

0 = disable USB endpoint 2 function.

1 = enable USB endpoint 2 function.

9.5.7 USB DATA POINTER 0 REGISTER

FIFO 0's address pointer. Use the point to set the FIFO address for reading data from FIFO and writing data to FIFO.

0A6H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
UDP0			UDP05	UDP04	UDP03	UDP02	UDP01	UDP00
Read/Write	-	-	R/W	R/W	R/W	R/W	R/W	R/W
After reset	-	-	0	0	0	0	0	0

Address [07]~address [00]: data buffer for endpoint 0. Check the bit 5 of the UCTRL register (0XA2H) to select the right FIFO.

Address [17]~address [08]: data buffer for endpoint 1. Check the bit 6 of the UCTRL register (0xA2H) to select the right FIFO.

Address [27]~address [18]: data buffer for endpoint 2. Check the bit 7 of the UCTRL register (0xA2H) to select the right FIFO.

The following examples show how to do select the right FIFO address pointer.

Example 1. Set endpoint 0's FIFO address, when reading data from FIFO.

```
EP0_FIFO_READ_Address:
        B0BTS0  UCTRL.5
        JMP udp0_address_set            ;Go to set the UDP0 address
        JMP udp1_address_set            ;Go to set the UDP1 address

udp0_address_set:
        MOV     A, address
        MOV     UDP0, A
        JMP     user_define_routine

udp1_address_set:
        MOV     A, address
        MOV     UDP1, A
        JMP     user_define_routine
```

Example 2. Read data from EP0 FIFO.

```
EP0_FIFO_RD_data:
        B0BTS0  UCTRL.5                 ;check the bit to select the right FIFO
        JMP     read_endpoint0_FIFO_UDR0
        JMP     read_endpoint0_FIFO_UDR1

read_endpoint0_FIFO_UDR0:
        MOV     A, UDR0                 ; move FIFO's data to A
        JMP     user_define_routine

read_endpoint0_FIFO_UDR1:
        MOV     A, UDR1                 ;move FIFO's data to A
        JMP     user_define_routine
```

Example 3. Set endpoint 1's FIFO address, when writing data to FIFO.

```
EP1_FIFO_WRITE_Address:
        B0BTS1  UCTRL.6
        JMP     udp0_address_set        ;Go to set the UDP0 address
        JMP     udp1_address_set        ;Go to set the UDP1 address

udp0_address_set:
        MOV     A, address              ; set the address to UDP.
        MOV     UDP0, A
```

```
        JMP     user_define_routine

udp1_address_set:
        MOV     A, address              ; set the address to UDP.
        MOV     UDP1, A
        JMP     user_define_routine
```

Example 4. Write data to EP2 FIFO.

```
EP2_FIFO_WRITE_data:
        B0BTS1  UCTRL.7                 ;check the bit to select the right FIFO
        JMP     write_endpoint2_FIFO_UDR0
        JMP     write_endpoint2_FIFO_UDR1

write _endpoint2_FIFO_UDR0
        MOV     A, UDR0
        JMP     user_define_routine

write _endpoint2_FIFO_UDR1
        MOV     A, UDR1
        JMP     user_define_routine
```

9.5.8 USB DATA REGISTER

Store the data, which UDP0 point to.

0A7H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
UDR0	UDR07	UDR06	UDR05	UDR04	UDR03	UDR02	UDR01	UDR00
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
After reset	0	0	0	0	0	0	0	0

9.5.9 USB DATA POINTER 1 REGISTER

FIFO 1's address pointer. Use the point to set the FIFO address for reading data from FIFO and writing data to FIFO.

0A8H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
UDP1			UDP15	UDP14	UDP13	UDP12	UDP11	UDP10
Read/Write	-	-	R/W	R/W	R/W	R/W	R/W	R/W
After reset	-	-	0	0	0	0	0	0

Address [07]~address [00]: data buffer for endpoint 0. Check the bit 5 of the UCTRL register (0XA2H) to select the right FIFO.

Address [17]~address [08]: data buffer for endpoint 1. Check the bit 6 of the UCTRL register (0xA2H) to select the right FIFO.

Address [27]~address [18]: data buffer for endpoint 2. Check the bit 7 of the UCTRL register (0xA2H) to select the right FIFO.

9.5.10 USB DATA REGISTER

Store the data, which UDP1 point to.

0A9H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
UDR1	UDR17	UDR16	UDR15	UDR14	UDR13	UDR12	UDR11	UDR10
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
After reset	0	0	0	0	0	0	0	0

9.5.11 UPID REGISTER

Forcing bits allow firmware to directly drive the D+ and D– pins.

0AAH	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
UPID	Reserved	Reserved	-	-	-	UBDE	DDP	DDN
Read/Write	-	-	-	-	-	W	W	W
After reset	1	1	-	-	-	0	0	0

Bit 0 **DDN:** Drive D- on the USB bus.

0 = drive D- low.

1 = drive D- high.

Bit 1 **DDP:** drive D+ on the USB bus.

0 = drive D+ low.

1 = drive D+ high.

Bit 2 **UBDE:** Enable to direct drive USB bus.

0 = disable.

1 = enable

.

9.5.12 USB ENDPOINT 1 OUT TOKEN DATA BYTES COUNTER

Endpoint 1's OUT TOKEN DATA BYTES COUNTER.

0ABH	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
EP1OUT_CNT	-	-	-	UEP1OC4	UEP1OC3	UEP1OC2	UEP1OC1	UEP1OC0
Read/Write	-	-	-	R/W	R/W	R/W	R/W	R/W
After reset	-	-	-	0	0	0	0	0

Bit [4:0] **UEP1CX:** Bytes counter of EP1 token data. Reset by firmware.

9.5.13 USB ENDPOINT 2 OUT TOKEN DATA BYTES COUNTER

Endpoint 1's OUT TOKEN DATA BYTES COUNTER.

0ABH	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
EP2OUT_CNT	-	-	-	UEP2OC4	UEP2OC3	UEP2OC2	UEP2OC1	UEP2OC0
Read/Write	-	-	-	R/W	R/W	R/W	R/W	R/W
After reset	-	-	-	0	0	0	0	0

Bit [4:0] **UEP2CX:** Bytes counter of EP2 token data. Reset by firmware.

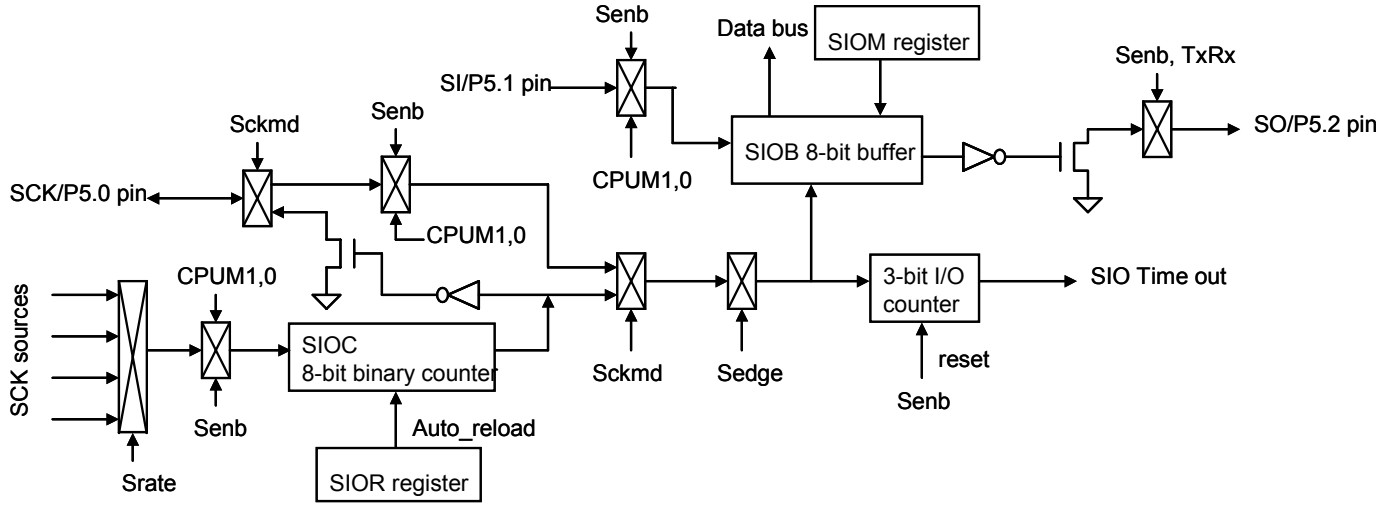
10 SERIAL INPUT/OUTPUT TRANSCEIVER

10.1 OVERVIEW

The SIO (serial input/output) transceiver allows high-speed synchronous data transfer between the SN8P2212 series MCU and peripheral devices or between several SN8P2212 devices. These peripheral devices may be Serial EEPROMs, shift registers, display drivers, etc. The SN8P2212 SIO features include the following:

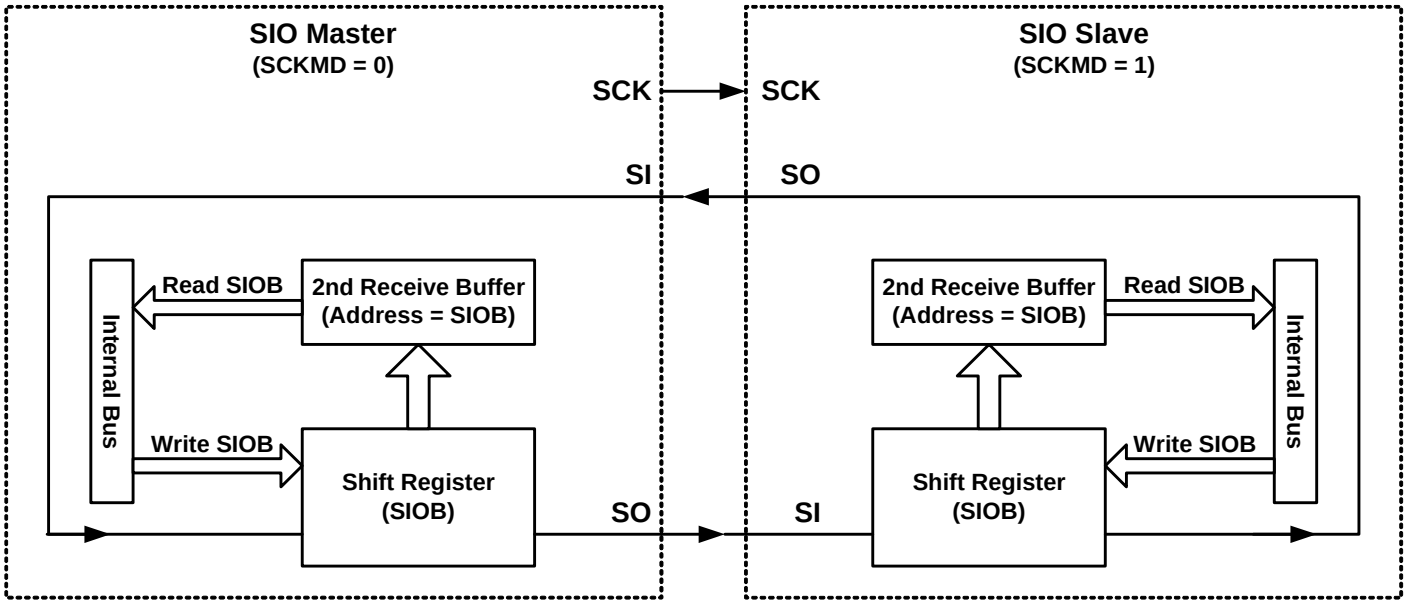
- **Full-duplex, 3-wire synchronous data transfer**
- **TX/RX or TX Only mode**
- **Master (SCK is clock output) or Slave (SCK is clock input) operation**
- **MSB first data transfer**
- **SO (P5.2) is programmable open-drain output pin for multiple salve devices application**
- **Two programmable bit rates (Only in master mode)**
- **End-of-Transfer interrupt**

The SIOM register can control SIO operating function, such as: transmit/receive, clock rate, transfer edge and starting this circuit. This SIO circuit will transmit or receive 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-bit data. After transferring 8-bit data, this circuit will be disabled automatically and re-transfer data by programming SIOM register.



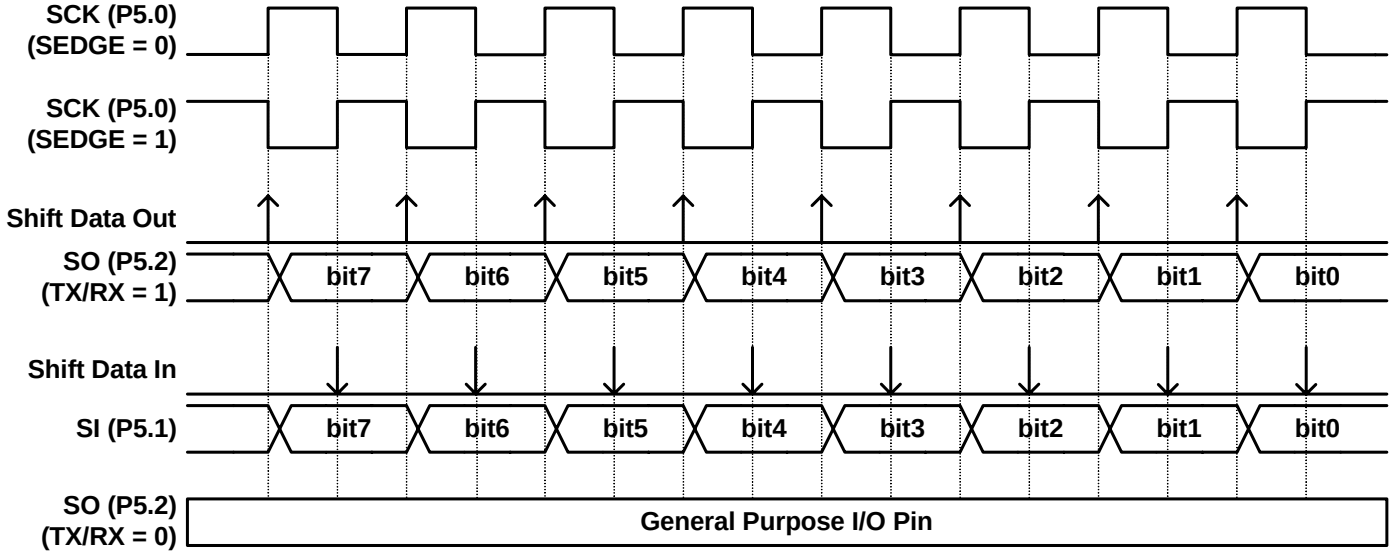
SIO Interface Circuit Diagram

The system is single-buffered in the transmit direction and double-buffered in the receive direction. This means that bytes to be transmitted cannot be written to the SIOB Data Register before the entire shift cycle is completed. When receiving data, however, a received byte must be read from the SIOB Data Register before the next byte has been completely shifted in. Otherwise, the first byte is lost. Following figure shows a typical SIO transfer between two SN8P2212 micro-controllers. Master MCU sends SCK for initial the data transfer. Both master and slave MCU must work in the same clock edge direction, and then both controllers would send and receive data at the same time.



SIO Data Transfer Diagram

The SIO data transfer timing as following figure:



SIO Data Transfer Timing

✷ ***Note: In any mode, SIO always transmit data in first SCK edge and receive data in second SCK edge.***

10.2 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	-	SCKMD	SEDGE	TXRX
Read/Write	R/W	R/W	R/W	R/W	-	R/W	R/W	R/W
After reset	0	0	0	0	-	0	0	0

Bit 7 **SENB:** SIO function control bit.
0 = disable (P5.0~P5.2 is general purpose port).
1 = enable (P5.0~P5.2 is SIO pins).

Bit 6 **START:** SIO progress control bit.
0 = End of transfer.
1 = progressing.

Bit [5:4] **SRATE1,0:** SIO's transfer rate select bit. **These 2-bits are workless when SCKMD=1.**
00 = fcpu.
01 = fcpu/32
10 = fcpu/16
11 = fcpu/8.

Bit 2 **SCKMD:** SIO's clock mode select bit.
0 = Internal.
1 = External.

Bit 1 **SEDGE:** SIO's transfer clock edge select bit.
0 = Falling edge.
1 = Rising edge.

Bit 0 **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.***
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	

10.3 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
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
After reset	0	0	0	0	0	0	0	0

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

10.4 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
Read/Write	W	W	W	W	W	W	W	W
After reset	0	0	0	0	0	0	0	0

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)

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 – (0.0002*895000)
= 256 – 179
= 77

Example: Master, duplex transfer and transmit data on rising edge

```
        MOV     A,TXDATA        ; Load transmitted data into SIOB register.
        B0MOV   SIOB,A
        MOV     A,#0FFH         ; Set SIO clock
        B0MOV   SIOR,A
        MOV     A,#10000001B    ; Setup SIOM and enable SIO function.
        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
```

Example: Master, duplex transfer and transmit data on falling edge

```
        MOV     A,TXDATA        ; Load transmitted data into SIOB register.
        B0MOV   SIOB,A
        MOV     A,#0FFH         ; Set SIO clock.
        B0MOV   SIOR,A
        MOV     A,#10000011B    ; Setup SIOM and enable SIO function.
        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
```

Example: Master, receive only and transmit data on rising edge

```
        MOV     A,#0FFH         ; Set SIO clock with auto-reload function.
        B0MOV   SIOR,A
        MOV     A,#10000000B    ; Setup SIOM and enable SIO function.
        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
```

Example: Master, receive only and transmit data on falling edge

```
        MOV     A,#0FFH         ; Set SIO clock.
        B0MOV   SIOR,A
        MOV     A,#10000010B    ; Setup SIOM and enable SIO function.
        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
```

Example: Slave, duplex transfer and transmit data on rising edge

```
        MOV     A,TXDATA        ; Load transfer data into SIOB register.
        B0MOV   SIOB,A
        MOV     A,# 10000101B   ; Setup SIOM and enable SIO function.
        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
```

Example: Slave, duplex transfer and transmit data on falling edge

```
        MOV     A,TXDATA        ; Load transfer data into SIOB register.
        B0MOV   SIOB,A
        MOV     A,# 10000111B   ; Setup SIOM and enable SIO function.
        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
```

Example: Slave, receive only and transmit data on rising edge

```
        MOV     A,# 10000100B   ; Setup SIOM and enable SIO function.
        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
```

Example: Slave, receive only and transmit data on falling edge

```
        MOV     A,# 10000110B   ; Setup SIOM and enable SIO function.
        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
```

11 INSTRUCTION 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 (bank 0)	-	-	√	1
V	B0MOV	M,A	M (bank 0) ← A	-	-	-	1
E	MOV	A,I	A ← I	-	-	-	1
	B0MOV	M,I	M ← I, "M" only supports 0x80~0x87 registers (e.g. PFLAG,R,Y,Z…)	-	-	-	1
	XCH	A,M	A ←→M	-	-	-	1+N
	B0XCH	A,M	A ←→M (bank 0)	-	-	-	1+N
	MOVC		R, A ← ROM [Y,Z]	-	-	-	2
	ADC	A,M	A ← A + M + C, if occur carry, then C=1, else C=0	√	√	√	1
A	ADC	M,A	M ← A + M + C, if occur carry, then C=1, else C=0	√	√	√	1+N
R	ADD	A,M	A (A + M, if occur carry, then C=1, else C=0	√	√	√	1
I	ADD	M,A	M (A + M, if occur carry, then C=1, else C=0	√	√	√	1+N
T	B0ADD	M,A	M (bank 0) (M (bank 0) + A, if occur carry, then C=1, else C=0	√	√	√	1+N
H	ADD	A,I	A (A + I, if occur carry, then C=1, else C=0	√	√	√	1
M	SBC	A,M	A (A - M - /C, if occur borrow, then C=0, else C=1	√	√	√	1
E	SBC	M,A	M (A - M - /C, if occur borrow, then C=0, else C=1	√	√	√	1+N
T	SUB	A,M	A (A - M, if occur borrow, then C=0, else C=1	√	√	√	1
I	SUB	M,A	M (A - M, if occur borrow, then C=0, else C=1	√	√	√	1+N
C	SUB	A,I	A ← A - I, if occur borrow, then C=0, else C=1	√	√	√	1
	AND	A,M	A ← A and M	-	-	√	1
L	AND	M,A	M ← A and M	-	-	√	1+N
O	AND	A,I	A ← A and I	-	-	√	1
G	OR	A,M	A ← A or M	-	-	√	1
I	OR	M,A	M ← A or M	-	-	√	1+N
C	OR	A,I	A ← A or I	-	-	√	1
	XOR	A,M	A ← A xor M	-	-	√	1
	XOR	M,A	M ← A xor M	-	-	√	1+N
	XOR	A,I	A ← A xor I	-	-	√	1
	SWAP	M	A (b3~b0, b7~b4) ←M(b7~b4, b3~b0)	-	-	-	1
P	SWAPM	M	M(b3~b0, b7~b4) ← M(b7~b4, b3~b0)	-	-	-	1+N
R	RRC	M	A ← RRC M	√	-	-	1
O	RRCM	M	M ← RRC M	√	-	-	1+N
C	RLC	M	A ← RLC M	√	-	-	1
E	RLCM	M	M ← RLC M	√	-	-	1+N
S	CLR	M	M ← 0	-	-	-	1
S	BCLR	M.b	M.b ← 0	-	-	-	1+N
	BSET	M.b	M.b ← 1	-	-	-	1+N
	B0BCLR	M.b	M(bank 0).b ← 0	-	-	-	1+N
	B0BSET	M.b	M(bank 0).b ← 1	-	-	-	1+N
	CMPRS	A,I	ZF,C ← A - I, If A = I, then skip next instruction	√	-	√	1 + S
B	CMPRS	A,M	ZF,C ← A – M, If A = M, then skip next instruction	√	-	√	1 + S
R	INCS	M	A ← M + 1, If A = 0, then skip next instruction	-	-	-	1+ S
A	INCMS	M	M ← M + 1, If M = 0, then skip next instruction	-	-	-	1+N+S
N	DECS	M	A ← M - 1, If A = 0, then skip next instruction	-	-	-	1+ S
C	DECMS	M	M ← M - 1, If M = 0, then skip next instruction	-	-	-	1+N+S
H	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 ACC and PFLAG (except NT0, NPD bit) into buffers.	-	-	-	1
C	POP		To pop ACC and PFLAG (except NT0, NPD bit) from buffers.	√	√	√	1
	NOP		No operation	-	-	-	1

Note: ***1. "M" is system register or RAM. If "M" is system registers then "N" = 0, otherwise "N" = 1.***
2. If branch condition is true then "S = 1", otherwise "S = 0".

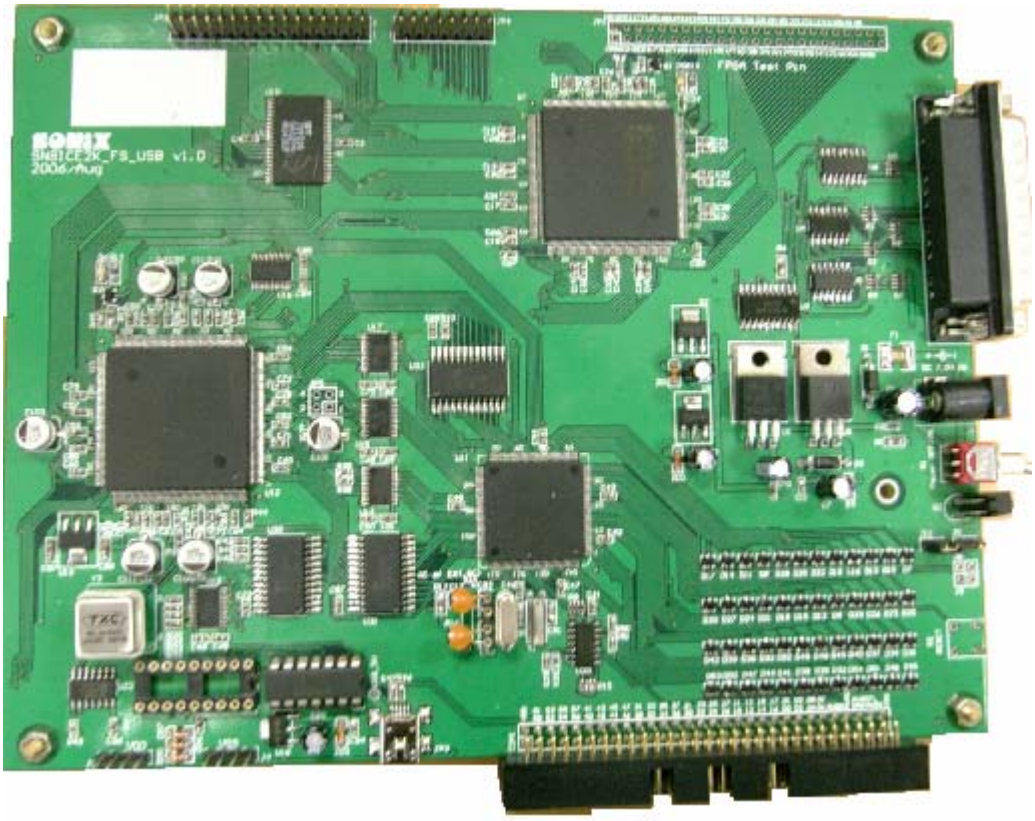
12 DEVELOPMENT TOOL

SONIX provides ICE (in circuit emulation), IDE (Integrated Development Environment), EV-kit and firmware library for USB application development. ICE and EV-kit are external hardware device and IDE is a friendly user interface for firmware development and emulation.

12.1 ICE (In Circuit Emulation)

The ICE called “SN8ICE2K-FS_USB_1.0”

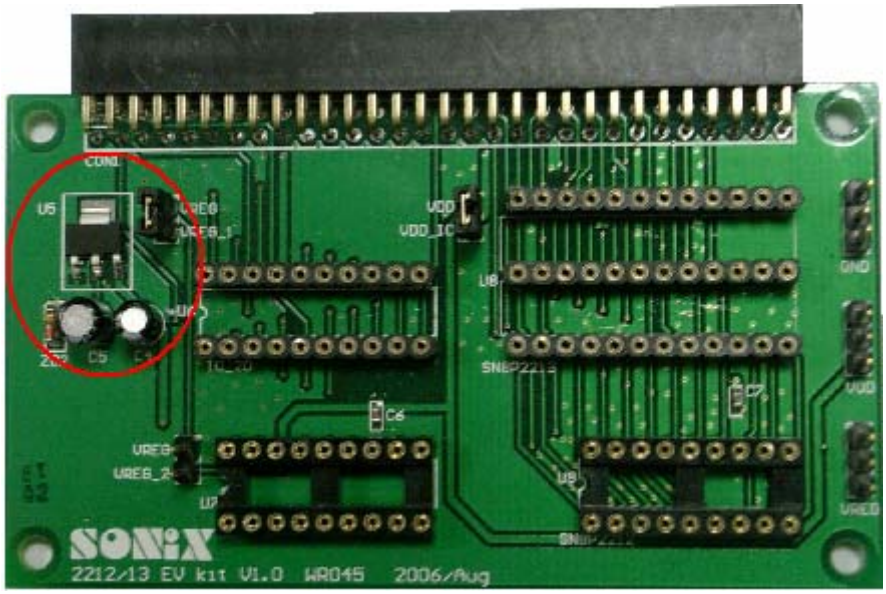
- **Support SN8P2212 serial microcontroller full function emulation.**
- **Not support EZ-Writer operation.**



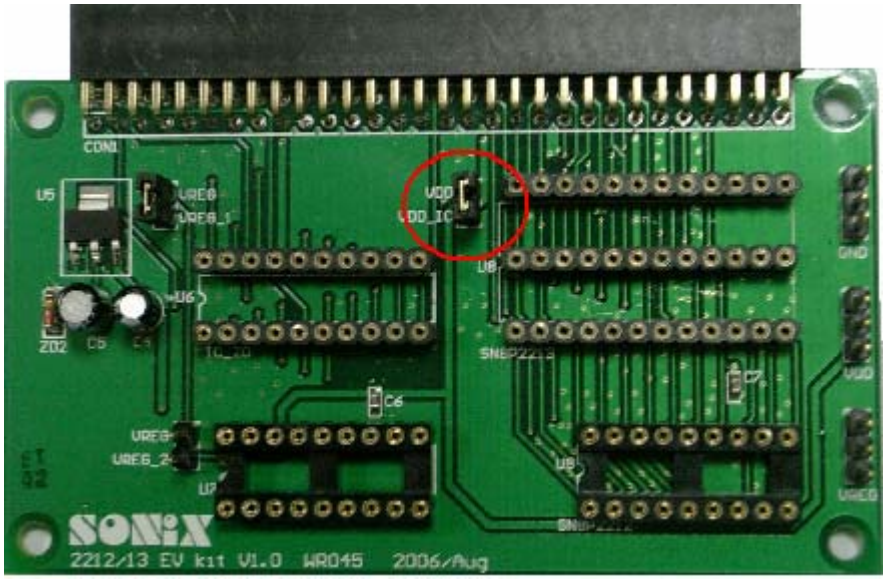
12.2 SN8P2213 EV-kit

SN8P2213 EV-kit includes ICE interface, GPIO interface and VREG 3.3V power supply.

- **ICE interface: Level shift peripheral circuit.**
- **3.3V power supply: Use 3.3V LDO to supply 3.3V power for VREG pin.**



- **Enable/Disable VDD power.**



13 ELECTRICAL CHARACTERISTIC

13.1 ABSOLUTE MAXIMUM RATING

Supply voltage (Vdd)..... - 0.3V ~ 6.0V
Input in voltage (Vin)..... Vss – 0.2V ~ Vdd + 0.2V
Operating ambient temperature (Topr)
SN8P2212/121/13P, SN8P2212/121/13S, SN8P22121/13X..... 0°C ~ + 70°C
Storage ambient temperature (Tstor) ..... –30°C ~ + 125°C

13.2 ELECTRICAL CHARACTERISTIC

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

PARAMETER	SYM.	DESCRIPTION		MIN.	TYP.	MAX.	UNIT
Operating voltage	Vdd1	Normal mode except USB transmitter specifications, Vpp = Vdd		3.0	5	5.5	V
	Vdd2	USB mode		4.25	5	5.25	V
RAM Data Retention voltage	Vdr			-	1.5*	-	V
Vdd rise rate	Vpor	Vdd rise rate to ensure power-on reset		0.05	-	-	V/ms
Input Low Voltage	ViL1	All input ports		Vss	-	0.3Vdd	V
	ViL2	Reset pin		Vss	-	0.2Vdd	V
Input High Voltage	ViH1	All input ports		0.7Vdd	-	Vdd	V
	ViH2	Reset pin		0.9Vdd	-	Vdd	V
Reset pin leakage current	Ilekg	Vin = Vdd		-	-	2	uA
I/O port pull-up resistor	Rup	Vin = Vss , Vdd = 5V		50	100	150	KΩ
I/O port input leakage current	Ilekg	Pull-up resistor disable, Vin = Vdd		-	-	2	uA
I/O output source current	IoH	Vop = Vdd – 0.5V		10	12*		mA
sink current	IoL	Vop = Vss + 0.5V		10	15*		
INTn trigger pulse width	Tint0	INT0 interrupt request pulse width		2/fcpu	-	-	cycle
Regulator current	IVREG	Max Regulator Output Current, Vcc > 4.35 volt with 10uF to GND				60	mA
Regulator GND current	IVREGnl	No loading. Reg pin output 3.3V ((Regulator enable)				150	uA
Regulator Output voltage	Vreg1	VCC > 4.35V, 0 < temp < 40°C, IVREG ≦ 60 mA with 10uF to GND		3.0		3.6	V
	Vreg2	VCC > 4.35V, 0 < temp < 40°C, IVREG ≦ 25 mA with 10uF to GND		3.1		3.6	V
Supply Current (Regulator disable)	Idd1	normal Mode (No loading, Fcpu = Fosc/4)	Vdd= 5V, 4Mhz	- -	2.5	5	mA
	Idd2	Slow Mode (Internal low RC)	Vdd= 5V, 32Khz	-	20	40	uA
	Idd3	Sleep Mode	Vdd= 5V	-	10	20	uA
	Idd4	Green Mode (No loading, Fcpu = Fosc/4 Watchdog Disable)	Vdd= 5V, 4Mhz	-	0.6	1.2	mA
			Vdd=5V, ILRC 32Khz	-	15	30	uA
LVD Voltage	Vdet0	Low voltage reset level.		2.0	2.4	2.9	V

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

14 OTP PROGRAMMING PIN

14.1 THE PIN ASSIGNMENT OF EASY WRITER TRANSITION BOARD SOCKET

Easy Writer JP1/JP2

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

JP1 for MP transition board

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

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

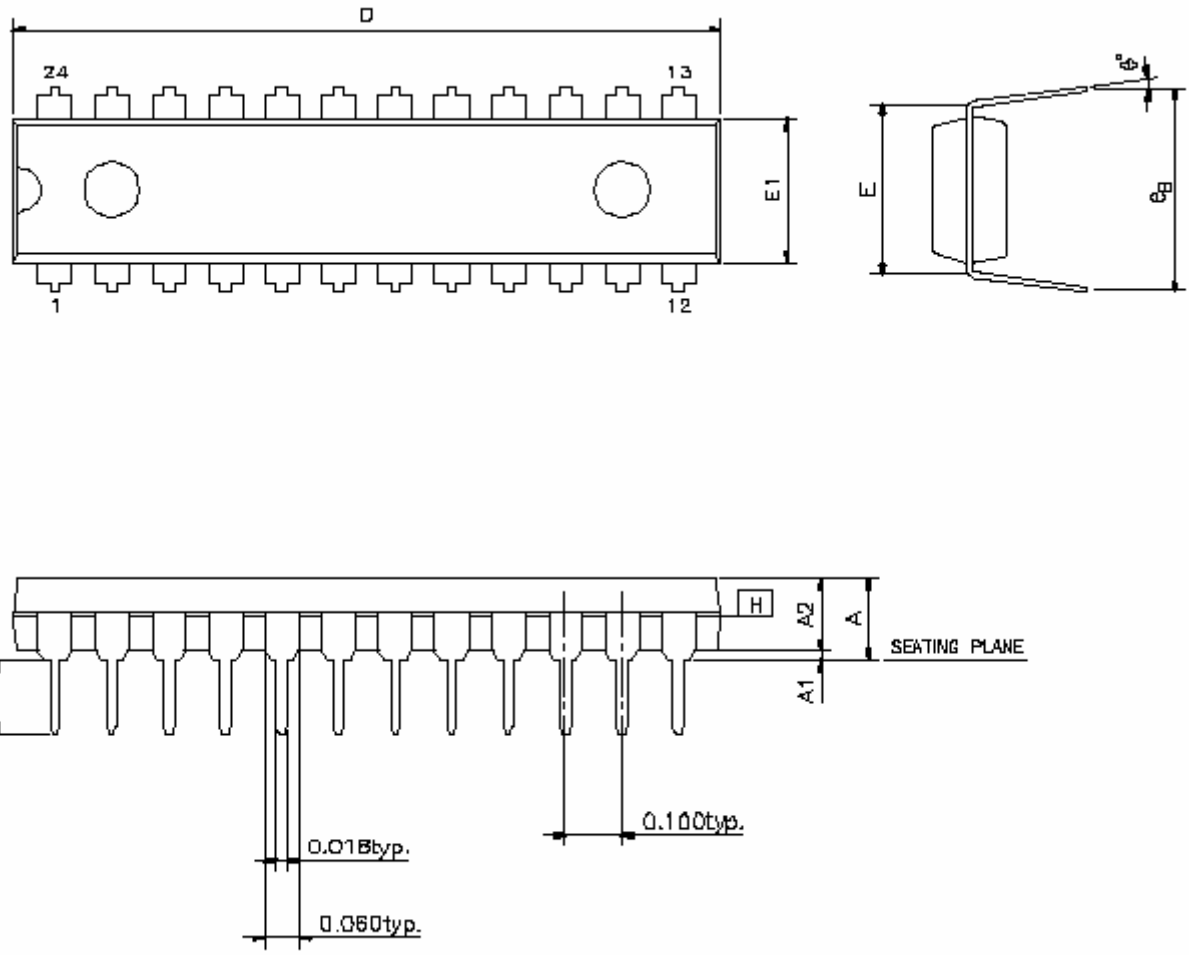
JP3 for MP transition board

14.2 PROGRAMMING PIN MAPPING

Programming Information of SN8P2213 Series									
Chip Name		SN8P2213K/S/X		SN8P22121P/S/X		SN8P2212P/S			
EZ Writer / MP Writer Connector		OTP IC / JP3 Pin Assigment							
Number	Name	Number	Pin	Number	Pin	Number	Pin		
1	VDD	19	VDD	17	VDD	16	VDD		
2	GND	15	VSS	13	VSS	12	VSS		
3	CLK	6	P5.0	6	P5.0	6	P5.0		
4	CE	-	-	-	-	-	-		
5	PGM	3	P1.0	3	P1.0	3	P1.0		
6	OE	4	P5.1	4	P5.1	4	P5.1		
7	D1	-	-	-	-	-	-		
8	D0	-	-	-	-	-	-		
9	D3	-	-	-	-	-	-		
10	D2	-	-	-	-	-	-		
11	D5	-	-	-	-	-	-		
12	D4	-	-	-	-	-	-		
13	D7	-	-	-	-	-	-		
14	D6	-	-	-	-	-	-		
15	VDD	-	-	-	-	-	-		
16	VPP	12	RST	10	RST	9	RST		
17	HLS	-	-	-	-	-	-		
18	RST	-	-	-	-	-	-		
19	-	-	-	-	-	-	-		
20	ALSB/PDB	2	P1.1	2	P1.1	2	P1.1		

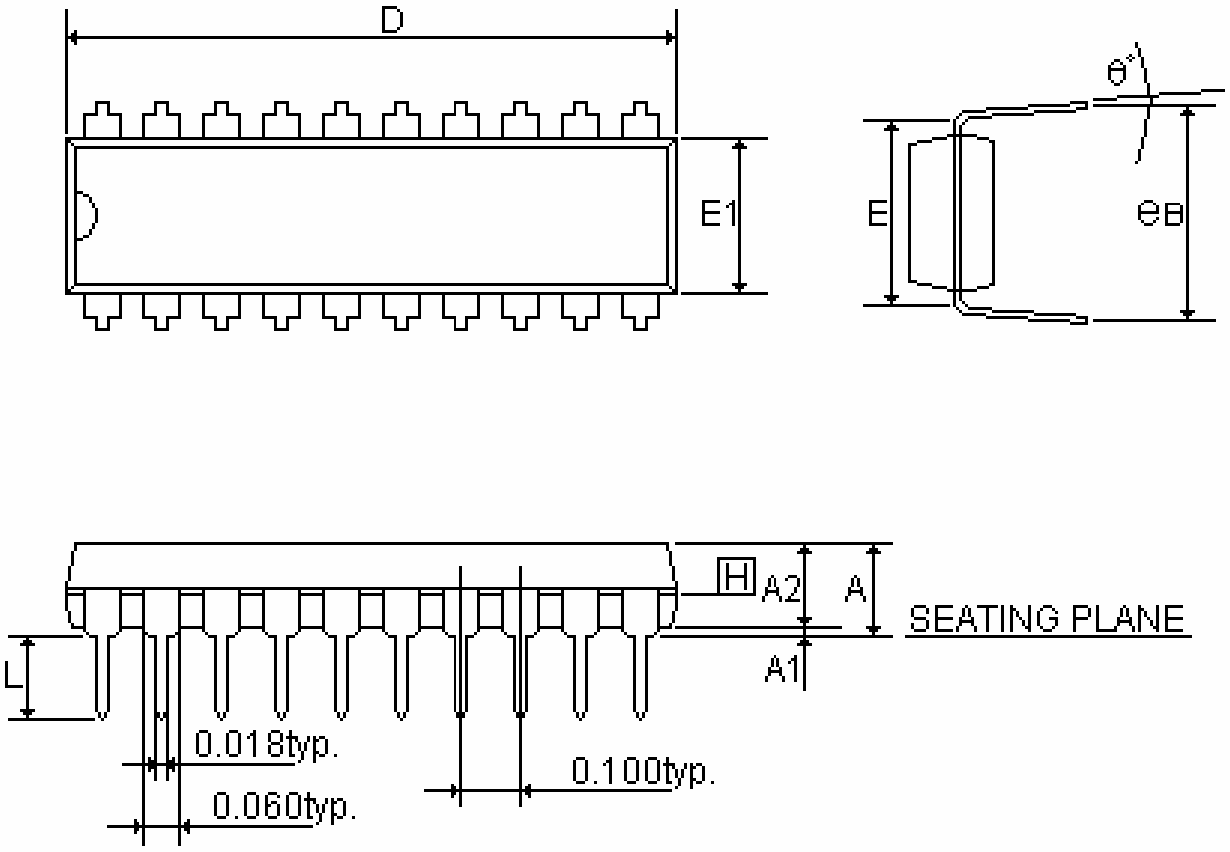
15 PACKAGE INFORMATION

15.1 SK-DIP 24 PIN



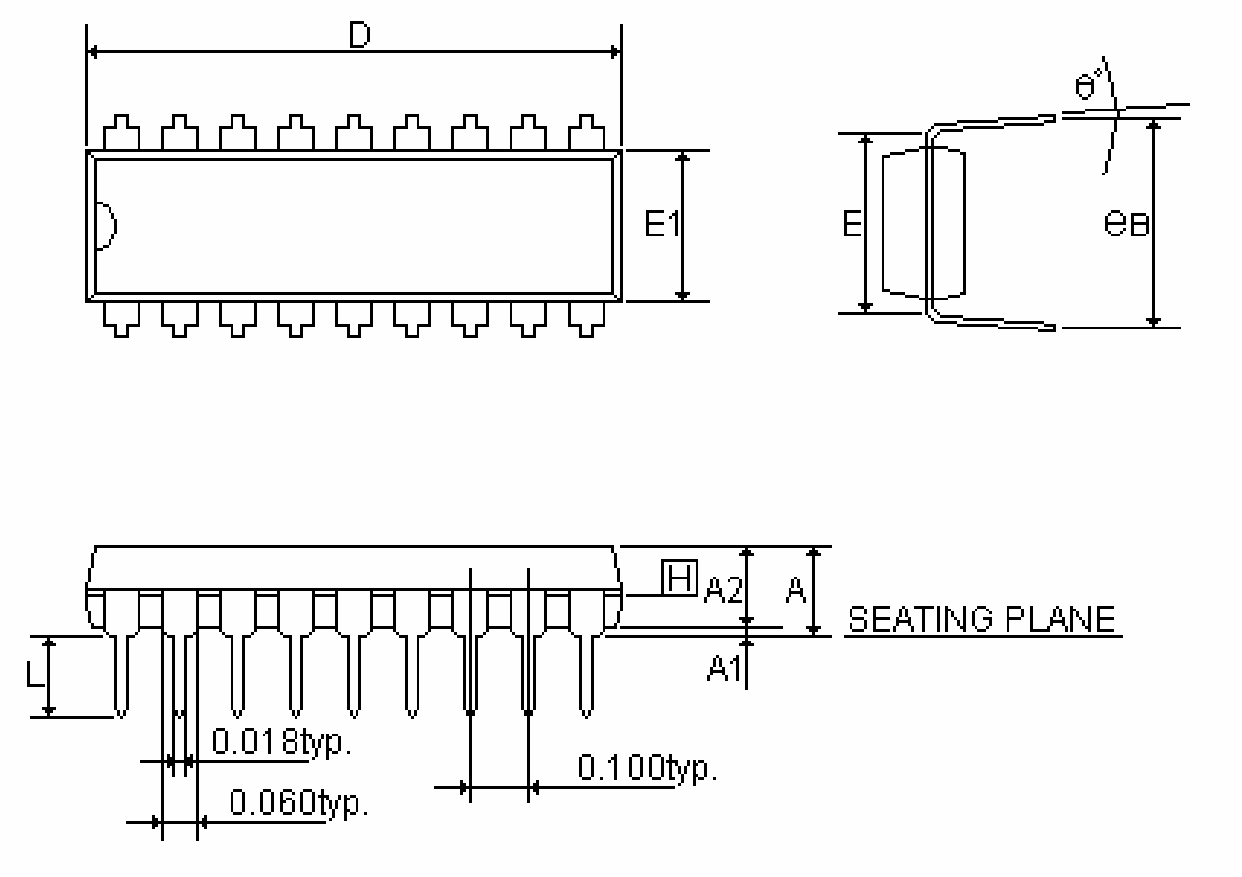
SYMBOLS	MIN	NOR	MAX	MIN	NOR	MAX
	(inch)			(mm)		
A	-	-	0.210	-	-	5.334
A1	0.015	-	-	0.381	-	-
A2	0.125	0.130	0.135	3.175	3.302	3.429
D	1.230	1.250	1.280	31.242	31.75	32.51
E	0.300 BSC			7.62 BSC		
E1	0.252	0.258	0.263	6.4	6.553	5.994
L	0.115	0.130	0.150	2.921	3.302	3.810
eB	0.335	0.355	0.375	8.509	9.017	9,525
θ°	0°	7°	15°	0°	7°	15°

15.2 P-DIP 20 PIN



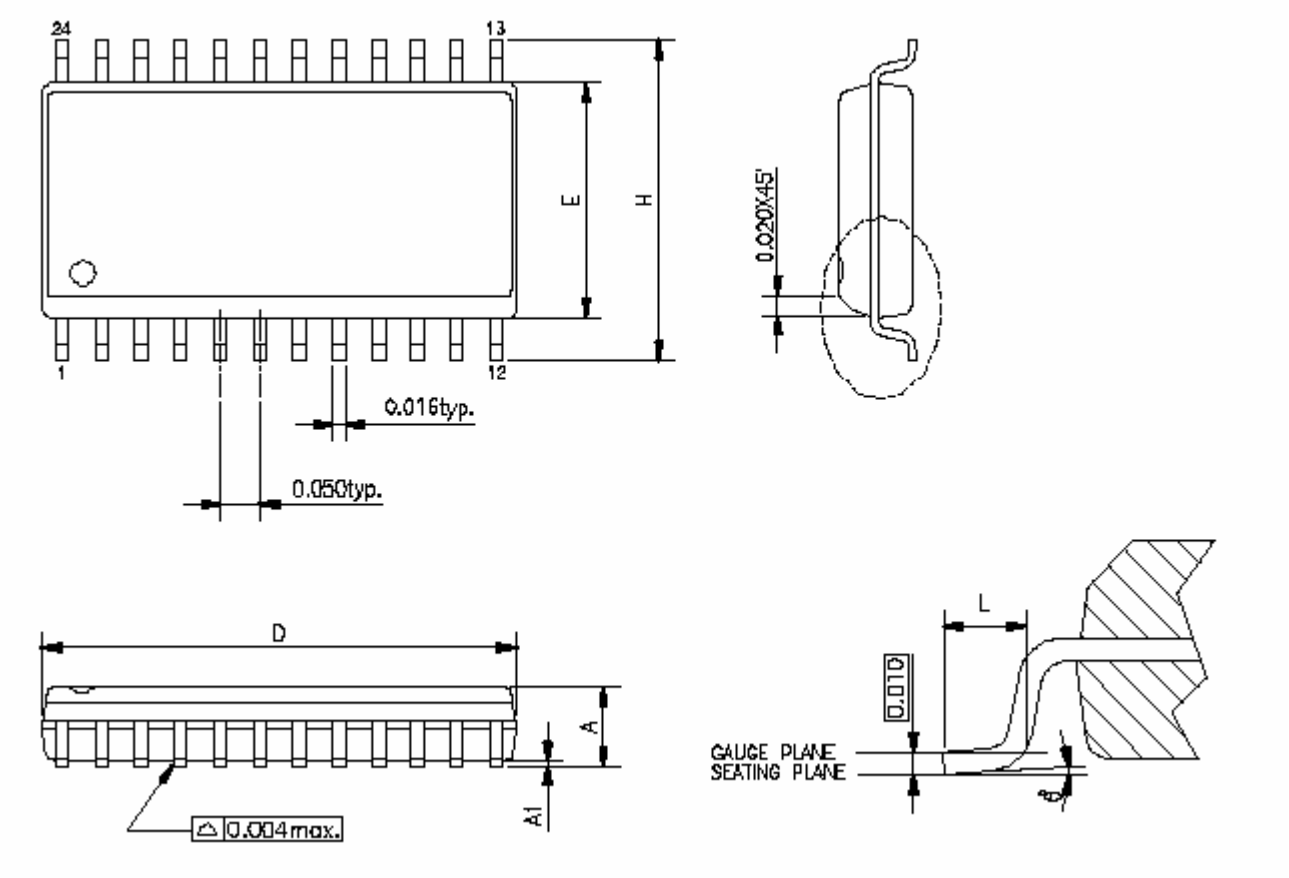
SYMBOLS	MIN	NOR	MAX	MIN	NOR	MAX
	(inch)			(mm)		
A	-	-	0.210	-	-	5.334
A1	0.015	-	-	0.381	-	-
A2	0.125	0.130	0.135	3.175	3.302	3.429
D	0.980	1.030	1.060	24.892	26.162	26.924
E		0.300			7.620	
E1	0.245	0.250	0.255	6.223	6.350	6.477
L	0.115	0.130	0.150	2.921	3.302	3.810
eB	0.335	0.355	0.375	8.509	9.017	9.525
θ°	0°	7°	15°	0°	7°	15°

15.3 P-DIP 18 PIN



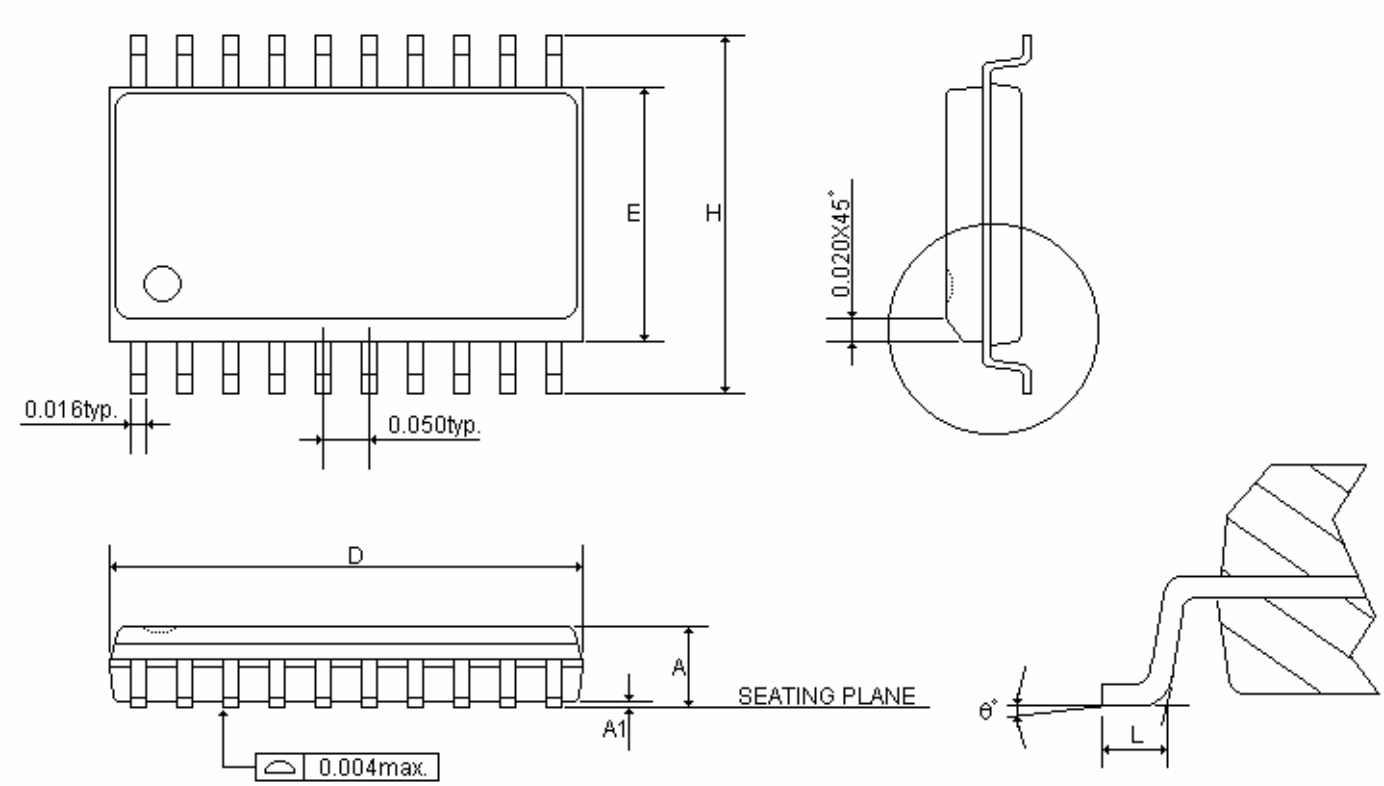
SYMBOLS	MIN	NOR	MAX	MIN	NOR	MAX
	(inch)			(mm)		
A	-	-	0.210	-	-	5.334
A1	0.015	-	-	0.381	-	-
A2	0.125	0.130	0.135	3.175	3.302	3.429
D	0.880	0.900	0.920	22.352	22.860	23.368
E		0.300			7.620	
E1	0.245	0.250	0.255	6.223	6.350	6.477
L	0.115	0.130	0.150	2.921	3.302	3.810
eB	0.335	0.355	0.375	8.509	9.017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

15.4 SOP 24 PIN



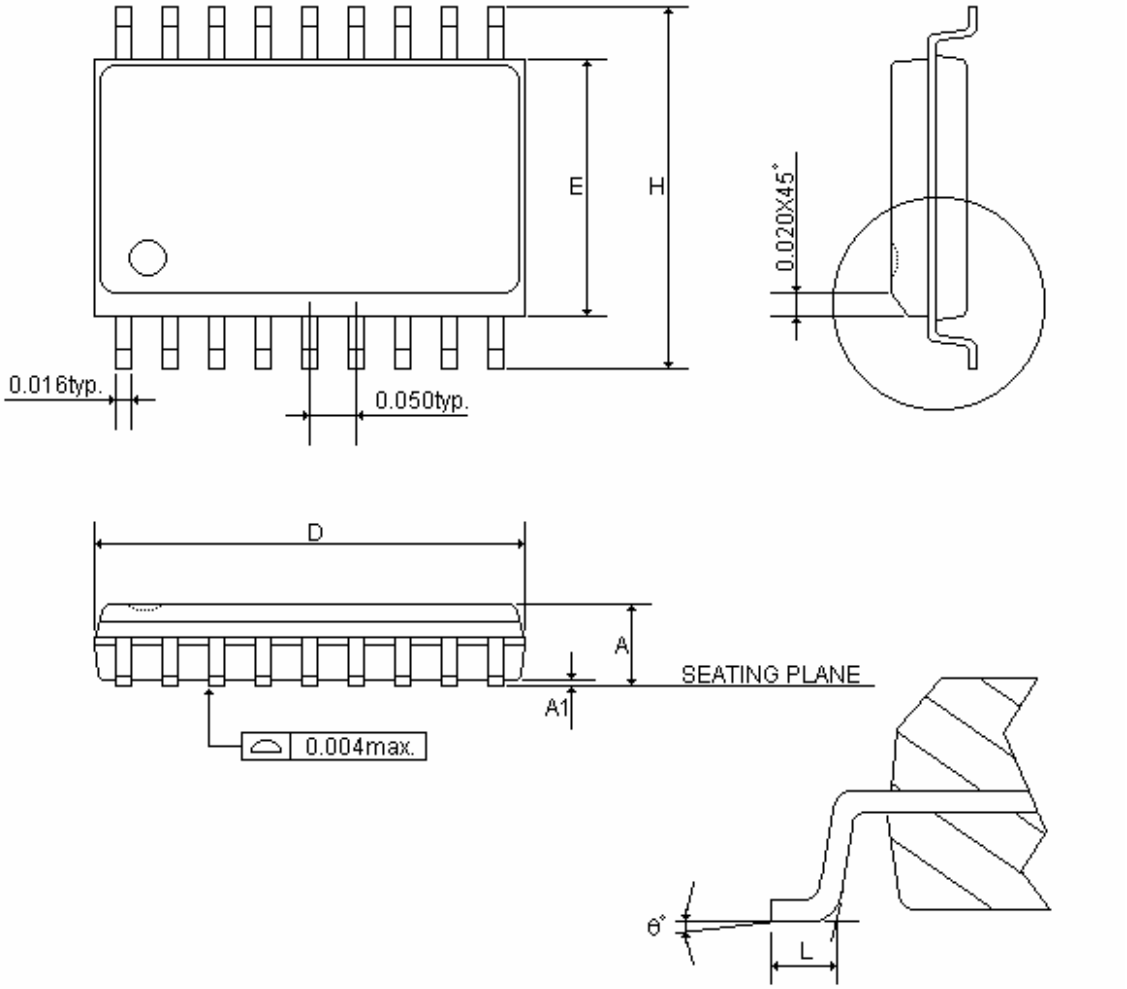
SYMBOLS	MIN	NOR	MAX	MIN	NOR	MAX
	(inch)			(mm)		
A	-	-	0.104	-	-	2.642
A1	0.004	-	-	0.102	-	-
D	0.599	0.600	0.624	15.214	15.24	15.84
E	0.291	0.295	0.299	7.391	7.493	7.595
H	0.394	0.407	0.419	10.008	10.337	10.643
L	0.016	0.035	0.050	0.406	0.889	1.270
θ°	0°	4°	8°	0°	4°	8°

15.5 SOP 20 PIN



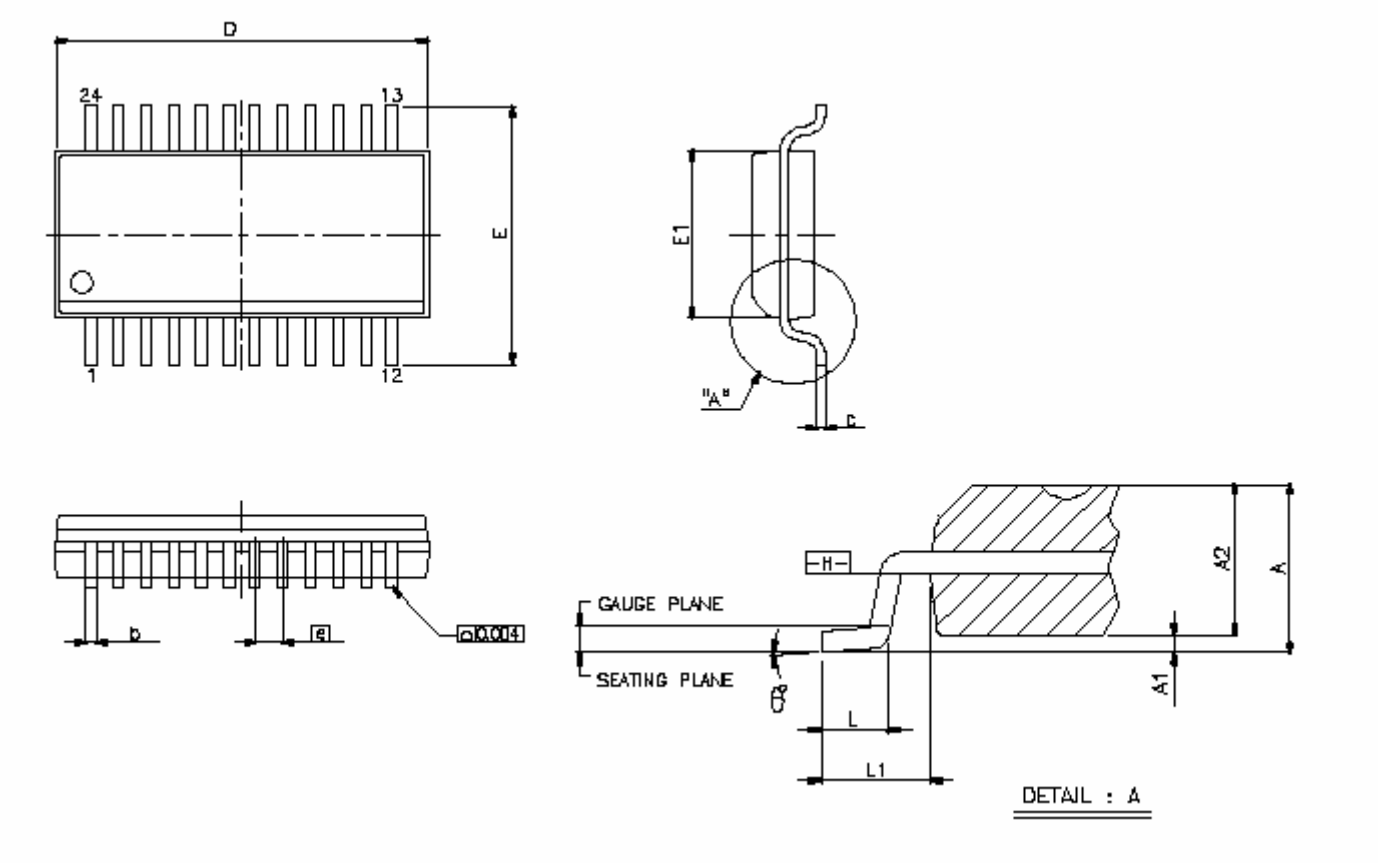
SYMBOLS	MIN	NOR	MAX	MIN	NOR	MAX
	(inch)			(mm)		
A	0.093	0.099	0.104	2.362	2.502	2.642
A1	0.004	0.008	0.012	0.102	0.203	0.305
D	0.496	0.502	0.508	12.598	12.751	12.903
E	0.291	0.295	0.299	7.391	7.493	7.595
H	0.394	0.407	0.419	10.008	10.325	10.643
L	0.016	0.033	0.050	0.406	0.838	1.270
θ°	0°	4°	8°	0°	4°	8°

15.6 SOP 18 PIN



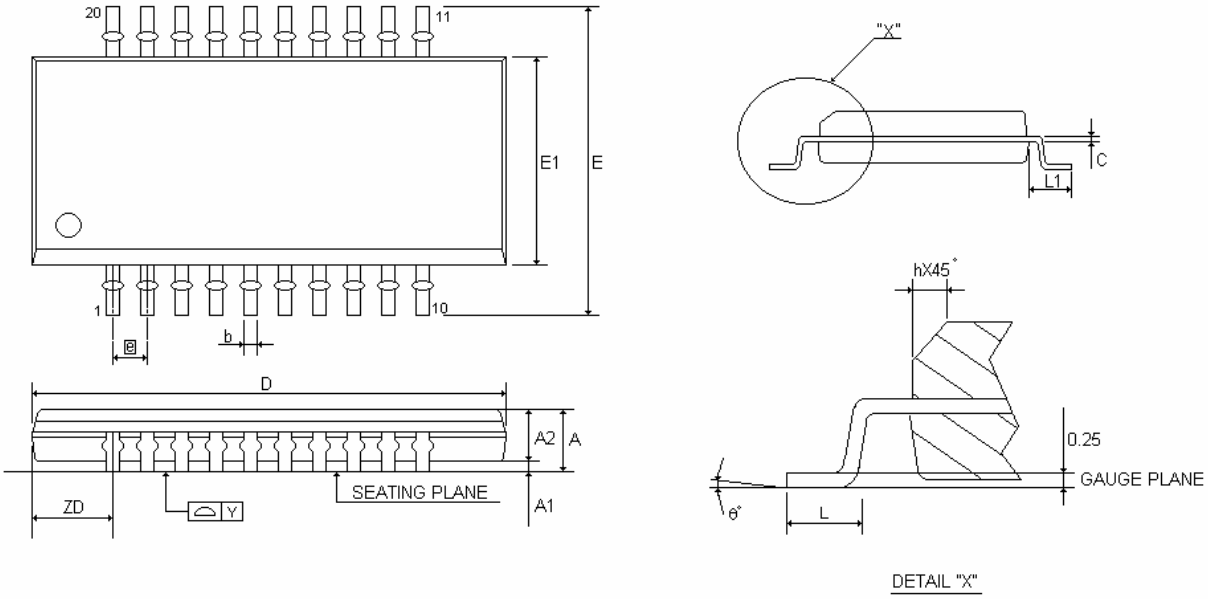
SYMBOLS	MIN	NOR	MAX	MIN	NOR	MAX
	(inch)			(mm)		
A	0.093	0.099	0.104	2.362	2.502	2.642
A1	0.004	0.008	0.012	0.102	0.203	0.305
D	0.447	0.455	0.463	11.354	11.557	11.760
E	0.291	0.295	0.299	7.391	7.493	7.595
H	0.394	0.407	0.419	10.008	10.325	10.643
L	0.016	0.033	0.050	0.406	0.838	1.270
θ°	0°	4°	8°	0°	4°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

15.7 SSOP 24 PIN



SYMBOLS	MIN	NOR	MAX	MIN	NOR	MAX
	(inch)			(mm)		
A	0.053	0.064	0.069	1.346	1.625	1.752
A1	0.004	0.006	0.010	0.101	0.152	0.254
A2	-	-	0.059	-	-	1.499
D	0.337	0.341	0.344	8.559	8.661	8.737
E	0.228	0.236	0.244	5.791	5.994	6.197
E1	0.150	0.154	0.157	3.81	3.911	3.987
b	0.008	-	0.012	0.203	-	0.304
C	0.007	-	0.010	0.177	-	0.254
[e]	0.025 BASIC			0.635 BASIC		
L	0.016	0.025	0.050	0.406	0.635	1.27
L1	0.041 BASIC			1.041 BASIC		
θ°	0°	-	8°	0°	-	8°

15.8 SSOP 20 PIN



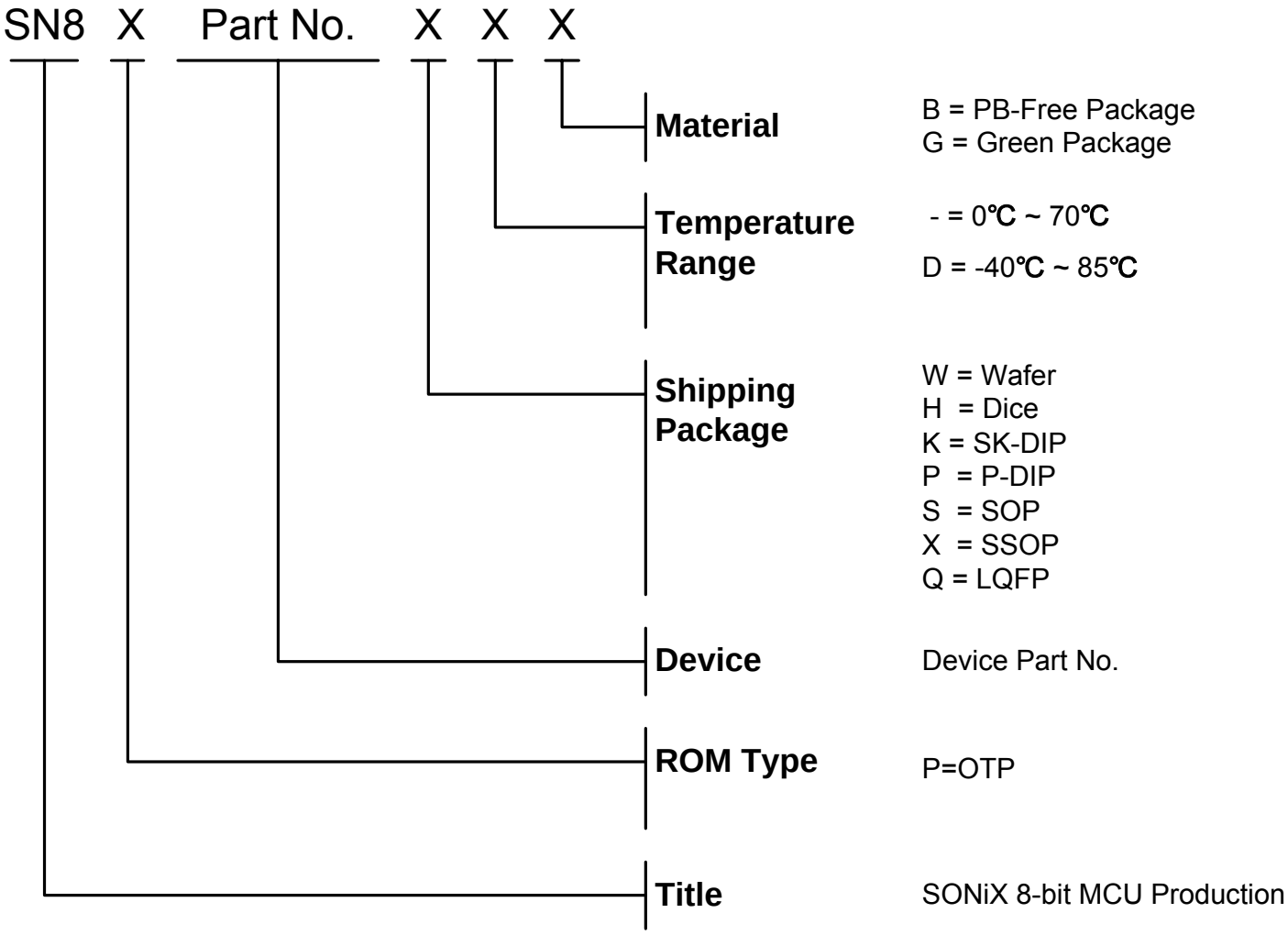
SYMBOLS	MIN	NOR	MAX	MIN	NOR	MAX
	(inch)			(mm)		
A	0.053	0.063	0.069	1.350	1.600	1.750
A1	0.004	0.006	0.010	0.100	0.150	0.250
A2	-	-	0.059	-	-	1.500
b	0.008	0.010	0.012	0.200	0.254	0.300
c	0.007	0.008	0.010	0.180	0.203	0.250
D	0.337	0.341	0.344	8.560	8.660	8.740
E	0.228	0.236	0.244	5.800	6.000	6.200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

16 Marking Definition

16.1 INTRODUCTION

There are many different types in Sonix 8-bit MCU production line. This note listed the production definition of all 8-bit MCU for order or obtain information. This definition is only for Blank OTP MCU.

16.2 MARKING INDETIFICATION SYSTEM



16.3 MARKING EXAMPLE

Name	ROM Type	Device	Package	Temperature	Material
SN8P2213KB	OTP	2213	SK-DIP	0℃~70℃	PB-Free Package
SN8P2213SB	OTP	2213	SOP	0℃~70℃	PB-Free Package
SN8P22121XB	OTP	22121	SSOP	0℃~70℃	PB-Free Package
SN8P2212PG	OTP	2212	P-DIP	0℃~70℃	Green Package
SN8P2212SG	OTP	2212	SOP	0℃~70℃	Green Package
SN8P22121XG	OTP	22121	SSOP	0℃~70℃	Green Package
SN8P2213W	OTP	2213	Wafer	0℃~70℃	-
SN8P2213H	OTP	2213	Dice	0℃~70℃	-

16.4 DATECODE SYSTEM