SN8P2602A

USER'S MANUAL

Version 1.2

SONiX 8-Bit Micro-Controller

AMENDMENT HISTORY

Version	Date	Description
VER 0.1	Jan. 2004	Preliminary Version 0.1 first issue
VER 0.2	Jan. 2004	Preliminary Version 0.2. Add body upgrade table.
VER 0.3	Apl. 2004	Preliminary Version 0.3. New form. Add AP-Notice and S8KD-2 ICE emulation description.
VER 0.4	July. 2004	1. The first instruction at ORG 8 must be “JMP” or “NOP”. 2. Modify the migration table. 3. Change the reset value of Y, Z, R registers from 00H to unknown. 4. In the operating ambient temperature, insert “D” after package type to indicate the temperature range is –40°C ~ + 85°C. 5. Modify some electrical characteristics: ViL, ViH, Ilekg, Rup, IoH. 6. Change the wakeup time from “2048” to “4096”. 7. Add “Development Tool Version” section in “Application Notice” chapter. 8. Change chapter name “TRANSITION SOCKET” to “OTP PROGRAMMING PIN” and modify the contents. 9. Remove “Template Code” chapter.
VER 0.5	July. 2004	Change the description of ”S” parameter in all instruction cycle related explanation.
VER 0.6	Aug. 2004	Modify T0 green mode wakeup setting example to remove “b0bset FT0GN” instruction.
VER 1.0	Dec. 2004	1. Add T0, TC0, PWM application notices. 2. T0C doesn’t support read and modify write instructions. 3. Modify operating mode, stack, reset, I/O diagrams. 4. Modify timer, system clock descriptions. 5. Modify system clock description. 6. Remove ORG4~7 limitation.
VER 1.1	Jun. 2005	1. Re-arrange partial edition layout. 2. Strongly recommend using SN8ICE-2K ICE to emulate SN8P2602A. SN8IDE V1.99S 3. VDD Brown-Out reset circuit. 4. VDD Working Voltage vs. Frequency graphs.
VER 1.2	Dec 2005	1. Modify Topr value. 2. Modify P7 Programmable open-drain. 3. Modify Brown-Out Reset description 4. Modify M2IDE 1.07 5. Remove power consumption(Pc) 6. Remove Noise Filter Enable Working Voltage. 7. Modify ELECTRICAL CHARACTERISTIC.

Table of Content

AMENDMENT HISTORY 2

1 PRODUCT OVERVIEW 7

1.1 FEATURES 7
1.2 SYSTEM BLOCK DIAGRAM 9
1.3 PIN ASSIGNMENT 10
1.4 PIN DESCRIPTIONS 11
1.5 PIN CIRCUIT DIAGRAMS 12

2 CENTRAL PROCESSOR UNIT (CPU) 13

2.1 MEMORY MAP 13
2.1.1 PROGRAM MEMORY (ROM) 13
2.1.1.1 RESET VECTOR (0000H) 14
2.1.1.2 INTERRUPT VECTOR (0008H) 15
2.1.1.3 LOOK-UP TABLE DESCRIPTION 17
2.1.1.4 JUMP TABLE DESCRIPTION 19
2.1.1.5 CHECKSUM CALCULATION 21
2.1.2 CODE OPTION TABLE 22
2.1.3 DATA MEMORY (RAM) 23
2.1.4 SYSTEM REGISTER 24
2.1.4.1 SYSTEM REGISTER TABLE 24
2.1.4.2 BIT DEFINITION of SYSTEM REGISTER 25
2.1.4.3 ACCUMULATOR 26
2.1.4.4 PROGRAM FLAG 27
2.1.4.5 PROGRAM COUNTER 28
2.1.4.6 Y, Z REGISTERS 31
2.1.4.7 R REGISTERS 32
2.2 ADDRESSING MODE 33
2.2.1 IMMEDIATE ADDRESSING MODE 33
2.2.2 DIRECTLY ADDRESSING MODE 33
2.2.3 INDIRECTLY ADDRESSING MODE 33
2.3 STACK OPERATION 34
2.3.1 OVERVIEW 34
2.3.2 STACK REGISTERS 35
2.3.3 STACK OPERATION EXAMPLE 36

3 RESET 37

3.1 OVERVIEW 37

3.2 POWER ON RESET 38
3.3 WATCHDOG RESET 38
3.4 BROWN OUT RESET 39
3.4.1 BROWN OUT DESCRIPTION 39
3.4.2 THE SYSTEM OPERATING VOLTAGE DECSRIPTION 40
3.4.3 BROWN OUT RESET IMPROVEMENT 40
3.5 EXTERNAL RESET 42
3.6 EXTERNAL RESET CIRCUIT 42
3.6.1 Simply RC Reset Circuit 42
3.6.2 Diode & RC Reset Circuit 43
3.6.3 Zener Diode Reset Circuit 43
3.6.4 Voltage Bias Reset Circuit 44
3.6.5 External Reset IC 45

4 SYSTEM CLOCK 46

4.1 OVERVIEW 46
4.2 CLOCK BLOCK DIAGRAM 46
4.3 OSCM REGISTER 47
4.4 SYSTEM HIGH CLOCK 48
4.4.1 EXTERNAL HIGH CLOCK 48
4.4.1.1 CRYSTAL/CERAMIC 49
4.4.1.2 RC 49
4.4.1.3 EXTERNAL CLOCK SIGNAL 50
4.5 SYSTEM LOW CLOCK 51
4.5.1 SYSTEM CLOCK MEASUREMENT 52

5 SYSTEM OPERATION MODE 53

5.1 OVERVIEW 53
5.2 SYSTEM MODE SWITCHING 54
5.3 WAKEUP 56
5.3.1 OVERVIEW 56
5.3.2 WAKEUP TIME 56
5.3.3 P1W WAKEUP CONTROL REGISTER 57

6 INTERRUPT 58

6.1 OVERVIEW 58
6.2 INTEN INTERRUPT ENABLE REGISTER 59
6.3 INTRQ INTERRUPT REQUEST REGISTER 60
6.4 GIE GLOBAL INTERRUPT OPERATION 60
6.5 PUSH, POP ROUTINE 61
6.6 INT0 (P0.0) INTERRUPT OPERATION 62

6.7 T0 INTERRUPT OPERATION 63
6.8 TC0 INTERRUPT OPERATION 64
6.9 MULTI-INTERRUPT OPERATION 65

7 I/O PORT 66

7.1 I/O PORT MODE 66
7.2 I/O PULL UP REGISTER 67
7.3 I/O OPEN-DRAIN REGISTER 68
7.4 I/O PORT DATA REGISTER 69

8 TIMERS 70

8.1 WATCHDOG TIMER 70
8.2 TIMER 0 (T0) 72
8.2.1 OVERVIEW 72
8.2.2 T0M MODE REGISTER 72
8.2.3 T0C COUNTING REGISTER 73
8.2.4 T0 TIMER OPERATION SEQUENCE 74
8.2.5 T0 TIMER NOTICE 75
8.3 TIMER/COUNTER 0 (TC0) 75
8.3.1 OVERVIEW 75
8.3.2 TC0M MODE REGISTER 77
8.3.3 TC0C COUNTING REGISTER 78
8.3.4 TC0R AUTO-LOAD REGISTER 79
8.3.5 TC0 CLOCK FREQUENCY OUTPUT (BUZZER) 80
8.3.6 TC0 TIMER OPERATION SEQUENCE 81
8.3.7 TC0 TIMER NOTICE 82
8.4 PWM MODE 83
8.4.1 OVERVIEW 83
8.4.2 TCnIRQ and PWM Duty 84
8.4.3 PWM Duty with TCnR Changing 85
8.4.4 PWM PROGRAM EXAMPLE 86

9 INSTRUCTION TABLE 87

10 ELECTRICAL CHARACTERISTIC 88

10.1 ABSOLUTE MAXIMUM RATING 88
10.2 ELECTRICAL CHARACTERISTIC 88
10.3 CHARACTERISTIC GRAPHS 89
10.3.1 SN8P2602A 89

11 APPLICATION NOTICE 90

11.1 DEVELOPMENT TOOL VERSION 90
11.1.1 ICE (In circuit emulation) 90
11.1.2 OTP Writer 90
11.1.3 SN8IDE 90
11.2 CODE OPTION 91
11.2.1 NOISE FILTER CODE OPTION 91
11.2.2 WATCHDOG 91
11.3 INTERRUPT VECTOR (ORG 8) 92
11.4 INSTRUCTION 93
11.4.1 B0MOV M,I 93
11.4.2 B0XCH A, M 93
11.5 S8KD-2 ICE ENULATION 94
11.5.1 ICE_MODE 94
11.5.2 INSTRUCTION CYCLE 95
11.5.3 SYSTEM CLOCK 97
11.5.4 WATCHDOG TIMER 98
11.5.5 P0 EMULATION 99
11.5.5.1 @P00_MODE 99
11.5.5.2 @P00_OUT 99
11.5.5.3 PEDGE 100
11.5.6 PWM DUTY 101
11.5.7 OTHER MACRO 102

12 OTP PROGRAMMING PIN 103

12.1.1 The pin assignment of Easy Writer transition board socket: 103
12.1.2 The pin assignment of Writer V3.0 and V2.5 transition board socket: 103
12.1.3 SN8P2602A Programming Pin Mapping: 104

13 PACKAGE INFORMATION 105

13.1 P-DIP 18 PIN 105
13.2 SOP 18 PIN 106
13.3 SSOP 20 PIN 107

1 PRODUCT OVERVIEW

1.1 FEATURES

- **Memory configuration**
  OTP ROM size: 1K * 16 bits.
  RAM size: 48 * 8 bits.
  Four levels stack buffer

- **I/O pin configuration**
  Bi-directional: P0, P1, P5.
  Input only: P1.5.
  Programmable open-drain: P1.0
  Wakeup: P0, P1 level change trigger
  Pull-up resisters: P0, P1, P5.
  External interrupt: P0.

- **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)

- **Three interrupt sources**
  Two internal interrupts: T0, TC0.
  One external interrupts: INT0.

- **Two 8-bit Timer/Counter**
  T0: Basic timer
  TC0: Auto-reload timer/Counter/PWM0/Buzzer output

- **On chip watchdog timer and clock source is internal low clock RC type (16KHz @3V, 32KHz @5V).**

- **Dual system clocks**
  External high clock: RC type up to 10 MHz
  External high clock: Crystal type up to 16 MHz
  Internal low clock: RC type 16KHz(3V), 32KHz(5V)

- **Operating modes**
  Normal mode: Both high and low clock active
  Slow mode: Low clock only
  Sleep mode: Both high and low clock stop
  Green mode: Periodical wakeup by T0 Timer

- **Package (Chip form support)**
  PDIP 18 pins
  SOP 18 pins
  SSOP20 pins

☞ **Features Selection Table**

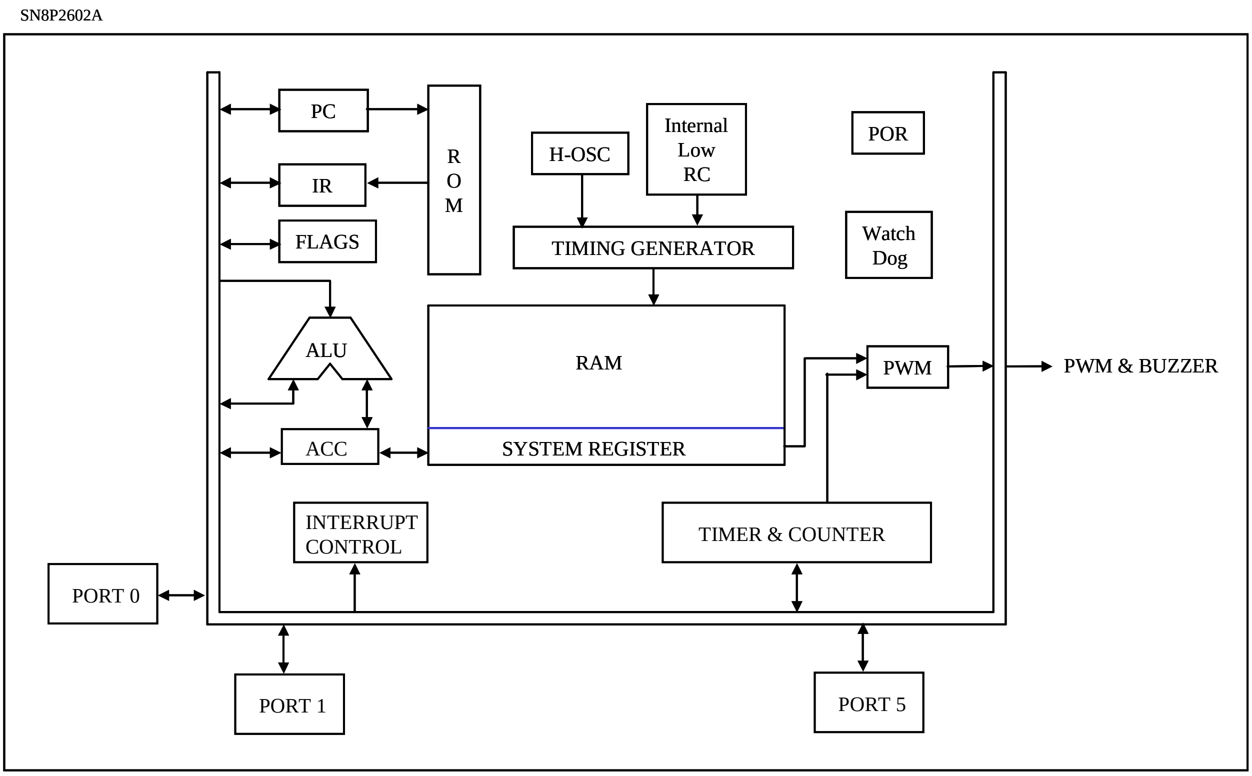
CHIP	ROM	RAM	Stack	Timer		I/O	Green Mode	PWM Buzzer	Wakeup Pin No.	Package
				T0	TC0					
SN8P1602B	1K*16	48	4	-	V	14	V	-	6	DIP18/SOP18/SSOP20
SN8P2602A	1K*16	48	4	V	V	15	V	V	7	DIP18/SOP18/SSOP20

Migration SN8P1602A/SN8P1602B to SN8P2602A

Item	SN8P2602A	SN8P1602A/SN8P1602B
DAA	Not available	Available
PUSH/POP (Save A and PFLAG)	Available	Not available
B0MOV M, I	I can't be 0E6h or 0E7h	-
B0XCH A, M	The address of M can't be 80h~FFh	-
Valid instruction in ROM address 8	JMP or NOP	No Limitation
AC Noise Immunity Capability	Excellent **(Add an 47uF bypass Capacitor between VDD and GND)**	Normal
Computation Power (16Mhz Crystal)	Up to 16 MIPS	Up to 4 MIPS
High Speed PWM	PWM Resolution: 8bit/6bit/5bit/4bit High Clock = 16MHz Up to 31.25K at 8bit resolution Up to 500K at 4bit resolution	PWM Resolution: 8bit only High Clock = 16MHz Up to 7.8125K at 8bit resolution
Timer	T0/TC0	TC0
Maximum I/O number	15 (P1.5)	14
Programmable Open-Drain Output	P1.0/P1.1	-
On-chip Pull-up Resistor	Yes	Yes
Green Mode	Yes	Yes
P0.0 Interrupt trigger Edge	Controlled by PEDGE register: P00G[1:0]: (bit4 and bit3) 00: Reserved 01: Rising edge 10: Falling edge 11: Level change	Controlled by PEDGE register: PEDGEN (bit7): 0: Falling edge 1: Defined by P00G[1:0] P00G[1:0]: (bit4 and bit3) 00: Reserved 01: Falling edge 10: Rising edge 11: Level change
P0.0 Wakeup	Level Change	Controlled by PEDGE register: PEDGEN (bit7): 0: Falling edge 1: Defined by P00G[1:0] P00G[1:0]: (bit4 and bit3) 00: Reserved 01: Falling edge 10: Rising edge 11: Level change
Port 1 Wakeup	Level Change	Controlled by PEDGE register: PEDGEN (bit7): 0: Low level wakeup 1: Level change
Wakeup Time	1/Fosc * 4096 (sec) + Oscillator settling time	1/Fosc * 2048 (sec) + Oscillator settling time
Schmitt Trigger input	All input pin	P0.0, P0.1, RST, XIN
Watchdog timer clock source	Internal Low RC Only	Internal Low RC External High Clock
Clear Watchdog	MOV A, #0x5A B0MOV WDTR, A	B0BSET FWDRST
Standby Current	1uA at 5V	SN8P1602A: 10uA at 5V SN8P1602B: 1uA at 5V
LVD	1.8V always ON	1.8V always ON

✸ ***Note: Level change trigger mean falling or rising edge trigger.***

1.2 SYSTEM BLOCK DIAGRAM



1.3 PIN ASSIGNMENT

SN8P2602AP (P-DIP 18 pins)
SN8P2602AS (SOP 18 pins)
SN8P2602AX (SSOP 20 pins)

P1.2	1	U	18	P1.1
P1.3	2		17	P1.0
P0.0/INT0	3		16	XIN
P1.5/RST/VPP	4		15	XOUT/P1.4
VSS	5		14	VDD
P5.0	6		13	P5.7
P5.1	7		12	P5.6
P5.2	8		11	P5.5
P5.3	9		10	P5.4/BZ0/PWM0

SN8P2602AP
SN8P2602AS

P1.2	1	U	20	P1.1
P1.3	2		19	P1.0
P0.0/INT0	3		18	XIN
P1.5/RST/VPP	4		17	XOUT/P1.4
VSS	5		16	VDD
VSS	6		15	VDD
P5.0	7		14	P5.7
P5.1	8		13	P5.6
P5.2	9		12	P5.5
P5.3	10		11	P5.4/BZ0/PWM0

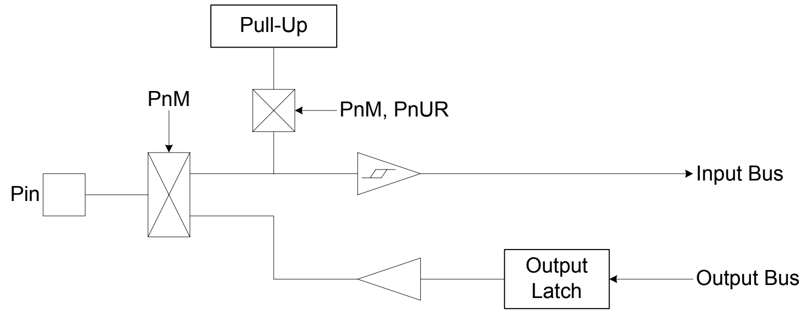
SN8P2602AX

1.4 PIN DESCRIPTIONS

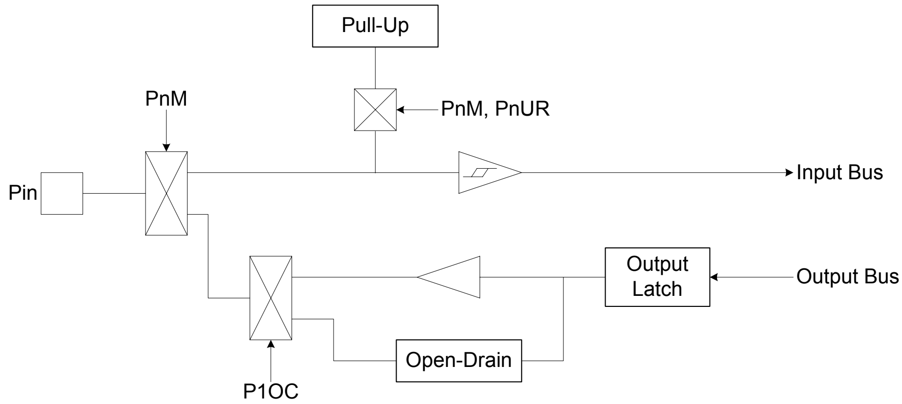
PAD NAME	TYPE	DESCRIPTION
VDD, VSS	P	Power supply input pins. Place the 0.1μF bypass capacitor between the VDD and VSS pin.
P1.5/RST/VPP	I, P	P1.5: Input only pin (Schmitt trigger) if disable external reset function. **P1.5 without build-in pull-up resister** RST: System reset input pin. Schmitt trigger structure, low active, normal stay to “high”. VPP: OTP programming pin.
XIN	I	XIN: Oscillator input pin if high clock select external oscillator (crystal or RC type).
XOUT/P1.4	I/O	P1.4: I/O pin (Schmitt trigger) if high clock select internal high RC oscillator / Built-in pull-up resister. XOUT: Oscillator output pin if high clock select external crystal type oscillator.
P0.0/INT0	I/O	P0.0: I/O pin (Schmitt trigger) and shared with INT0 trigger pin (Schmitt trigger) / Built-in pull-up resister. TC0 event counter clock input pin.
P1.0~P1.3	I/O	Port 1.0~P1.3 bi-direction pin (Schmitt trigger) / Built-in pull-up resister / Open-Drain pin.
P5.4/BZ0/PWM0	I/O	Port 5.4 bi-direction pin (Schmitt trigger) / Built-in pull-up resister. TC0 ÷ 2 signal output pin for buzzer and PWM output pin.
P5.0~P5.7	I/O	Port 5.0~Port 5.7 bi-direction pins (Schmitt trigger) / Built-in pull-up resisters.

1.5 PIN CIRCUIT DIAGRAMS

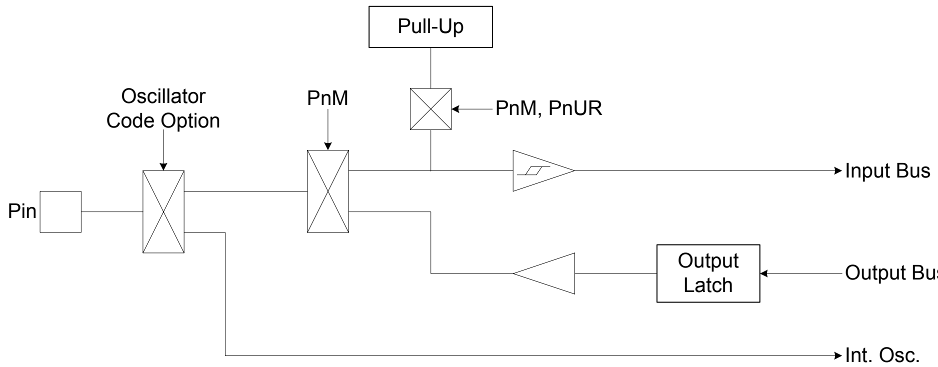
Port 0, 1, 5 structure:



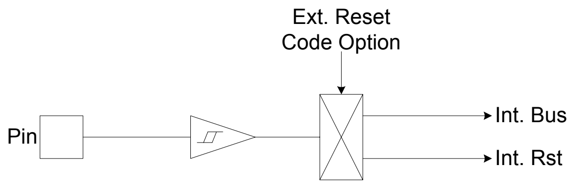
Port 1.0 structure:



Port 1.4 structure:



Port 1.5 structure:



2 CENTRAL PROCESSOR UNIT (CPU)

2.1 MEMORY MAP

2.1.1 PROGRAM MEMORY (ROM)

☞ ***1K words ROM***

Address	ROM	
0000H	***Reset vector***	User reset vector
0001H	***General purpose area***	Jump to user start address
0002H		Jump to user start address
0003H		Jump to user start address
0004H	***General purpose area***	
0005H		
0006H		
0007H		
0008H	***Interrupt vector***	User interrupt vector
0009H	***General purpose area***	User program
.		
.		
000FH		
0010H		
0011H		
.		
.		
.		
.		
.		
03FBH		End of user program
03FCH	***Reserved***	
03FDH		
03FEH		
03FFH		

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 and the first instruction at ORG 8 must be **"JMP"** or **"NOP"**. 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.***

> ✸ ***Note: The first instruction at ORG 8 must be "JMP" or "NOP".***

➢ **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.
            NOP                 ; The first instruction at ORG 8.
            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.**

```
.DATA           ACCBUF     DS   1         ; Define ACCBUF for store ACC data.
                PFLAGBUF   DS   1         ; Define PFLAGBUF for store PFLAG data.
.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 and the first instruction must be "NOP" or "JMP".***
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.

> ✸ ***Note: "B0MOV M, I" instruction doesn't support "I=0xE6" and "I=0xE7". In look-up table application, users have to check Y,Z value not be "0xE6" and "0xE7". To set ROM address of table start and avoid Y,Z to be "0xE6" and "0xE7".***

➢ **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.
            …                       ;

            ORG     0x0100          ; Set TABLE1 start address is 0x0100 to avoid "B0MOV M, I"
                                    instruction doesn't support "I=0xE6" and "I=0xE7".
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.
                …                        ;

                ORG       0x0100         ; Set TABLE1 start address is 0x0100 to avoid "B0MOV M, I"
                                         instruction doesn't support "I=0xE6" and "I=0xE7".
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
                …

                ORG       0x0100         ; Set TABLE1 start address is 0x0100 to avoid "B0MOV M, I"
                                         instruction doesn't support "I=0xE6" and "I=0xE7".
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 address 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	RC	Low cost RC for external high clock oscillator and XOUT becomes to general purpose I/O (P1.4).
	32K X'tal	Low frequency, power saving crystal (e.g. 32.768KHz) for external high clock oscillator.
	12M X'tal	High speed crystal /resonator (e.g. 12MHz) for external high clock oscillator.
	4M X'tal	Standard crystal /resonator (e.g. 4M) for external high clock oscillator.
Watch_Dog	Always_On	Watchdog timer always on 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	Fosc/1	Instruction cycle is oscillator clock. Notice: In Fosc/1, Noise Filter must be disabled.
	Fosc/2	Instruction cycle is 2 oscillator clocks. Notice: In Fosc/2, Noise Filter must be disabled.
	Fosc/4	Instruction cycle is 4 oscillator clocks.
	Fosc/8	Instruction cycle is 8 oscillator clocks.
	Fosc/16	Instruction cycle is 16 oscillator clocks.
	Fosc/32	Instruction cycle is 32 oscillator clocks.
	Fosc/64	Instruction cycle is 64 oscillator clocks.
	Fosc/128	Instruction cycle is 128 oscillator clocks.
Reset_Pin	Reset	Enable External reset pin.
	P15	Enable P1.5 input only without pull-up resister.
Security	Enable	Enable ROM code Security function.
	Disable	Disable ROM code Security function.
Noise_Filter	Enable	Enable Noise Filter and the Fcpu is Fosc/4~Fosc/128.
	Disable	Disable Noise Filter and the Fcpu is Fosc/1~Fosc/128.

✸ ***Note:***

1. ***In high noisy environment, enable "Noise Filter" and set Watch_Dog as "Always_On" is strongly recommended. Enable "Noise_Filter" will limit the Fcpu = Fosc/4 ~ Fosc/128.***
2. ***If users define watchdog as "Always_On", assembler will Enable "Watch_Dog" automatically.***
3. ***Fcpu code option is only available for High Clock. Fcpu of slow mode is Fosc/4 (the Fosc is internal low clock).***

2.1.3 DATA MEMORY (RAM)

☞ ***48 X 8-bit RAM***

	Address	RAM location	
BANK 0	000h " " " " " 02Fh	*General purpose area*	
	080h " " " " "	*System register*	80h~FFh of Bank 0 store system registers (128 bytes).
	0FFh	*End of bank 0 area*	

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	-	-	-	-	-	-	-	-	-
9	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
A	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
B	-	-	-	-	-	-	-	-	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	-	-	-	-	-	-	-	-	STK3L	STK3H	STK2L	STK2H	STK1L	STK1H	STK0L	STK0H

Description

PFLAG = ROM page and special flag register.
P1W = Port 1 wakeup register.
PEDGE = P0.0 edge direction register.
PnM = Port n input/output mode register.
P1OC = Port 1 open-drain control register.
INTRQ = Interrupt request register.
OSCM = Oscillator mode register.
T0M = T0 mode register.
TC0M = TC0 mode register.
TC0R = TC0 auto-reload data buffer.
STKP = Stack pointer buffer.

R = Working register and ROM look-up data buffer.
Y, Z = Working, @YZ and ROM addressing register.
@YZ = RAM YZ indirect addressing index pointer.
Pn = Port n data buffer.
PnUR = Port n pull-up resister control register.
INTEN = Interrupt enable register.
PCH, PCL = Program counter.
T0C = TC0 counting register.
TC0C = TC0 counting register.
WDTR = Watchdog timer clear register.
STK0~STK3 = Stack 0 ~ stack 3 buffer.

2.1.4.2 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
0B8H	-	-	-	-	-	-	-	P00M	R/W	P0M
0BFH	-	-	-	P00G1	P00G0	-	-	-	R/W	PEDGE
0C0H	-	-	P15W	P14W	P13W	P12W	P11W	P10W	W	P1W wakeup register
0C1H	-	-	-	P14M	P13M	P12M	P11M	P10M	R/W	P1M I/O direction
0C5H	P57M	P56M	P55M	P54M	P53M	P52M	P51M	P50M	R/W	P5M I/O direction
0C8H	-	-	TC0IRQ	T0IRQ	-	-	-	P00IRQ	R/W	INTRQ
0C9H	-	-	TC0IEN	T0IEN	-	-	-	P00IEN	R/W	INTEN
0CAH	0	0	0	CPUM1	CPUM0	CLKMD	STPHX	0	R/W	OSCM
0CCH	WDTR7	WDTR6	WDTR5	WDTR4	WDTR3	WDTR2	WDTR1	WDTR0	W	WDTR
0CDH	TC0R7	TC0R6	TC0R5	TC0R4	TC0R3	TC0R2	TC0R1	TC0R0	W	TC0R
0CEH	PC7	PC6	PC5	PC4	PC3	PC2	PC1	PC0	R/W	PCL
0CFH	-	-	-	-	-	-	PC9	PC8	R/W	PCH
0D0H	-	-	-	-	-	-	-	P00	R/W	P0 data buffer
0D1H	-	-	P15	P14	P13	P12	P11	P10	R/W	P1 data buffer
0D5H	P57	P56	P55	P54	P53	P52	P51	P50	R/W	P5 data buffer
0D8H	T0ENB	T0rate2	T0rate1	T0rate0	-	-	-	-	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 stack pointer
0E0H	-	-	-	-	-	-	-	P00R	W	P0 pull-up register
0E1H	-	-	-	P14R	P13R	P12R	P11R	P10R	W	P1 pull-up register
0E5H	P57R	P56R	P55R	P54R	P53R	P52R	P51R	P50R	W	P5 pull-up register
0E7H	@YZ7	@YZ6	@YZ5	@YZ4	@YZ3	@YZ2	@YZ1	@YZ0	R/W	@YZ index pointer
0E9H	-	-	-	-	-	-	-	P10OC	W	P1OCopen-drain
0F8H	S3PC7	S3PC6	S3PC5	S3PC4	S3PC3	S3PC2	S3PC1	S3PC0	R/W	STK3L
0F9H	-	-	-	-	-	-	S3PC9	S3PC8	R/W	STK3H
0FAH	S2PC7	S2PC6	S2PC5	S2PC4	S2PC3	S2PC2	S2PC1	S2PC0	R/W	STK2L
0FBH	-	-	-	-	-	-	S2PC9	S2PC8	R/W	STK2H
0FCH	S1PC7	S1PC6	S1PC5	S1PC4	S1PC3	S1PC2	S1PC1	S1PC0	R/W	STK1L
0FDH	-	-	-	-	-	-	S1PC9	S1PC8	R/W	STK1H
0FEH	S0PC7	S0PC6	S0PC5	S0PC4	S0PC3	S0PC2	S0PC1	S0PC0	R/W	STK0L
0FFH	-	-	-	-	-	-	S0PC9	S0PC8	R/W	STK0H

✷ ***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".***

2.1.4.3 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
```

2.1.4.4 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.***

2.1.4.5 PROGRAM COUNTER

The program counter (PC) is a 10-bit binary counter separated into the high-byte 2 and the low-byte 8 bits. This counter is responsible for pointing a location in order to fetch an instruction for kernel circuit. Normally, the program counter is automatically incremented with each instruction during program execution.
Besides, it can be replaced with specific address by executing CALL or JMP instruction. When JMP or CALL instruction is executed, the destination address will be inserted to bit 0 ~ bit 9.

	Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
PC	-	-	-	-	-	-	PC9	PC8	PC7	PC6	PC5	PC4	PC3	PC2	PC1	PC0
After reset	-	-	-	-	-	-	0	0	0	0	0	0	0	0	0	0
	PCH								PCL							

☞ **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
            …
            …
```

2.1.4.6 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
            …
```

2.1.4.7 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).

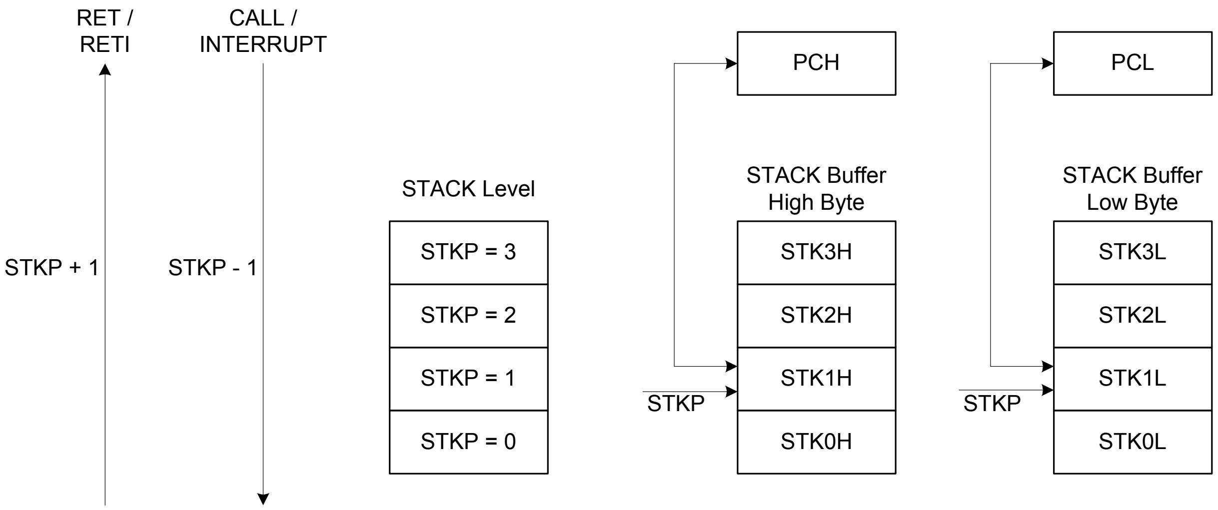
- **Example: Indirectly addressing mode with @YZ register.**

```
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 4-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, 10-bit data memory (STKnH and STKnL) set aside for temporary storage of stack addresses.

The two stack operations are writing to the top of the stack (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	-	-	-	-	-	-	SnPC9	SnPC8
Read/Write	-	-	-	-	-	-	R/W	R/W
After reset	-	-	-	-	-	-	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 = 3 ~ 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	-
> 4	0	1	0	-	-	**Stack Over, error**

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

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

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)

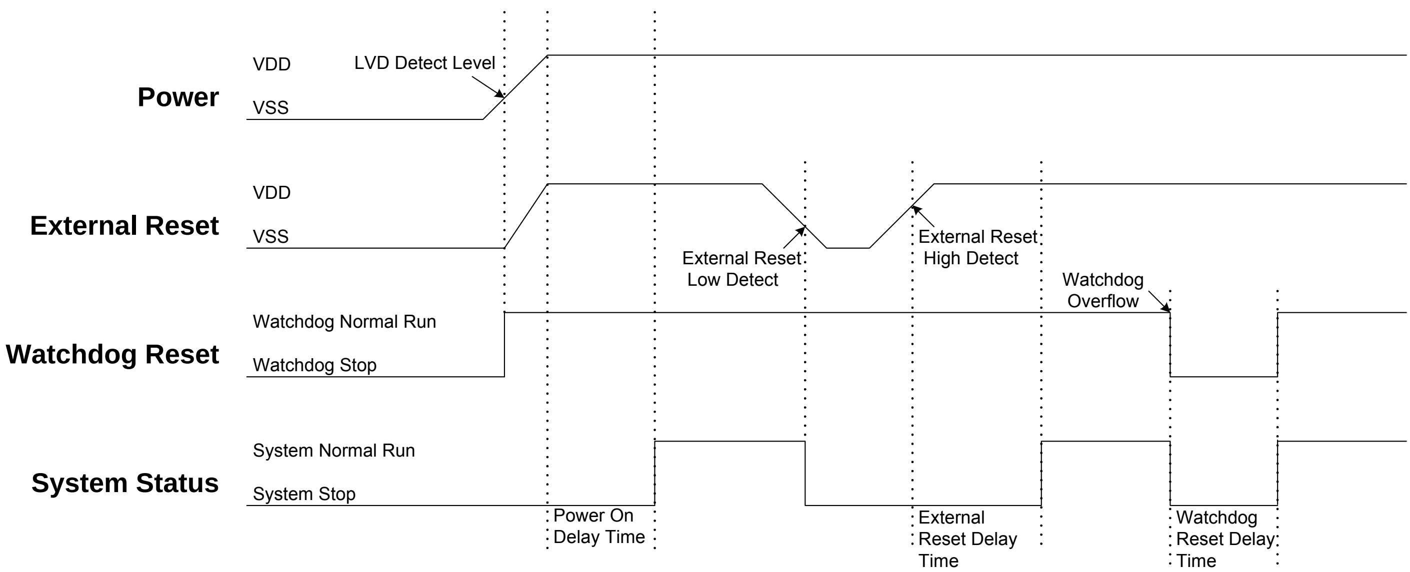
When any reset condition occurs, all system registers keep initial status, program stops and program counter is cleared. After reset status released, the system boots up and program starts to execute from ORG 0. The NT0, NPD flags indicate system reset status. The system can depend on NT0, NPD status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

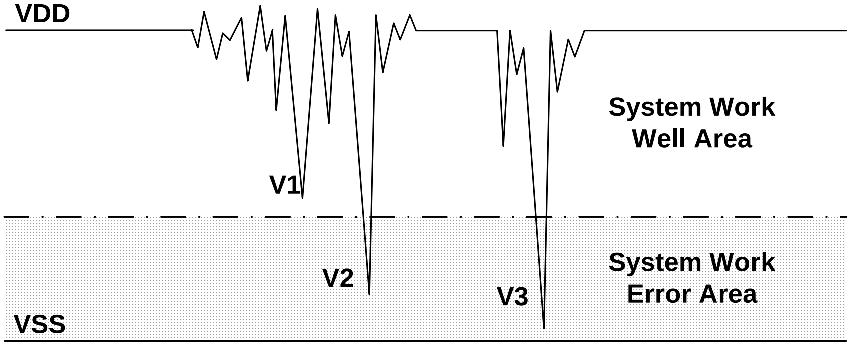
- Before clearing watchdog timer, check I/O status and check RAM contents can improve system error.
- Don't clear watchdog timer in interrupt vector and interrupt service routine. That can improve main routine fail.
- Clearing watchdog timer program is only at one part of the program. This way is the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

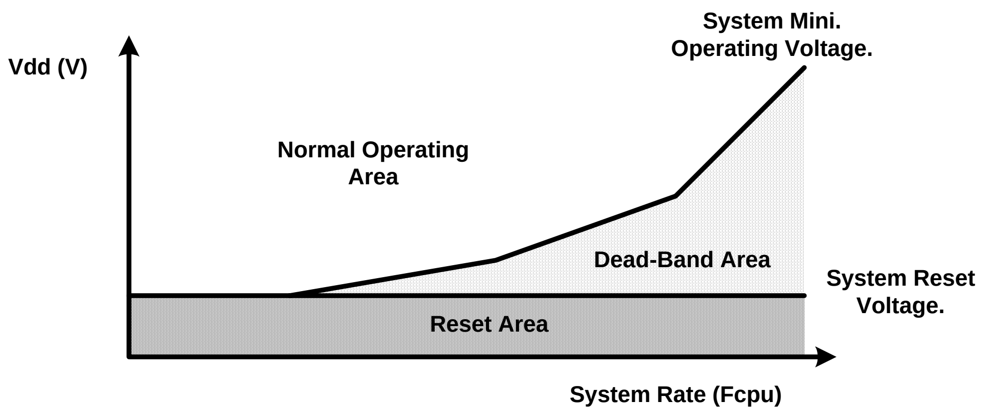
AC application:

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.

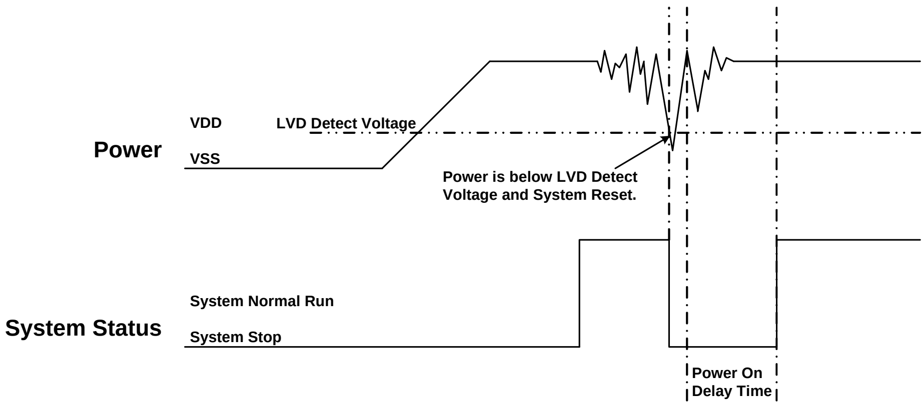
3.4.3 BROWN OUT RESET IMPROVEMENT

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

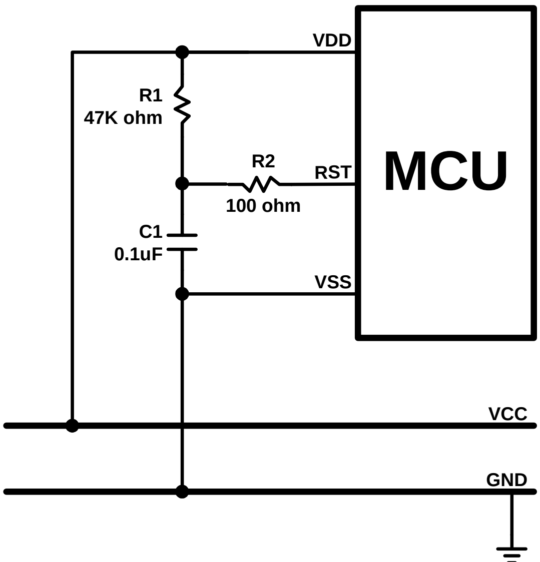
External reset function is controlled by "Reset_Pin" code option. Set the code option as "Reset" option to enable external reset function. External reset pin is Schmitt Trigger structure and low level active. The system is running when reset pin is high level voltage input. The reset pin receives the 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

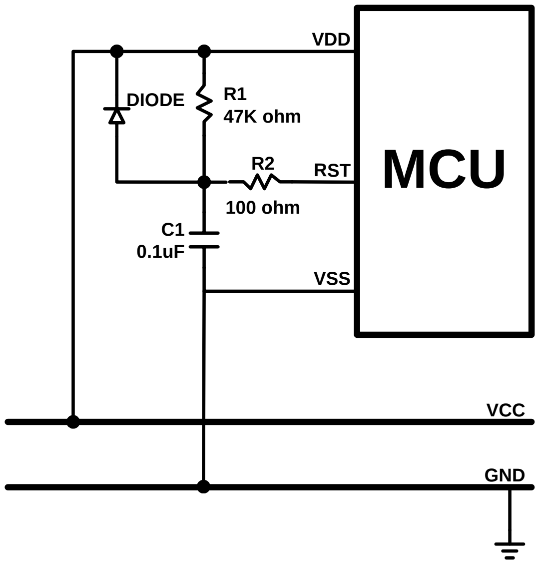
3.6.1 Simply RC Reset Circuit



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.***

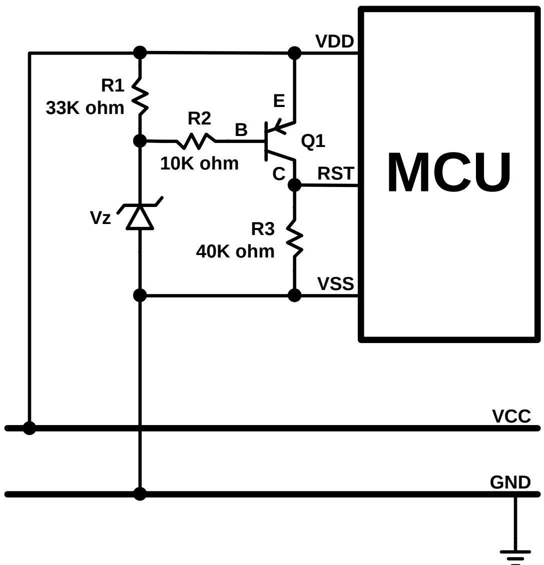
3.6.2 Diode & RC Reset Circuit



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.

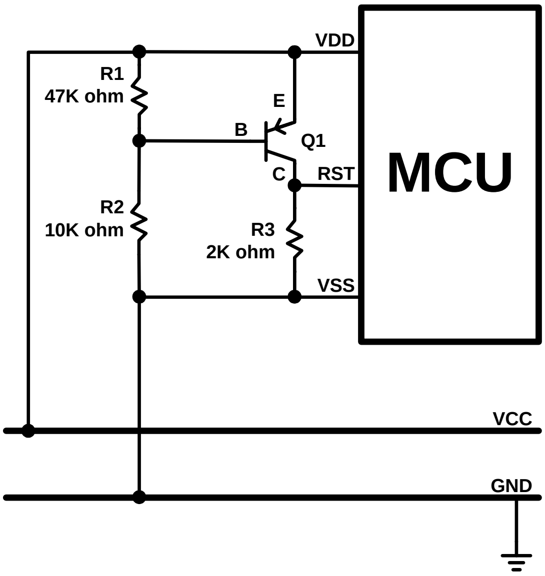
> ✸ ***Note: The R2 100 ohm resistor of "Simply reset circuit" and "Diode & RC reset circuit" is necessary to limit any current flowing into reset pin from external capacitor C in the event of reset pin breakdown due to Electrostatic Discharge (ESD) or Electrical Over-stress (EOS).***

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.

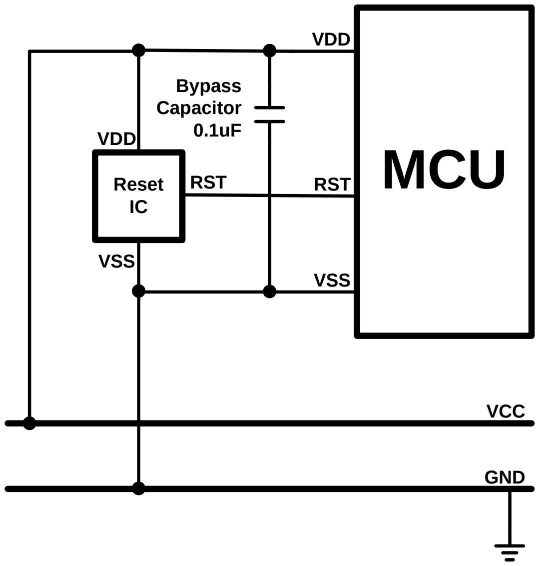
3.6.4 Voltage Bias Reset Circuit



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).

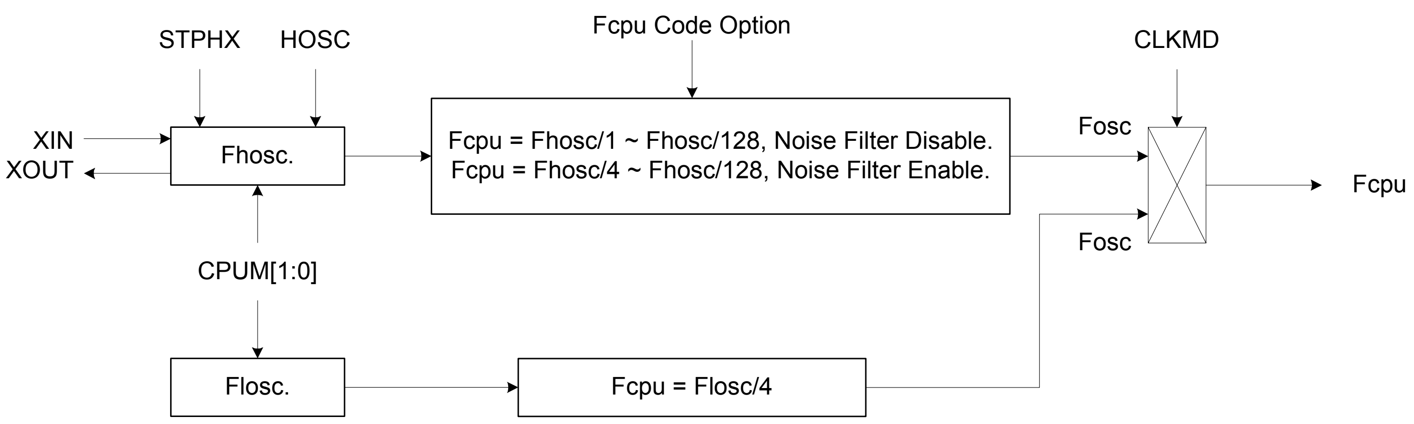
Both the high-speed clock and the low-speed clock can be system clock (Fosc). The system clock in slow mode is divided by 4 to be the instruction cycle (Fcpu).

☞ **Normal Mode (High Clock):** **Fcpu = Fhosc / N**, N = 1 ~ 128, 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

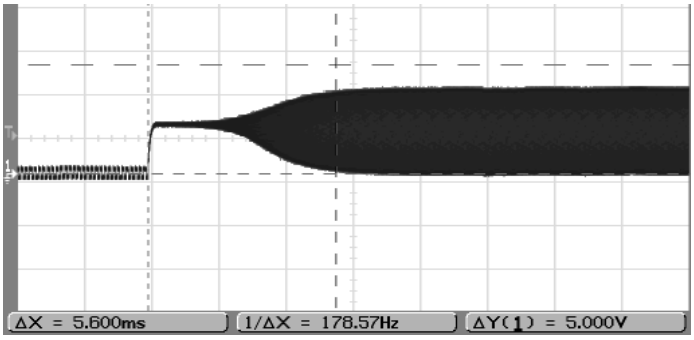
The system high clock is from external oscillator. The high clock type is controlled by “High_Clk” code option.

High_Clk Code Option	Description
RC	The high clock is external RC type oscillator. XOUT pin is general purpose I/O pin.
32K	The high clock is external 32768Hz low speed oscillator.
12M	The high clock is external high speed oscillator. The typical frequency is 12MHz.
4M	The high clock is external oscillator. The typical frequency is 4MHz.

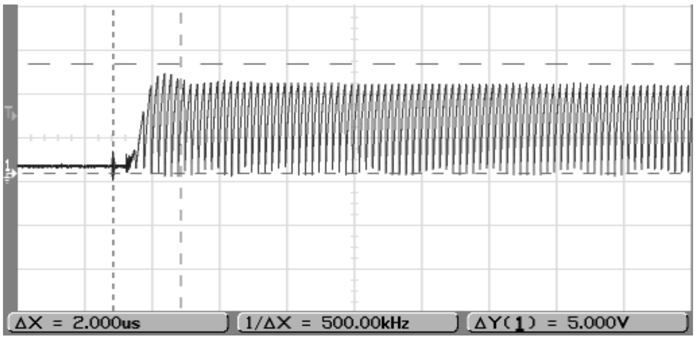
4.4.1 EXTERNAL HIGH CLOCK

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.

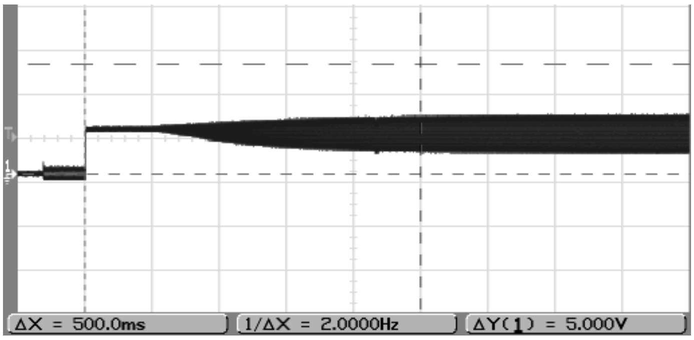
4MHz Crystal



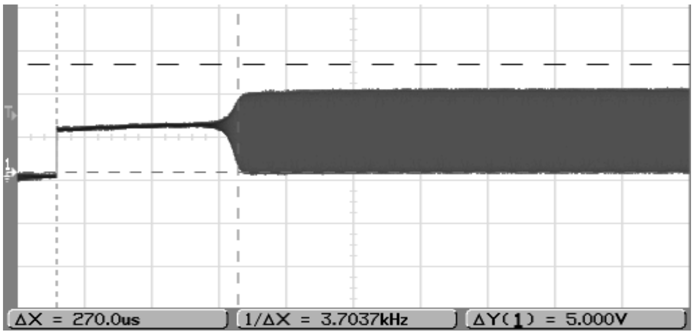
RC



32768Hz Crystal

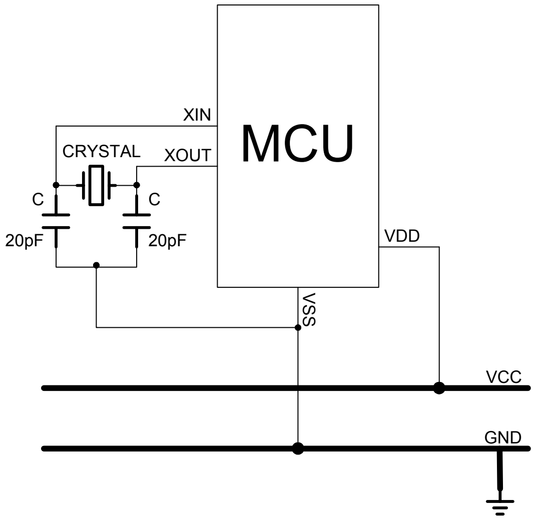


4MHz Ceramic



4.4.1.1 CRYSTAL/CERAMIC

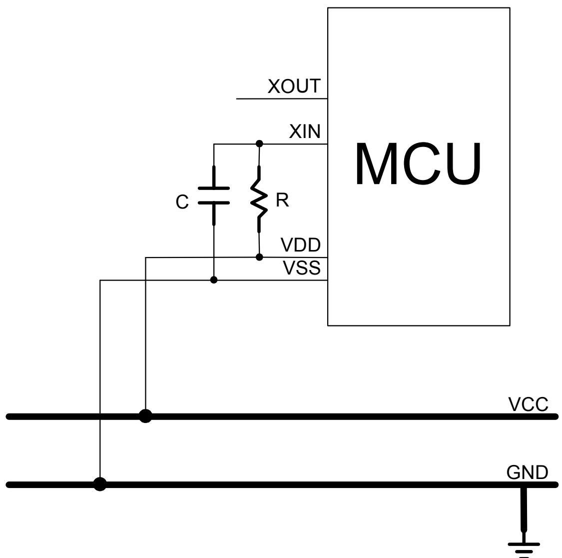
Crystal/Ceramic devices are driven by XIN, XOUT pins. For high/normal/low frequency, the driving currents are different. High_Clk code option supports different frequencies. 12M option is for high speed (ex. 12MHz). 4M option is for normal speed (ex. 4MHz). 32K option is for low speed (ex. 32768Hz).



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

4.4.1.2 RC

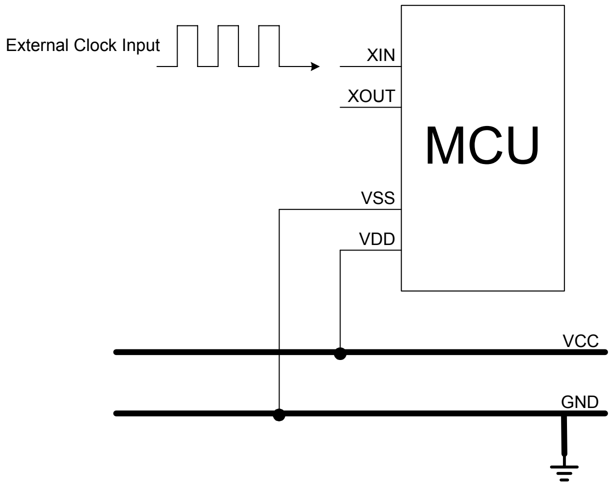
Selecting RC oscillator is by RC option of High_Clk code option. RC type oscillator's frequency is up to 10MHz. Using "R" value is to change frequency. 50P~100P is good value for "C". XOUT pin is general purpose I/O pin.



✸ ***Note: Connect the R and C as near as possible to the VDD pin of micro-controller.***

4.4.1.3 EXTERNAL CLOCK SIGNAL

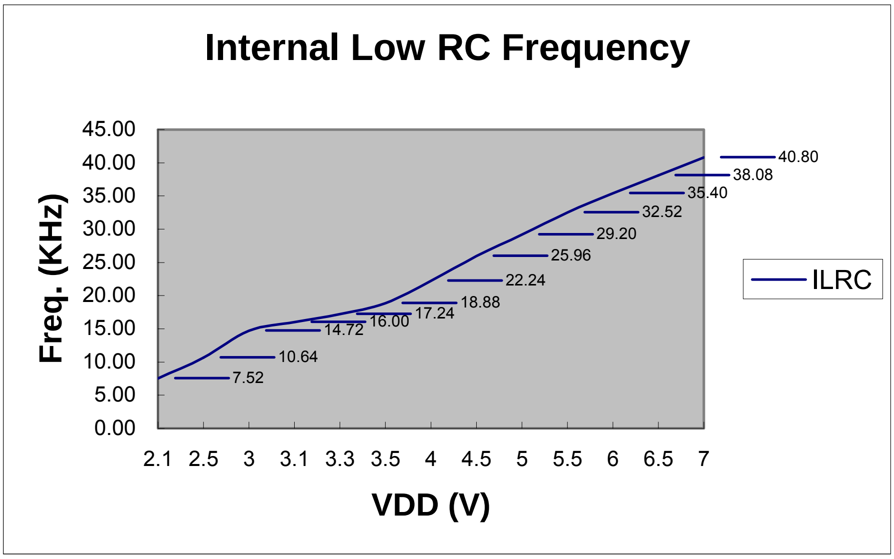
Selecting external clock signal input to be system clock is by RC option of High_Clk code option. The external clock signal is input from XIN pin. XOUT pin is general purpose I/O pin.



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

4.5 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.***

4.5.1 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
```

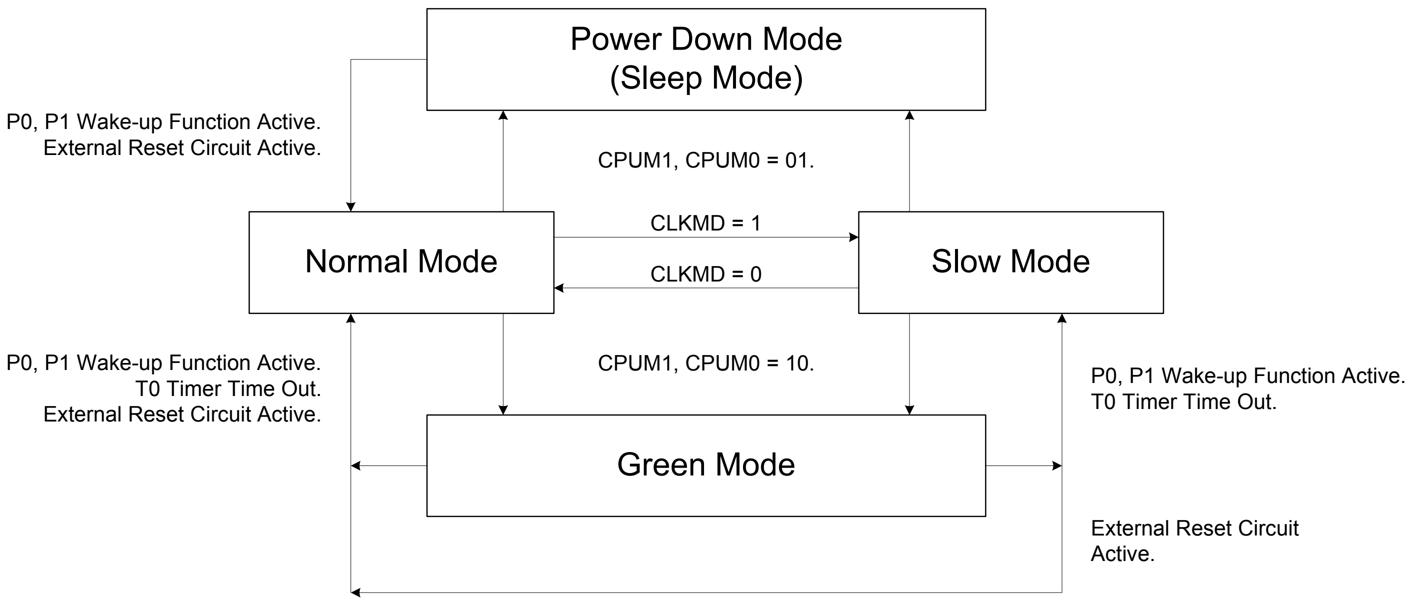
✸ ***Note: Do not measure the RC frequency directly from XIN; the probe impendence will affect the RC frequency.***

5 SYSTEM OPERATION MODE

5.1 OVERVIEW

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

- Normal mode (High-speed mode)
- Slow 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	*Active	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, TC0	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: 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 20ms for external clock stable.

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

        B0MOV       Z, #54          ; If VDD = 5V, internal RC=32KHz (typical) will delay
@@:     DECMS       Z               ; 0.125ms X 162 = 20.25ms 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 wakeup 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 pins, reset pin and T0 can wakeup the system back to the last mode. T0 wake-up period is controlled by program and T0ENB must be set.***

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) and internal trigger (T0 timer overflow).

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

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 2048 external high-speed oscillator clocks as the wakeup time to stable the oscillator circuit. After the wakeup time, the system goes into the normal mode.

> ✷ ***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.

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 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 = 0.512 ms (Fosc = 4MHz)
The total wakeup time = 0.512 ms + oscillator start-up time

5.3.3 P1W WAKEUP CONTROL REGISTER

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

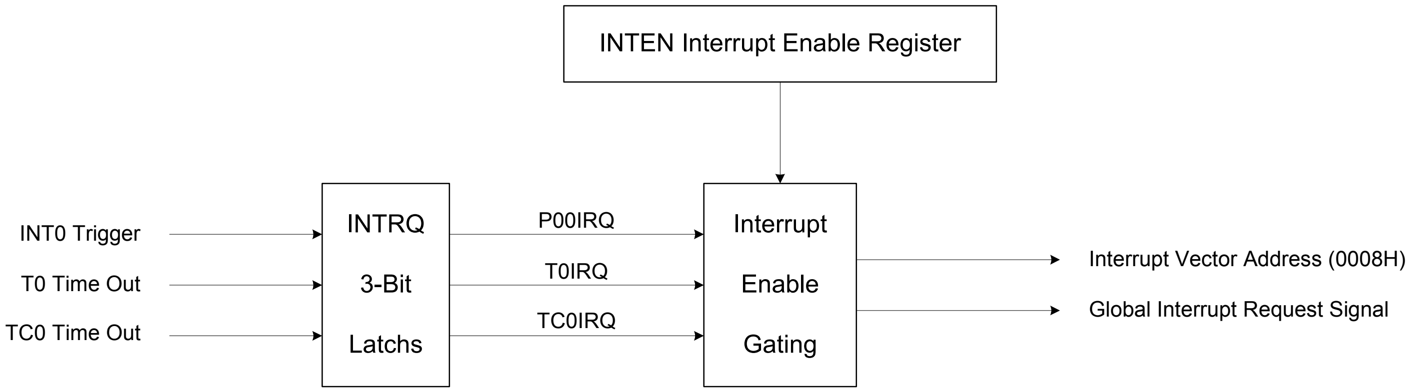
0C0H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
P1W	-	-	P15W	P14W	P13W	P12W	P11W	P10W
Read/Write	-	-	W	W	W	W	W	W
After reset	-	-	0	0	0	0	0	0

Bit[5:0] **P10W~P15W:** Port 1 wakeup function control bits.
0 = Disable P1n wakeup function.
1 = Enable P1n wakeup function.

6 INTERRUPT

6.1 OVERVIEW

This MCU provides three interrupt sources, including two internal interrupt (T0/TC0) 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	-	-	TC0IEN	T0IEN	-	-	-	P00IEN
Read/Write	-	-	R/W	R/W	-	-	-	R/W
After reset	-	-	0	0	-	-	-	0

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

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	-	-	TC0IRQ	T0IRQ	-	-	-	P00IRQ
Read/Write	-	-	R/W	R/W	-	-	-	R/W
After reset	-	-	0	0	-	-	-	0

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

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 instruction 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.

> ✸ ***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 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

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          ; Jump to INT0 interrupt service routine
INTT0CHK:                                   ; Check T0 interrupt request
                B0BTS1      FT0IEN          ; Check T0IEN
                JMP         INTTC0CHK       ; Jump check to next interrupt
                B0BTS0      FT0IRQ          ; Check T0IRQ
                JMP         INTT0           ; Jump to T0 interrupt service routine
INTTC0CHK:                                  ; Check TC0 interrupt request
                B0BTS1      FTC0IEN         ; Check TC0IEN
                JMP         INT_EXIT        ; Jump to exit of IRQ
                B0BTS0      FTC0IRQ         ; Check TC0IRQ
                JMP         INTTC0          ; Jump to TC0 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	-	-	-	-	-	-	-	P00M
Read/Write	-	-	-	-	-	-	-	R/W
After reset	-	-	-	-	-	-	-	0

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

0C5H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
P5M	P57M	P56M	P55M	P54M	P53M	P52M	P51M	P50M
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] **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. ***P1.5 is input only pin, and the P1M.5 keeps "1".***

➢ **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	-	-	-	-	-	-	-	P00R
Read/Write	-	-	-	-	-	-	-	W
After reset	-	-	-	-	-	-	-	0

0E1H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
P1UR	-	-	-	P14R	P13R	P12R	P11R	P10R
Read/Write	-	-	-	W	W	W	W	W
After reset	-	-	-	0	0	0	0	0

0E5H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
P5UR	P57R	P56R	P55R	P54R	P53R	P52R	P51R	P50R
Read/Write	W	W	W	W	W	W	W	W
After reset	0	0	0	0	0	0	0	0

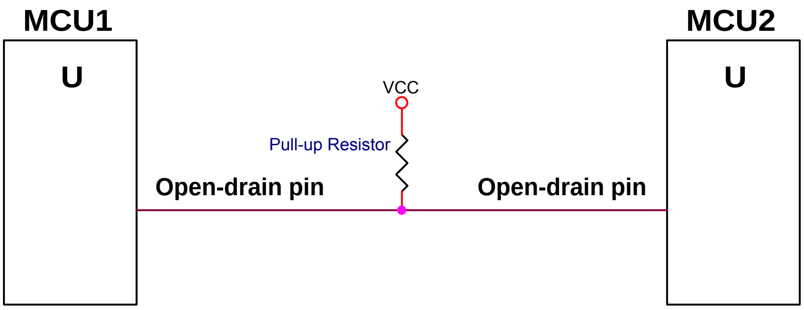
✸ ***Note: P1.5 is input only pin and without pull-up resister. The P1UR.5 keeps "1".***

➢ **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 is built-in open-drain function. P1.0 must be set as output mode when enable P1.0 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.

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

Bit 0 **P10OC:** P1.0 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 write only register. Setting P10OC must be used "MOV" instructions.***

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

```
MOV       A, #0       ; 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	-	-	-	-	-	-	-	P00
Read/Write	-	-	-	-	-	-	-	R/W
After reset	-	-	-	-	-	-	-	0

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

0D5H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
P5	P57	P56	P55	P54	P53	P52	P51	P50
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
After reset	0	0	0	0	0	0	0	0

✸ ***Note: The P15 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
```

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:***
1. ***If watchdog is "Always_On" mode, it keeps running event under power down mode or green mode.***
2. ***For S8KD ICE simulation, clear watchdog timer using "@RST_WDT" macro is necessary. Or the S8KD watchdog would be error.***

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
```

➢ **Example: Clear watchdog timer by @RST_WDT macro.**

```
Main:
            @RST_WDT                        ; Clear the watchdog timer.
            …
            …
            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
```

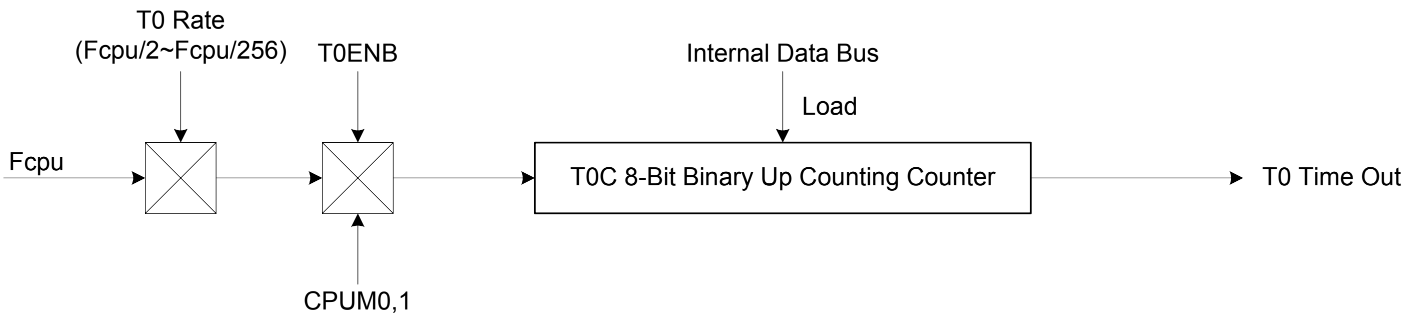
8.2 TIMER 0 (T0)

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 purposes 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 10ms interval time for T0 interrupt. High clock is external 4MHz. Fcpu=Fosc/4. Select T0RATE=010 (Fcpu/64).**

*T0C initial value = 256 - (T0 interrupt interval time * input clock)*
*= 256 - (10ms * 4MHz / 4 / 64)*
$= 256 - (10^{-2} * 4 * 10^{6} / 4 / 64)$
= 100
= 64H

The basic timer table interval time of T0.

T0RATE	T0CLOCK	High speed mode (Fcpu = 4MHz / 4)		Low speed mode (Fcpu = 32768Hz / 4)	
		Max overflow interval	One step = max/256	Max overflow interval	One step = max/256
000	Fcpu/256	65.536 ms	256 us	8000 ms	31250 us
001	Fcpu/128	32.768 ms	128 us	4000 ms	15625 us
010	Fcpu/64	16.384 ms	64 us	2000 ms	7812.5 us
011	Fcpu/32	8.192 ms	32 us	1000 ms	3906.25 us
100	Fcpu/16	4.096 ms	16 us	500 ms	1953.125 us
101	Fcpu/8	2.048 ms	8 us	250 ms	976.563 us
110	Fcpu/4	1.024 ms	4 us	125 ms	488.281 us
111	Fcpu/2	0.512 ms	2 us	62.5 ms	244.141 us

✹ ***Note: T0C doesn’t support read and modify write instructions as “B0ADD M,A . INCMS…”.***

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.
```

8.2.5 T0 TIMER NOTICE

When T0C.7 is from "1" to "0", T0IRQ is set "1" whether T0 is operating or not. If T0IRQ = 0 and T0C is changed by program, T0IRQ might be set as T0C.7 is from "1" to "0". The condition makes unexpected T0 interrupt occurring.

> **Example: T0C = 0x80 (T0C.7 = 1) and T0IRQ = 0. T0IRQ will set as "1" when T0C is cleared by program (T0C.7 = 0).**

```
MOV       A, #0       ; Clear T0C and T0C.7 is from "1" to "0".
B0MOV     T0C, A      ; T0IRQ changed from "0" to "1".

B0BSET    FT0IEN      ; Enable T0 interrupt function and system jumps to interrupt
                      ; vector (ORG 8) at next cycle.
```

If T0C changing in system operating duration is necessary, to disable T0 interrupt function (T0IEN = 0) before changing T0C value. The solution can avoid unexpected T0 interrupt occurring and example is as following.

> **Example: T0C = 0x80 and T0IRQ = 0. T0IRQ will change to "1" when T0C is cleared by program.**

```
B0BCLR    FT0IEN      ; Disable T0 interrupt function.

MOV       A, #0       ; Clear T0C and T0C.7 is from "1" to "0".
B0MOV     T0C, A      ; T0IRQ changed from "0" to "1".

B0BCLR    FT0IRQ      ; Clear T0IRQ flag.

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

✸ ***Note: Disable T0 interrupt function first, and load new T0C value into T0C buffer. This way can avoid unexpected T0 interrupt occurring.***

✸ ***Note: T0C doesn't support read and modify write instructions as "B0ADD M,A . INCMS…".***

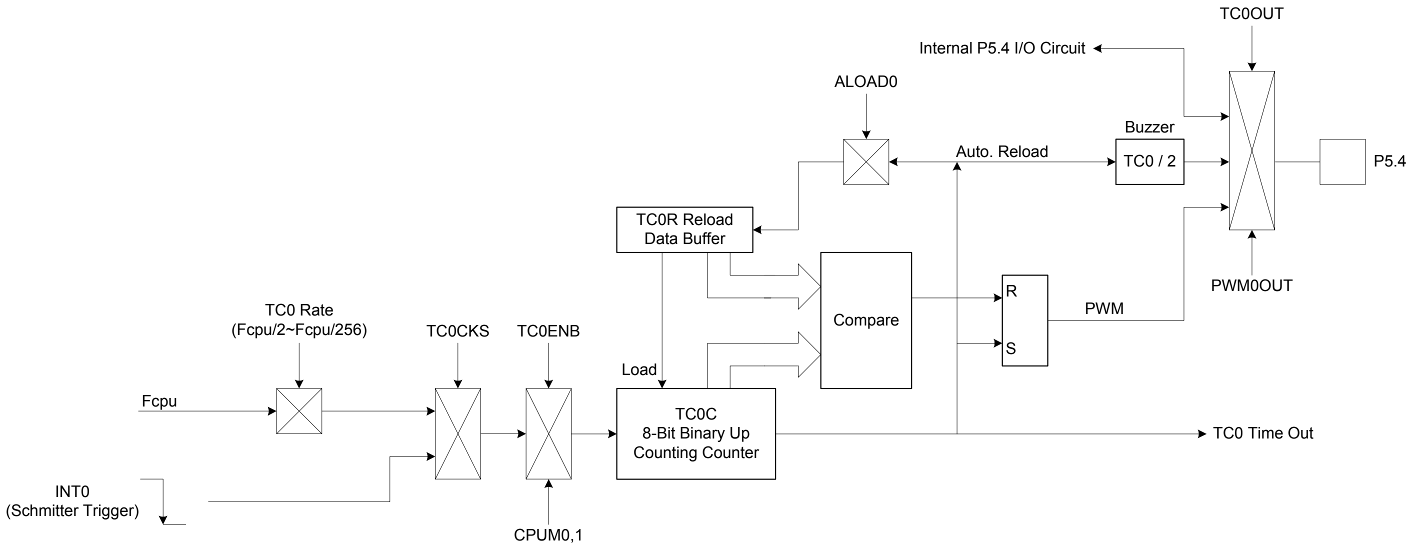
8.3 TIMER/COUNTER 0 (TC0)

8.3.1 OVERVIEW

The TC0 is an 8-bit binary up counting timer. TC0 has two clock sources including internal clock and external clock for counting a precision time. The internal clock source is from Fcpu. The external clock is INT0 from P0.0 pin (Falling edge trigger). Using TC0M register selects TC0C's clock source from internal or external. If TC0 timer occurs an overflow, it will continue counting and issue a time-out signal to trigger TC0 interrupt to request interrupt service. TC0 overflow time is 0xFF to 0X00 normally. Under PWM mode, TC0 overflow is still 256 counts.

The main purposes 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.
- **External event counter:** Counts system “events” based on falling edge detection of external clock signals at the INT0 input pin.
- **Buzzer output**
- **PWM output**



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	TC0CKS	ALOAD0	TC0OUT	PWM0OUT
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 0 **PWM0OUT:** PWM output control bit.
0 = Disable PWM output.
1 = Enable PWM output. PWM duty controlled by TC0OUT, ALOAD0 bits.

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

Bit 2 **ALOAD0:** Auto-reload control bit. **Only valid when PWM0OUT = 0.**
0 = Disable TC0 auto-reload function.
1 = Enable TC0 auto-reload function.

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

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.

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

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 10ms interval time for TC0 interrupt. TC0 clock source is Fcpu (TC0KS=0). High clock is external 4MHz. Fcpu=Fosc/4. Select TC0RATE=010 (Fcpu/64).**

*TC0C initial value = 256 - (TC0 interrupt interval time * input clock)*
*= 256 - (10ms * 4MHz / 4 / 64)*
*= 256 - (10^{-2} * 4 * 10^{6} / 4 / 64)*
= 100
= 64H

The basic timer table interval time of TC0.

TC0RATE	TC0CLOCK	High speed mode (Fcpu = 4MHz / 4)		Low speed mode (Fcpu = 32768Hz / 4)	
		Max overflow interval	One step = max/256	Max overflow interval	One step = max/256
000	Fcpu/256	65.536 ms	256 us	8000 ms	31250 us
001	Fcpu/128	32.768 ms	128 us	4000 ms	15625 us
010	Fcpu/64	16.384 ms	64 us	2000 ms	7812.5 us
011	Fcpu/32	8.192 ms	32 us	1000 ms	3906.25 us
100	Fcpu/16	4.096 ms	16 us	500 ms	1953.125 us
101	Fcpu/8	2.048 ms	8 us	250 ms	976.563 us
110	Fcpu/4	1.024 ms	4 us	125 ms	488.281 us
111	Fcpu/2	0.512 ms	2 us	62.5 ms	244.141 us

8.3.4 TC0R AUTO-LOAD REGISTER

TC0 timer is with auto-load function controlled by ALOAD0 bit of TC0M. When TC0C overflow occurring, TC0R value will load to TC0C by system. It is easy to generate an accurate time, and users don't reset TC0C during interrupt service routine.

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

The equation of TC0R initial value is as following.

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

N is TC0 overflow boundary number. TC0 timer overflow time has five types (TC0 timer, TC0 event counter, TC0 Fcpu clock source, PWM mode and no PWM mode). These parameters decide TC0 overflow time and valid value as follow table.

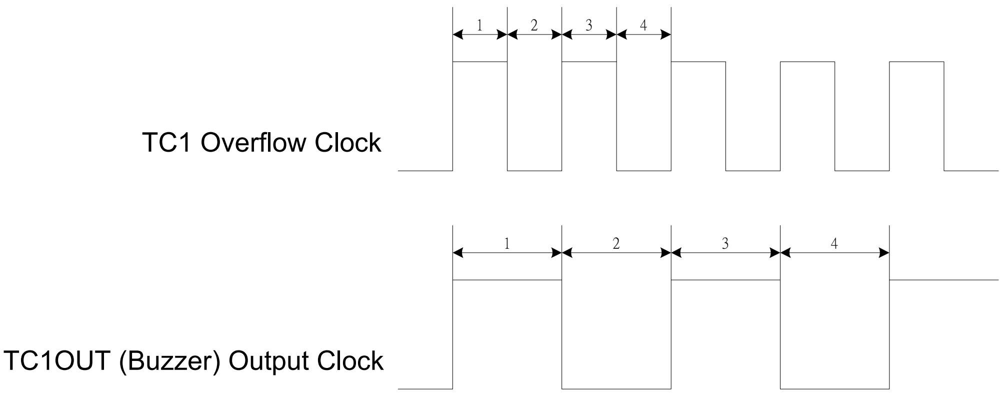
TC0CKS	**PWM0**	**ALOAD0**	**TC0OUT**	**N**	**TC0R valid value**	**TC0R value binary type**
0	0	x	x	256	0x00~0xFF	00000000b~11111111b
	1	0	0	256	0x00~0xFF	00000000b~11111111b
	1	0	1	64	0x00~0x3F	xx000000b~xx111111b
	1	1	0	32	0x00~0x1F	xxx00000b~xxx11111b
	1	1	1	16	0x00~0x0F	xxxx0000b~xxxx1111b
1	-	-	-	256	0x00~0xFF	00000000b~11111111b

- **Example: To set 10ms interval time for TC0 interrupt. TC0 clock source is Fcpu (TC0KS=0) and no PWM output (PWM0=0). High clock is external 4MHz. Fcpu=Fosc/4. Select TC0RATE=010 (Fcpu/64).**

*TC0R initial value = N - (TC0 interrupt interval time * input clock)*
*= 256 - (10ms * 4MHz / 4 / 64)*
*= 256 - (10^{-2} * 4 * 10^{6} / 4 / 64)*
= 100
= 64H

8.3.5 TC0 CLOCK FREQUENCY OUTPUT (BUZZER)

Buzzer output (TC0OUT) is from TC0 timer/counter frequency output function. By setting the TC0 clock frequency, the clock signal is output to P5.4 and the P5.4 general purpose I/O function is auto-disable. The TC0OUT frequency is divided by 2 from TC0 interval time. TC0OUT frequency is 1/2 TC0 frequency. The TC0 clock has many combinations and easily to make difference frequency. The TC0OUT frequency waveform is as following.



- **Example: Setup TC0OUT output from TC0 to TC0OUT (P5.4). The external high-speed clock is 4MHz. The TC0OUT frequency is 0.5KHz. Because the TC0OUT signal is divided by 2, set the TC0 clock to 1KHz. The TC0 clock source is from external oscillator clock. TC0 rate is Fcpu/4. The TC0RATE2~TC0RATE1 = 110. TC0C = TC0R = 131.**

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

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

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

✸ ***Note: Buzzer output is enable, and "PWM0OUT" must be "0".***

8.3.6 TC0 TIMER OPERATION SEQUENCE

TC0 timer operation includes timer interrupt, event counter, TC0OUT and PWM. The 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, TC0OUT and PWM stop.
        B0BCLR      FTC0IEN         ; TC0 interrupt function is disabled.
        B0BCLR      FTC0IRQ         ; TC0 interrupt request flag is cleared.
```

- **Set TC0 timer rate. (Besides event counter mode.)**

```
        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 timer clock source.**

```
; Select TC0 internal / external clock source.
        B0BCLR      FTC0CKS         ; Select TC0 internal clock source.
or
        B0BSET      FTC0CKS         ; Select TC0 external clock source.
```

- **Set TC0 timer auto-load mode.**

```
        B0BCLR      FALOAD0         ; Enable TC0 auto reload function.
or
        B0BSET      FALOAD0         ; Disable TC0 auto reload function.
```

- **Set TC0 interrupt interval time, TC0OUT (Buzzer) frequency or PWM duty cycle.**

```
; Set TC0 interrupt interval time, TC0OUT (Buzzer) frequency or PWM duty.
        MOV         A,#7FH          ; TC0C and TC0R value is decided by TC0 mode.
        B0MOV       TC0C,A          ; Set TC0C value.
        B0MOV       TC0R,A          ; Set TC0R value under auto reload mode or PWM mode.

; In PWM mode, set PWM cycle.
        B0BCLR      FALOAD0         ; ALOAD0, TC0OUT = 00, PWM cycle boundary is 0~255.
        B0BCLR      FTC0OUT
or
        B0BCLR      FALOAD0         ; ALOAD0, TC0OUT = 01, PWM cycle boundary is 0~63.
        B0BSET      FTC0OUT
or
        B0BSET      FALOAD0         ; ALOAD0, TC0OUT = 10, PWM cycle boundary is 0~31.
        B0BCLR      FTC0OUT
or
        B0BSET      FALOAD0         ; ALOAD0, TC0OUT = 11, PWM cycle boundary is 0~15.
        B0BSET      FTC0OUT
```

☞ **Set TC0 timer function mode.**

```
        B0BSET      FTC0IEN       ; Enable TC0 interrupt function.
or
        B0BSET      FTC0OUT       ; Enable TC0OUT (Buzzer) function.
or
        B0BSET      FPWM0OUT      ; Enable PWM function.
```

☞ **Enable TC0 timer.**

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

8.3.7 TC0 TIMER NOTICE

When TC0C value changes from "0xFF" to not "0xFF", TC0IRQ is set "1" whether TC0 is operating or not. If TC0IRQ = 0 and TC0C is changed by program, TC0IRQ might be set as TC0C is from "0xFF" to not "0xFF". The condition makes unexpected TC0 interrupt occurring.

➢ **Example: TC0C = 0xFF and TC0IRQ = 0. TC0IRQ will set as "1" when TC0C is cleared by program (TC0C = 0).**

```
        MOV         A, #0         ; Clear TC0C.
        B0MOV       TC0C, A       ; TC0IRQ changed from "0" to "1".

        B0BSET      FTC0IEN       ; Enable TC0 interrupt function and system jumps to interrupt
                                  ; vector (ORG 8) at next cycle.
```

If TC0C changing in system operating duration is necessary, to disable TC0 interrupt function (TC0IEN = 0) before changing TC0C value. The solution can avoid unexpected TC0 interrupt occurring and example is as following.

➢ **Example: TC0C = 0xFF and TC0IRQ = 0. Clearing TC0C must be after TC0 interrupt disable.**

```
        B0BCLR      FTC0IEN       ; Disable TC0 interrupt function.

        MOV         A, #0         ; Clear TC0C.
        B0MOV       TC0C, A       ; TC0IRQ changed from "0" to "1".

        B0BCLR      FTC0IRQ       ; Clear TC0IRQ flag.

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

✸ ***Note: Disable TC0 interrupt function first, and load new TC0C value into TC0C buffer. This way can avoid unexpected TC0 interrupt occurring.***

8.4 PWM MODE

8.4.1 OVERVIEW

PWM function is generated by TCn timer counter and output the PWM signal to PWMnOUT pin (P5.3/P5.4). The 8-bit counter counts modulus 256, 64, 32, 16 controlled by ALOADn, TCnOUT bits. The value of the 8-bit counter (TCnC) is compared to the contents of the reference register (TCnR). When the reference register value (TCnR) is equal to the counter value (TCnC), the PWM output goes low. When the counter reaches zero, the PWM output is forced high. The low-to-high ratio (duty) of the PWMn output is TCnR/256, 64, 32, 16.

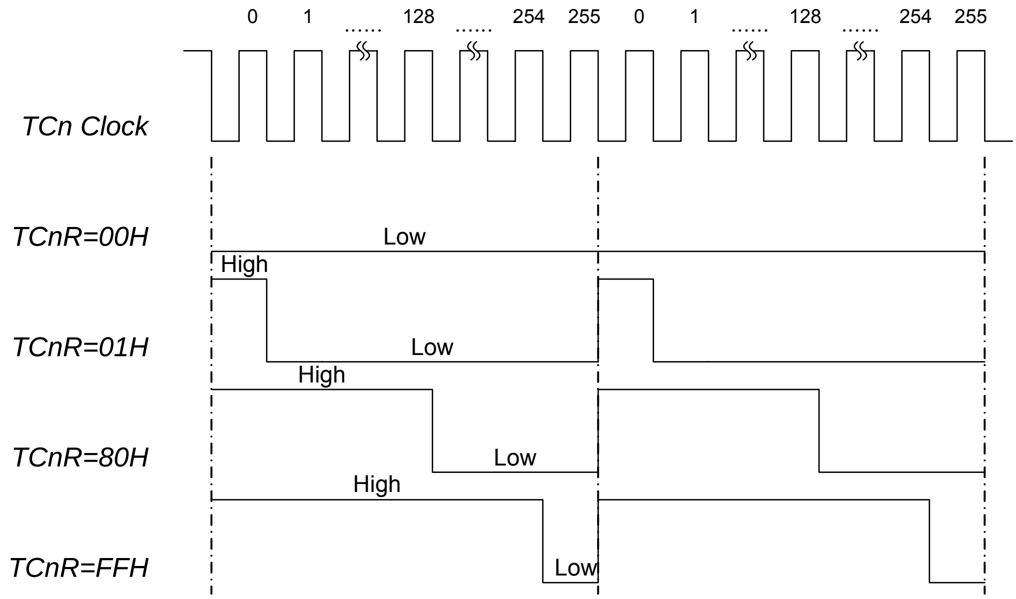
PWM output can be held at low level by continuously loading the reference register with 00H. Under PWM operating, to change the PWM's duty cycle is to modify the TCnR.

✸ ***Note: The "n" of TCn,TCnC… is 0 or 1 follow timer mode. "n=0" is TC0 mode. "n=1" is TC1 mode.***

✸ ***Note: TCn is double buffer design. Modifying TCnR to change PWM duty by program, there is no glitch and error duty signal in PWM output waveform. Users can change TCnR any time, and the new reload value is loaded to TCnR buffer at TCn overflow.***

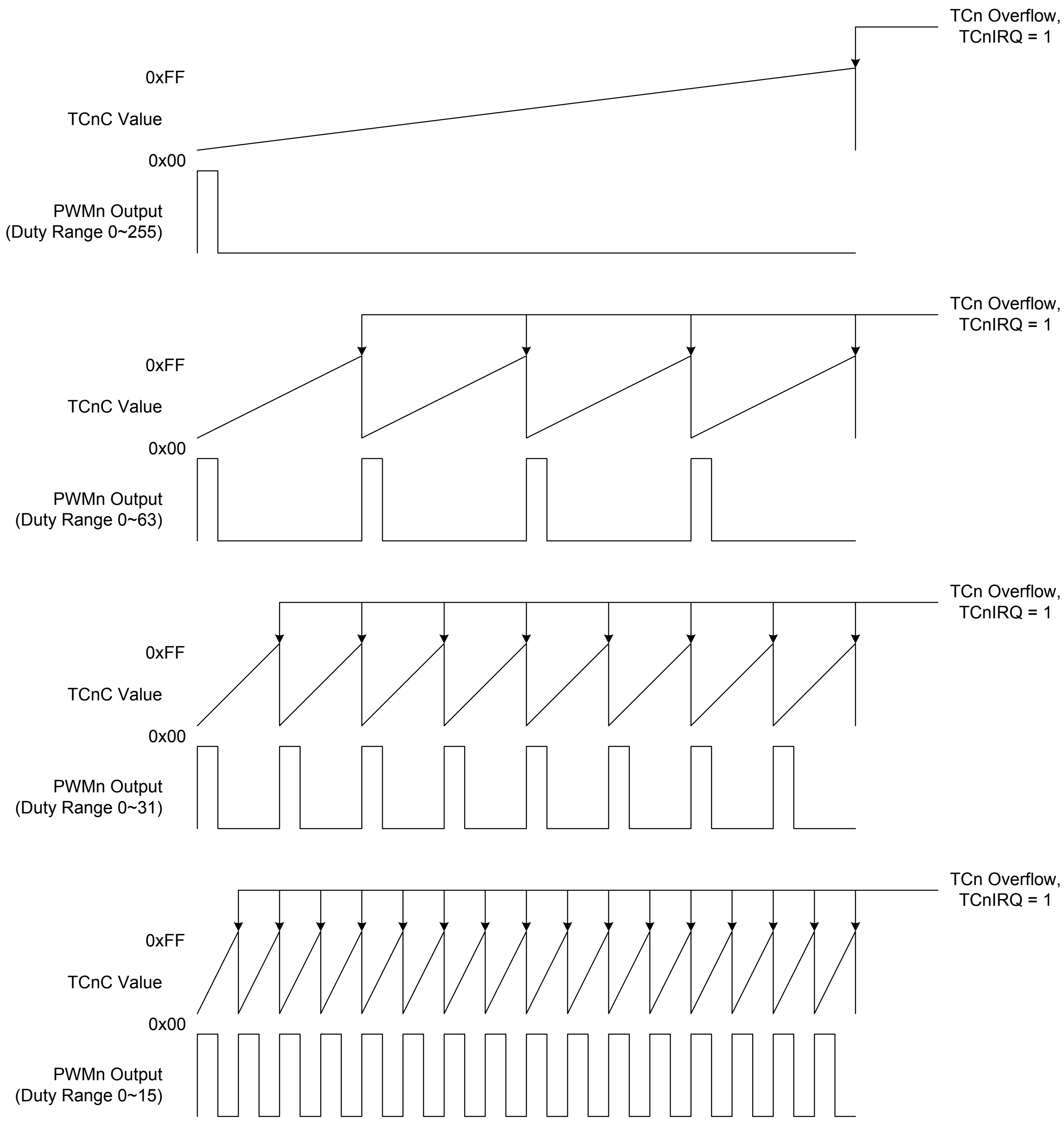
ALOADn	TCnOUT	PWM duty range	TCnC valid value	TCnR valid bits value	MAX. PWM Frequency (Fcpu = 4MHz)	Remark
0	0	0/256~255/256	0x00~0xFF	0x00~0xFF	7.8125K	Overflow per 256 count
0	1	0/64~63/64	0x00~0x3F	0x00~0x3F	31.25K	Overflow per 64 count
1	0	0/32~31/32	0x00~0x1F	0x00~0x1F	62.5K	Overflow per 32 count
1	1	0/16~15/16	0x00~0x0F	0x00~0x0F	125K	Overflow per 16 count

The Output duty of PWM is with different TCnR. Duty range is from 0/256~255/256.



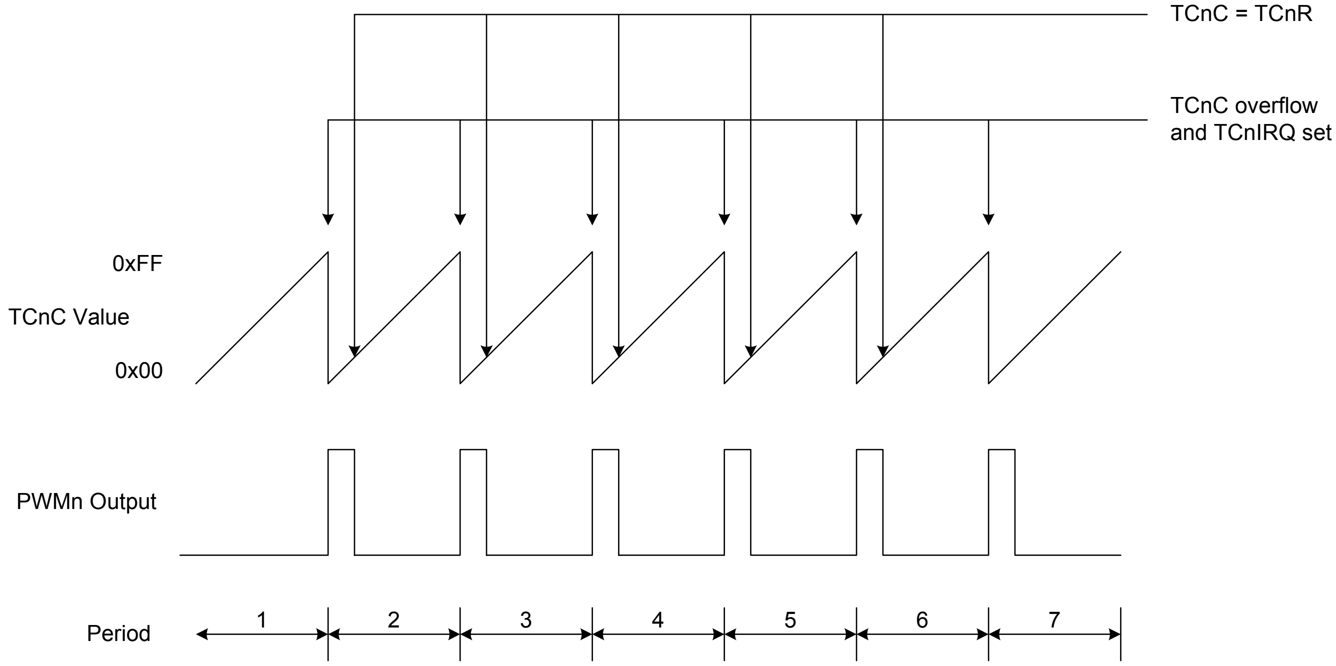
8.4.2 TCnIRQ and PWM Duty

In PWM mode, the frequency of TCnIRQ is depended on PWM duty range. From following diagram, the TCnIRQ frequency is related with PWM duty.

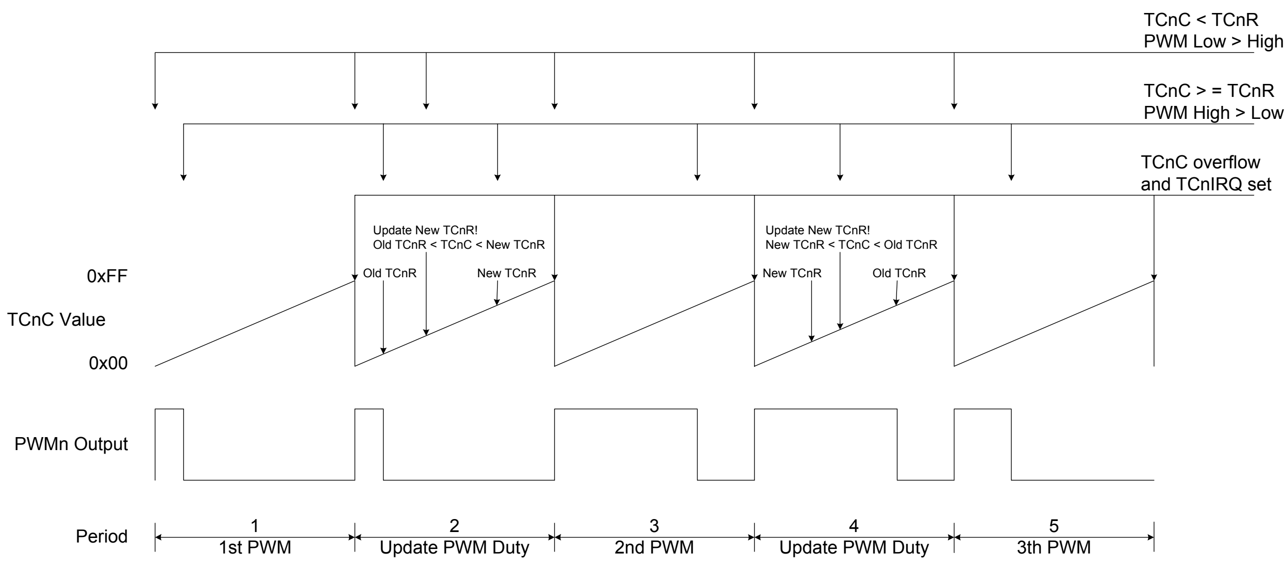


8.4.3 PWM Duty with TCnR Changing

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



Above diagram is shown the waveform with fixed TCnR. In every TCnC overflow PWM output “High, when TCnC≧ TCnR PWM output ”Low”. If TCnR is changing in the program processing, the PWM waveform will became as following diagram.



In period 2 and period 4, new Duty (TCnR) is set. TCn is double buffer design. The PWM still keeps the same duty in period 2 and period 4, and the new duty is changed in next period. By the way, system can avoid the PWM not changing or H/L changing twice in the same cycle and will prevent the unexpected or error operation.

8.4.4 PWM PROGRAM EXAMPLE

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

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

MOV       A,#30           ; Set the PWM duty to 30/256
B0MOV     TC0C,A
B0MOV     TC0R,A

B0BCLR    FTC0OUT         ; Set duty range as 0/256~255/256.
B0BCLR    FALOAD0
B0BSET    FPWM0OUT        ; Enable PWM0 output to P5.4 and disable P5.4 I/O function
B0BSET    FTC0ENB         ; Enable TC0 timer
```

✸ ***Note: The TCnR is write-only register. Don't process them using INCMS, DECMS instructions.***

➢ **Example: Modify TC0R registers' value.**

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

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

✸ ***Note: The PWM can work with interrupt request.***

9 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…), ***"I" can't be E6h, E7h.***	-	-	-	1
	XCH	A,M	A ←→M	-	-	-	1+N
	B0XCH	A,M	A ←→M (bank 0), ***"M" can't be 0x80~0xFF registers.***	-	-	-	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".
3. "I" of "B0MOV M,I" doesn't support "E6h" and "E7h".
4. "M" of "B0XCH" doesn't support 0x80~0xFF system registers.

10 ELECTRICAL CHARACTERISTIC

10.1 ABSOLUTE MAXIMUM RATING

Supply voltage (Vdd).. - 0.3V ~ 6.0V
Input in voltage (Vin).. Vss – 0.2V ~ Vdd + 0.2V
Operating ambient temperature (Topr)
SN8P2602AP, SN8P2602AS, SN8P2602AX .... 0°C ~ + 70°C
SN8P2602APD, SN8P2602ASD, SN8P2602AXD .... –40°C ~ + 85°C
Storage ambient temperature (Tstor) .... –30°C ~ + 125°C

10.2 ELECTRICAL CHARACTERISTIC

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

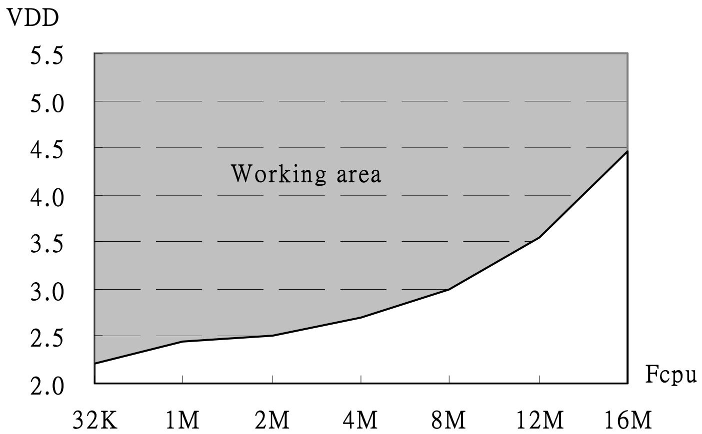
PARAMETER	SYM.	DESCRIPTION		MIN.	TYP.	MAX.	UNIT
Operating voltage	Vdd	Normal mode, Vpp = Vdd		2.4	5.0	5.5	V
RAM Data Retention voltage	Vdr			-	1.5*	-	V
Vdd rise rate	Vpor	Vdd rise rate to ensure internal power-on reset		0.05	-	-	V/ms
Input Low Voltage	ViL1	All input ports		Vss	-	0.3Vdd	V
	ViL2	Reset 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 = 3V		100	200	300	KΩ
		Vin = Vss , Vdd = 5V		50	100	180	
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		-	12*	-	mA
sink current	IoL	Vop = Vss + 0.5V		-	15*	-	
INTn trigger pulse width	Tint0	INT0 interrupt request pulse width		2/fcpu	-	-	cycle
Supply Current	Idd1	normal Mode (No loading, Fcpu = Fosc/4)	Vdd= 5V, 4Mhz	-	2.5	5	mA
			Vdd= 3V, 4Mhz	-	1	2	mA
	Idd2	Slow Mode (Internal low RC, Stop high clock)	Vdd=5V, ILRC 32Khz	-	25	50	uA
			Vdd=3V, ILRC 16Khz	-	5	10	uA
	Idd3	Sleep Mode	Vdd= 5V	-	1	2	uA
			Vdd= 3V	-	0.8	2	uA
	Idd4	Green Mode (No loading, Fcpu = Fosc/4)	Vdd= 5V, 4Mhz	-	0.60	1.2	mA
			Vdd= 3V, 4Mhz	-	0.25	0.5	mA
			Vdd=5V, ILRC 32Khz	-	15	30	uA
			Vdd=3V, ILRC 16Khz	-	3	6	uA
LVD detect level	V_{LVD}	Low voltage detect level		1.5	1.8	2.2	V

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

10.3 CHARACTERISTIC GRAPHS

The Graphs in this section are for design guidance, not tested or guaranteed. In some graphs, the data presented are outside specified operating range. This is for information only and devices are guaranteed to operate properly only within the specified range.

10.3.1 SN8P2602A



Working Voltage vs. Frequency
(Noise Filter Disable、25℃)

11 APPLICATION NOTICE

11.1 Development Tool Version

11.1.1 ICE (In circuit emulation)

- **S8KD-2 ICE:** S8KD-2 ICE is designed for SN8P1XXX series emulation. There are some limitations if use S8KD-2 ICE emulates SN8P2602A please refer following S8KD-2 ICE emulation notice section
- **SN8ICE 2K:** Full function emulates SN8P2604

11.1.2 OTP Writer

- **Writer 3.0:** Support SN8P2602A but no Stand-alone mode.
- **Easy Writer V1.0:** OTP programming is controlled by ICE without firmware upgrade suffers. Please refer easy writer user manual for detailed information.
- **MP-Easy Writer V1.0:** Stand-alone operation to support SN8P2602A mass production

11.1.3 SN8IDE

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

- **For S8KD-2 ICE:** SN8IDE_V1.99R. **SN8IDE V1.99S or later No More support SN8P2000 series**
- **For SN8ICE 2K:** M2IDE_V107 or later
- **For Writer 3.0, Easy Writer and MP-Easy Writer:** M2IDE V1.04 or later

11.2 CODE OPTION

11.2.1 NOISE FILTER CODE OPTION

For high EFT request condition, Noise_Filter enable is necessary. Enable Noise_Filter can reduce external noise affecting system of operating. **If Noise_Filter enable, the Fcpu is limited Fosc/4~Fosc/128.**

	Noise_Filter	
	Enable	Disable
Fcpu	**Fosc/4~Fosc/128**	**Fosc/1~Fosc/128**

11.2.2 WATCHDOG

Watchdog of SN8P2602A includes three operations as Enable, Disable, Always On controlled by code option.

- **Watchdog Enable:** If watchdog timer overflow during program running, the system resets. Watchdog timer stops in green mode and power down mode (sleep mode). After system wake-up, watchdog timer is running again.

- **Watchdog Always On:** Watchdog timer keeps running in green mode or power down mode (sleep mode). If watchdog overflow occurs under the two modes, the system would be reset. It is not easy to make system to keep green mode/ power down mode situations. The main purpose of the design is supporting high EFT application. The watchdog timer guards system to work well and good reliability in high noisy situation.

11.3 INTERRUPT VECTOR (ORG 8)

When one of interrupt requests occurs, system will jump to interrupt vector and execute interrupt service routine. The first instruction of interrupt vector (ORG 8) must be “JMP” or “NOP”. The SN8ASM199L and later version compilers alerts the message if the interrupt vector first instruction is not “JMP” or “NOP”.

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

```
.CODE
                ORG     0               ; 0000H
                JMP     START           ; Jump to user program address.
                …
                ORG     8               ; Interrupt service routine
                NOP                     ; The first instruction at ORG 8.

                …
                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: The interrupt service routine is following user program.**

```
.CODE
                ORG     0               ; 0000H
                JMP     START           ; Jump to user program address.
                …
                ORG     08
                JMP     MY_IRQ          ; 0008H, Jump to interrupt service routine address.

                ORG     10H
START:                                  ; 0010H, The head of user program.
                …                       ; User program.
                …
                …
                JMP     START           ; End of user program.
                …
MY_IRQ:                                 ;The head of interrupt service routine.
                …
                RETI                    ; End of interrupt service routine.
                …
                ENDP                    ; End of program.
```

11.4 INSTRUCTION

11.4.1 B0MOV M,I

"M" of "B0MOV M, I" is working register (Y,Z,R,PFLAG)(0x80~0x86). "I" is immediate data.

> ✸ ***Note: The "I" can't be "E6H" and "E7H". If "I" of "B0MOV M, I" is "E6H" or "E7H", S8ASM V1.99K and later version will alert and show the error message on screen.***

Users need to check the value of "B0MOV M,I" instruction again. This instruction is often used for look-up table function to set table address. It is easy to check the table address from listing file (.LST) after compiling. User also can use ORG to set table start address and avoid "E6H" and "E7H".

➢ **Example: Set table start address by ORG.**

```
                ORG         0x0100            ; Set table address from 0x0100.
Table:
                DW          0x9876
                …
```

11.4.2 B0XCH A, M

"B0XCH A, M" exchanges A and M contents. It only supports user defined RAM.

> ✸ ***Note: The "M" can't be system register area (0x80~0xFF).***

➢ **Example: Save PFLAG.**

Error case!

```
                B0XCH       A, PFLAG          ; PFLAG is a system register. "B0XCH A, M" can't process it.
                B0MOV       PFLAGBUF, A
```

Correct case!

```
                B0MOV       A, PFLAG
                B0MOV       PFLAGBUF, A       ; PFLAGBUF is user defining RAM,
```

➢ **Example: Save ACC.**

```
ACCBUF          DS          1                 ; Define ACCBUF to be ACC buffer.
                ORG         0
                …
                ORG         10
                …
                B0XCH       A, ACCBUF
                …
```

11.5 S8KD-2 ICE ENULATION

SN8IDE is SONIX 8-bit development software including Assembler/ICE Debugger/OTP Writer. S8KD-2 is SONIX 8-bit ICE EV chip. There are different specifications between SN8P2602A and EV Chip. SONIX provides macros to solve the difference and make emulation correct. SN8IDE_V1.99R_S8KD2 and later version support these macros.

> ✷ ***Note:***
> 1. ***Please use SN8IDE V1.99R to develop projects!***
> 2. ***SN8IDE V1.99S or later No More support SN8P2000 series emulation.***

11.5.1 ICE_MODE

ICE_MODE setting is necessary. ICE_MODE = 1 supports S8KD-2 ICE emulation. ICE_MODE = 0 is real chip mode.

Syntax: ICE_MODE Val

Val: 0 = Real chip. 1 = S8KD-2 ICE emulation.

➢ **Example: Setting ICE mode for ICE emulation. After compiling, the code only supports ICE emulation and can’t work correct in real chip.**

```
CHIP        SN8P2602A
.DATA

ICE_MODE    EQU          1               ; Set ICE_MODE for ICE emulation.

            INCLUDESTD   SN8P2X_ICE.H
.CODE
            User program
            …
```

➢ **Example: Setting ICE mode for real chip. After compiling, the code only supports SN8P2602A and can’t work correct in ICE emulation.**

```
CHIP        SN8P2602A
.DATA

ICE_MODE    EQU          0               ; Set ICE_MODE for real chip.

            INCLUDESTD   SN8P2X_ICE.H
.CODE
            User program
            …
```

> ✷ ***Note:1. After ICE emulation and verifying all functions, set ICE_MODE = 0 and compile firmware again for real chip programming.***
> ***2. Please use the ICE_MODE = 0 checksum for real chip code. Don’t use ICE_MODE = 1 checksum which only support ICE emulation, not real chip.***

11.5.2 INSTRUCTION CYCLE

I Instruction cycles of some instructions are different between SN8P2602A and EV chip. These differences makes ICE instruction timing isn't consistent with SN8P2602A. SN8IDE assembler provides some macros to solve instruction cycle difference as following. Users just only use built-in instruction macro to replace corresponding instruction. In "ICE_MODE EQU 1" ICE emulation mode, assembler maybe insert some extra code to synchronize instruction timing between ICE and real chip. Therefore, the maximum available ROM size is larger than real chip. In "ICE_MODE EQU 0" the ROM size is same as real chip.

SN8P2602A				S8KD-2 EV CHIP				INSTRUCTION MACRO	DESCRIPTION
Field	Mnemonic		Cycle	Field	Mnemonic		Cycle		
	MOV	A,M	1		MOV	A,M	1	-	1. M = user define RAM, N = 0. M = system register, N = 1. 2. EV Chip : Read OSCM = 1 cycle Write OSCM = 2 cycle SN8P2602A : Read OSCM = 1 cycle Write OSCM = 1 cycle 3. PUSH,POP instructions are different between SN8P2602A and S8KD-2 ICE.
M	MOV	M,A	1	M	MOV	M,A	1	-	
O	B0MOV	A,M	1	O	B0MOV	A,M	1	-	
V	B0MOV	M,A	1	V	B0MOV	M,A	1	-	
E	MOV	A,I	1	E	MOV	A,I	1	-	
	B0MOV	M,I	1		B0MOV	M,I	1	-	
	XCH	**A,M**	**1+N**		**XCH**	**A,M**	**1**	**@XCH A,M**	
	B0XCH	**A,M**	**1+N**		**B0XCH**	**A,M**	**1**	**@B0XCH A,M**	
	MOVC		2		MOVC		2	-	
	ADC	A,M	1		ADC	A,M	1	-	
A	**ADC**	**M,A**	**1+A**	A	**ADC**	**M,A**	**1**	**@ADC M,A**	
R	ADD	A,M	1	R	ADD	A,M	1	-	
I	**ADD**	**M,A**	**1+N**	I	**ADD**	**M,A**	**1**	**@ADD M,A**	
T	**B0ADD**	**M,A**	**1+N**	T	**B0ADD**	**M,A**	**1**	**@B0ADD M,A**	
H	ADD	A,I	1	H	ADD	A,I	1	-	
M	SBC	A,M	1	M	SBC	A,M	1	-	
E	**SBC**	**M,A**	**1+N**	E	**SBC**	**M,A**	**1**	**@SBC M,A**	
T	SUB	A,M	1	T	SUB	A,M	1	-	
I	**SUB**	**M,A**	**1+N**	I	**SUB**	**M,A**	**1**	**@SUB M,A**	
C	SUB	A,I	1	C	SUB	A,I	1	-	
	AND	A,M	1		AND	A,M	1	-	
L	**AND**	**M,A**	**1+N**	L	**AND**	**M,A**	**1**	**@AND M,A**	
O	AND	A,I	1	O	AND	A,I	1	-	
G	OR	A,M	1	G	OR	A,M	1	-	
I	**OR**	**M,A**	**1+N**	I	**OR**	**M,A**	**1**	**@OR M,A**	
C	OR	A,I	1	C	OR	A,I	1	-	
	XOR	A,M	1		XOR	A,M	1	-	
	XOR	**M,A**	**1+N**		**XOR**	**M,A**	**1**	**@XOR M,A**	
	XOR	A,I	1		XOR	A,I	1	-	
	SWAP	M	1		SWAP	M	1	-	
P	**SWAPM**	**M**	**1+N**	P	**SWAPM**	**M**	**1**	**@SWAPM M**	
R	RRC	M	1	R	RRC	M	1	-	
O	**RRCM**	**M**	**1+N**	O	**RRCM**	**M**	**1**	**@RRCM M**	
C	RLC	M	1	C	RLC	M	1	-	
E	**RLCM**	**M**	**1+N**	E	**RLCM**	**M**	**1**	**@RLCM M**	
S	CLR	M	1	S	CLR	M	1	-	
S	**BCLR**	**M.b**	**1+N**	S	**BCLR**	**M.b**	**1**	**@BSET M.b**	
	BSET	**M.b**	**1+N**		**BSET**	**M.b**	**1**	**@BCLR M.b**	
	B0BCLR	**M.b**	**1+N**		**B0BCLR**	**M.b**	**1**	**@B0BSET M.b**	
	B0BSET	**M.b**	**1+N**		**B0BSET**	**M.b**	**1**	**@B0BCLR M.b**	
	CMPRS	A,I	1 + S		CMPRS	A,I	1 + S	-	
B	CMPRS	A,M	1 + S	B	CMPRS	A,M	1 + S	-	
R	INCS	M	1 + S	R	INCS	M	1 + S	-	
A	**INCMS**	**M**	**1+N+S**	A	**INCMS**	**M**	**1 + S**	**@INCMS M**	
N	DECS	M	1 + S	N	DECS	M	1 + S	-	
C	**DECMS**	**M**	**1+N+S**	C	**DECMS**	**M**	**1 + S**	**@DECMS M**	
H	BTS0	M.b	1 + S	H	BTS0	M.b	1 + S	-	
	BTS1	M.b	1 + S		BTS1	M.b	1 + S	-	
	B0BTS0	M.b	1 + S		B0BTS0	M.b	1 + S	-	
	B0BTS1	M.b	1 + S		B0BTS1	M.b	1 + S	-	
M	JMP	d	2	M	JMP	d	2	-	
I	CALL	d	2	I	CALL	d	2	-	
S	RET		2	S	RET		2	-	
C	RETI		2	C	RETI		2	-	
	NOP		1		NOP		1	-	
	PUSH		1		PUSH		1		
	POP		1		POP		1		

✸ ***Note: S8KD-2 ICE can't emulate SN8P2602A's "PUSH, POP" instructions.***

✸ ***Note: These macros are built in "SN8P2X_ICE.H". The file must be included in user program.***

➢ **Example: Including SN8P2X_ICE.H in user program.**

```
CHIP        SN8P2602A
.DATA
ICE_MODE    EQU           0

            INCLUDESTD    SN8P2X_ICE.H      ; SN8P2X_ICE.H is a standard macro file and included by
                                            "INCLUDESTD".

.CODE
            User program…
            …
```

➢ **Example: Instructions are replaced by instruction macro.**

```
CHIP        SN8P2602A
.DATA
ICE_MODE    EQU           0

            INCLUDESTD    SN8P2X_ICE.H

.CODE
            User program…

;           ADD           BUF1, A
            @ADD          BUF1, A           ; "ADD M,A" is replaced by "@ADD M,A".
            …
;           AND           BUF1, A
            @AND          BUF1, A           ; "AND M,A" is replaced by "@AND M,A".
            …
```

11.5.3 SYSTEM CLOCK

SONIX 2 series 8-bit MCU has multi-system clock (Fosc/1~Fosc/8,Fosc/64,Fosc/128,), but ICE is fixed Fosc/4. In ICE emulation, user must be sure the Fcpu speeds of SN8P2602A and ICE are identical. Before emulation, change ICE crystal frequency to match with SN8P2602A system clock.

➢ **Example: SN8P2602A system clock vs. ICE Table. The SN8P2602A clock source frequency is external 4MHz crystal.**

SN8P2602A Fcpu Option	Fcpu Frequency	ICE Fcpu	ICE Crystal Frequency
Fosc/1	4MHz	Fosc/4	16MHz
Fosc/2	2MHz	Fosc/4	8MHz
Fosc/4	1MHz	Fosc/4	4MHz
Fosc/8	0.5MHz	Fosc/4	2MHz

✸ ***Note: For different speed crystals, modifying ICE "HIGH CLK" option is necessary. If ICE crystal is 16MHz, set "HIGH CLK" options to "X'TAL 12M".***

➢ **Example: SN8P2602A system clock vs. ICE Table. The SN8P2602A clock source frequency is external 16MHz crystal.**

SN8P2602A Fcpu Option	Fcpu Frequency	ICE Fcpu	ICE Crystal Frequency
Fosc/1	16MHz	Fosc/4	-
Fosc/2	8MHz	Fosc/4	-
Fosc/4	4MHz	Fosc/4	16MHz
Fosc/8	2MHz	Fosc/4	8MHz

✸ ***Note: ICE external high clock crystal is up to 16MHz. Fcpu = Fosc/1 and Fosc/2 can't be emulated. For the applications, users should use SN8P2602A to verify functions.***

11.5.4 WATCHDOG TIMER

Watchdog timer clear routine of SN8P2602A is setting WDTR register 0x5A. S8KD-2 ICE is not. SN8IDE provides "@RST_WDT" macro to make watchdog timer function correctly.

➢ **Example: Reset watchdog timer by setting WDTR as 0x5A.**

```
CHIP        SN8P2602A
.DATA
ICE_MODE    EQU          0

            INCLUDESTD   SN8P2X_ICE.H

.CODE
            User program…

            MOV          A, #5Ah           ; Reset watchdog timer.
            B0MOV        WDTR, A

            …
```

✸ ***Note: This way can't be emulated in S8KD-2 ICE. Using following routine to replace it.***

➢ **Example: Reset watchdog timer by "@RST_WDT".**

```
CHIP        SN8P2602A
.DATA
ICE_MODE    EQU          0

            INCLUDESTD   SN8P2X_ICE.H

.CODE
            User program…

            @RST_WDT                       ; Reset watchdog timer by macro.

            …
```

11.5.5 P0 EMULATION

SN8P2602A's P0.0 is bi-direction I/O, but ICE's P0.0 is input only. PEDGE controls are different, too. SN8IDE provides macros to control P0.0 emulation. These macros are built in assembler software.

11.5.5.1 @P00_MODE

Syntax: @P00_MODE Val

Val: 0 = Set P0.0 input mode. 1 = Set P0.0 output mode.

➢ **Example: Set P0.0 as input mode.**

```
        @P00_MODE    0
```

✸ ***Note: If P0.0 set as input mode, the input pin is P0.0 of S8KD-2 ICE.***

➢ **Example: Set P0.0 as output mode.**

```
        @P00_MODE    1
```

✸ ***Note: If P0.0 set as output mode, the output pin is P6.0 of S8KD-2 ICE.***

11.5.5.2 @P00_OUT

Syntax: @P00_OUT Val

Val: 0 = Set P0.0 output low. 1 = Set P0.0 output high.

➢ **Example: Set P0.0 as output high.**

```
        @P00_OUT     1
```

➢ **Example: Set P0.0 as output low.**

```
        @P00_OUT     0
```

✸ ***Note: Under P0.0 output mode, the signal is output from P6.0 of S8KD-2 ICE.***

11.5.5.3 PEDGE

P00G[1:0] of PEDGE register definition is different from S8KD-2 ICE. ICE emulation and real chip PEDGE function are different.

PEDGE		SN8P2602A	S8KD-2 ICE
P00G1	P00G0		
0	0	Reserved	Reserved
0	1	**Rising Edge**	**Falling Edge**
1	0	**Falling Edge**	**Rising Edge**
1	1	Bi-Direction	Bi-Direction

SONIX provides "@P00_EDGE" macro to emulate real chip PEDGE function in ICE. The ICE_MODE must be 1 for ICE Emulation. After ICE emulation, set ICE_MODE as 0 and compile again to get SN8 file for real chip.

Syntax: @P00_EDGE Val

Val: 1 = Rising edge. 2 = Falling edge. 3 = Level change (bi-direction).

- **Example: Set P0.0 interrupt trigger as rising edge and ICE emulation.**

```
CHIP        SN8P2602A
.DATA
ICE_MODE    EQU         1               ; Set ICE mode.

            INCLUDESTD  SN8P2X_ICE.H

.CODE
            User program…

            @P00_EDGE   1               ; Set P0.0 interrupt trigger as rising edge.
            B0BSET      FP00IEN
            …
```

- **Example: Set P0.0 interrupt trigger as falling edge and for real chip "SN8".**

```
CHIP        SN8P2602A
.DATA
ICE_MODE    EQU         0               ; Set real chip mode.

            INCLUDESTD  SN8P2X_ICE.H

.CODE
            User program…

            @P00_EDGE   2               ; Set P0.0 interrupt trigger as falling edge.
            B0BSET      FP00IEN
            …
```

11.5.6 PWM DUTY

PWM duty of SN8P2602A is controlled by ALOAD0, TC0OUT bits.

PWM0OUT = 1					
ALOAD0	**TC0OUT**	**TC0R Boundary**	**PWM duty range**	**Max PWM Frequency (Fcpu = 4M)**	**Note**
0	0	00h to FFh	0/256 ~ 255/256	7.8125K	Overflow per 256 count
0	1	00h to 3Fh	0/64 ~ 63/64	31.25K	Overflow per 64 count
1	0	00h to 1Fh	0/32 ~ 31/32	62.5K	Overflow per 32 count
1	1	00h to 0Fh	0/16 ~ 15/16	125K	Overflow per 16 count

S8KD-2 ICE doesn't support PWM duty setting function. SONIX provides PWM Duty setting macro. Users can use it to emulate PWM function and don't affect other functions. The macro is built in assembler software. Users have to set ICE_MODE as ICE or real chip.

Syntax: @PWM0_MAX_DUTY Max_Duty

Max_Duty	TC0 Overflow Boundary	PWM Duty Range	PWM Resolution
256	FFh to 00h	0/255 ~ 255/256	8-bit
64	3Fh to 40h	0/63 ~ 63/64	6-bit
32	1Fh to 20h	0/31 ~ 31/32	5-bit
16	0Fh to 10h	0/15 ~ 15/16	4-bit

➢ **Example: Set PWM Max. Duty = 64,Duty = 2:1.**

```
CHIP        SN8P2602A
.DATA
ICE_MODE    EQU              0        ; Set real chip mode.

            INCLUDESTD       SN8P2X_ICE.H

.CODE
            User program…

            @PWM0_MAX_DUTY   64       ; Set PWM0 max. duty as 64.

            MOV              A,#42    ; 42 = 63 (Max. TC0R) / 3 X 2
            B0MOV            TC0R,A
            B0BSET           FPWM0OUT
            B0BSET           FTC0ENB
            …
```

11.5.7 OTHER MACRO

A macro routine includes many instructions. It will be error after test instruction with skipping function.

```
            BTS0        BUF.0
            @RST_WDT
            JMP         TEST_CODE
            …
TEST_CODE:
            …
```

BTS0 instruction skipping function only skip one instruction. @RST_WDT is a macro and composed of several instructions. The skipping function of above routine would be error. It can't skip to "JMP TEST_CODE" successfully. Using following routines can solve the problem.

```
            BTS0        BUF.0
            JMP         CLR_WDT
            JMP         TEST_CODE
            …
CLR_WDT:
            @RST_WDT
            …
TEST_CODE:
            …
```

SN8IDE provides user defined forward/backward jump directive to processing skipping function easier. "Macro_Start" and "Macro_End" is user define label name. Using @@**.Macro_Start** and @@**.Macro_End** to syntax. These lable names can be repeated in main program.

```
            BTS0        BUF.0
            JMP         @F.Macro_Start    ; Jump to nearest user define @@.Macro_Start:
            JMP         @F.Macro_End      ; Jump to nearest user define @@.Macro_End:
            …
@@.Macro_Start:
            @RST_WDT
@@.Macro_End:

TEST_CODE:
            …
```

> ✸ ***Note: Only S8ASM V1.99K or later version support user defined forward/backward jump directive!***

> ✸ ***Note: Macro possible affects Accumulator and PFLAG result. Use have to check it!***

12 OTP PROGRAMMING PIN

12.1.1 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
JP2 for Writer V3.0 transition board

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

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

JP3 for MP transition board

12.1.2 The pin assignment of Writer V3.0 and V2.5 transition board socket:

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

Writer V2.5 JP1 Pin Assignment

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

Writer V3.0 JP1 Pin Assignment

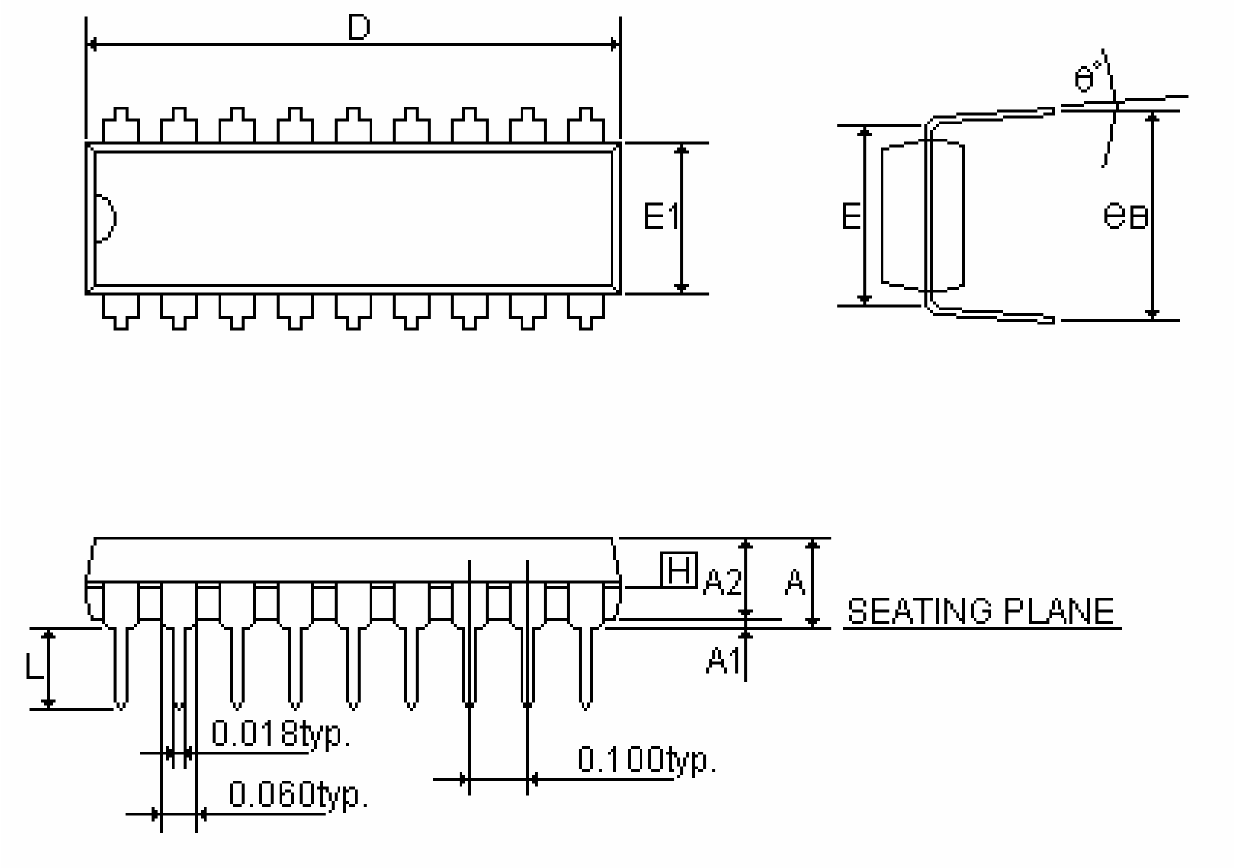
✸ ***Note: For supporting the body programming, SONIX writer V2.5 must update V3.0 firmware and modify circuit. Please contact SONIX agent about SONIX Writer V2.5 upgrade.***

12.1.3 SN8P2602A Programming Pin Mapping:

Programming Information of SN8P2600A Series											
Chip Name				**SN8P2602A**		**SN8P2602AX**					
Writer V2.5 Connector		**EZ Writer / Writer V3.0 Connector**		**OTP IC / JP3 Pin Assigment**							
Number	**Name**	**Number**	**Name**	**Number**	**Pin**	**Number**	**Pin**				
2	VDD	1	VDD	14	VDD	15,16	VDD				
1	GND	2	GND	5	VSS	5,6	VSS				
4	CLK	3	CLK	6	P5.0	7	P5.0				
3	CE	4	CE	-	-	-	-				
6	PGM	5	PGM	17	P1.0	19	P1.0				
5	OE	6	OE	7	P5.1	8	P5.1				
8	D1	7	D1	-	-	-	-				
7	D0	8	D0	-	-	-	-				
10	D3	9	D3	-	-	-	-				
9	D2	10	D2	-	-	-	-				
12	D5	11	D5	-	-	-	-				
11	D4	12	D4	-	-	-	-				
14	D7	13	D7	-	-	-	-				
13	D6	14	D6	-	-	-	-				
16	VDD	15	VDD	-	-	-	-				
15	VPP	16	VPP	4	RST	4	RST				
18	HLS	17	HLS	-	-	-	-				
17	RST	18	RST	-	-	-	-				
-	-	19	-	-	-	-	-				
-	-	20	ALSB/PDB	18	P1.1	20	P1.1				

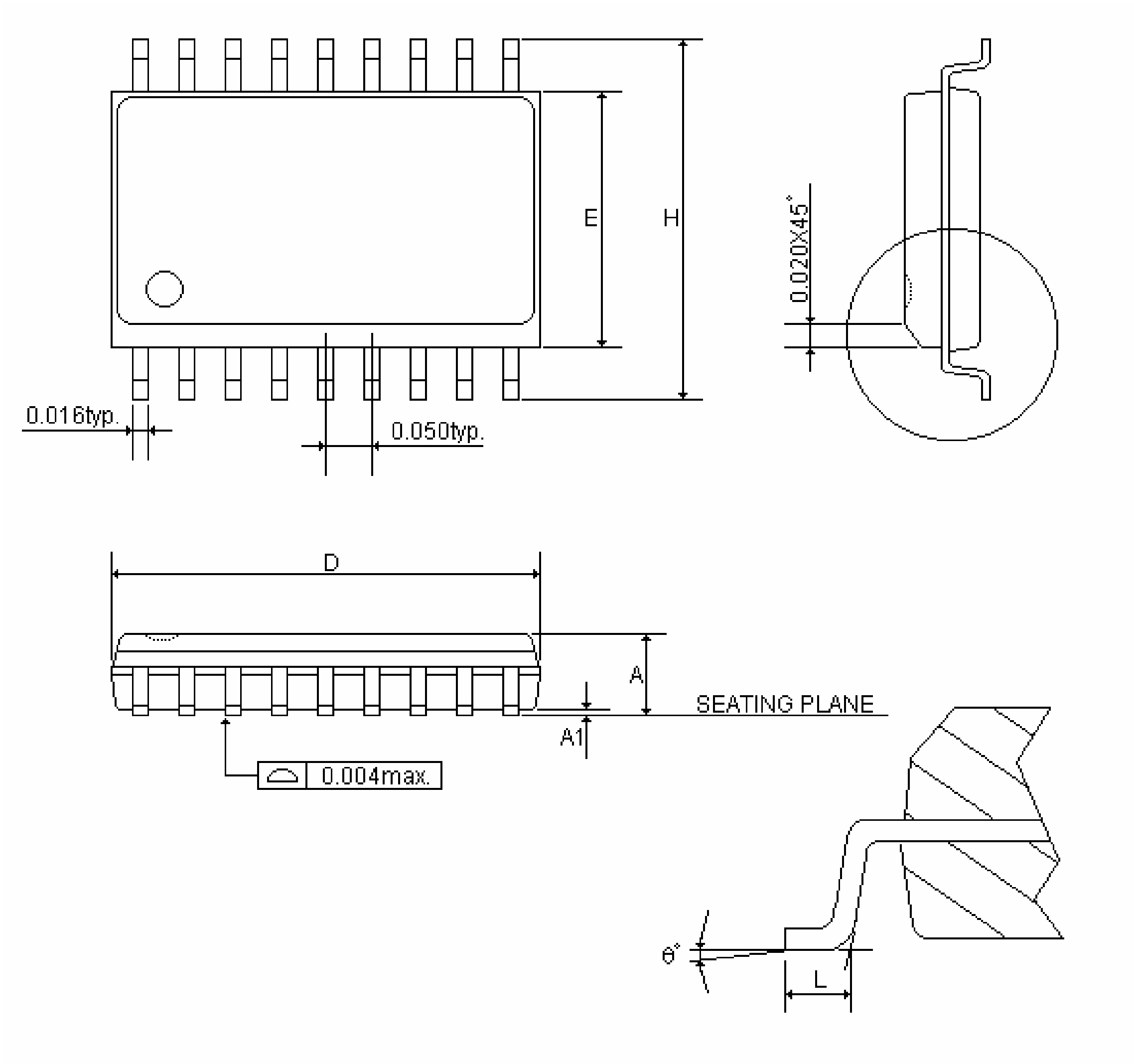
13 PACKAGE INFORMATION

13.1 P-DIP 18 PIN



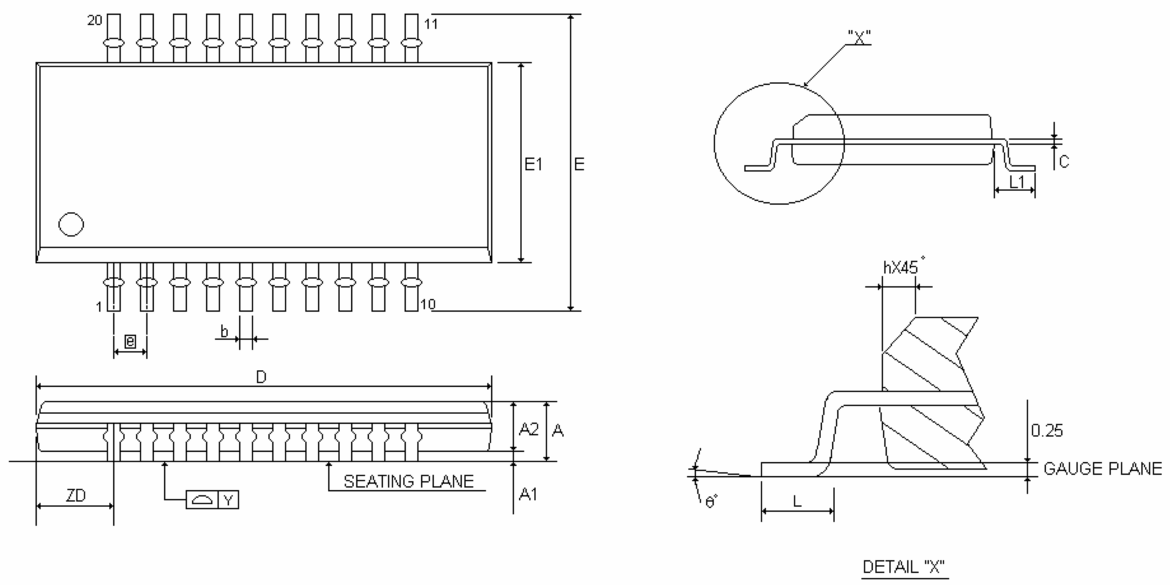
SYMBOLS	MIN	NOR	MAX	MIN	NOR	MAX
	(inch)			(mm)		
A	-	-	0.210	-	-	5.334
A1	0.015	-	-	0.381	-	-
A2	0.125	0.130	0.135	3.175	3.302	3.429
D	0.880	0.900	0.920	22.352	22.860	23.368
E	0.300			7.620		
E1	0.245	0.250	0.255	6.223	6.350	6.477
L	0.115	0.130	0.150	2.921	3.302	3.810
eB	0.335	0.355	0.375	8.509	9.017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

13.2 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°

13.3 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°

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

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

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

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