JE8MCU – TO3 Evaluation

Sample received day: 13 June 2018

Type of samples: 48pin and 20pin

Prepared by: Kamny Tang (kamnytang@astri.org)

Contents

[1 Summary 4](#_Toc518890049)

[1.1 TO3 new or fix 4](#_Toc518890050)

[1.2 Module status 5](#_Toc518890051)

[2 General 6](#_Toc518890052)

[3 Internal RC Oscillator 7](#_Toc518890053)

[3.1 80MHz/40MHz 7](#_Toc518890054)

[3.2 Default RC oscillator frequency 11](#_Toc518890055)

[4 Clock source, frequency and current 12](#_Toc518890056)

[4.1 External 32M Oscillator 12](#_Toc518890057)

[4.2 Internal 40M RC Oscillator 13](#_Toc518890058)

[4.3 Internal 80M RC Oscillator (48pin only) 13](#_Toc518890059)

[4.4 32K RC Oscillator 14](#_Toc518890060)

[4.5 Lowest current consumption 15](#_Toc518890061)

[5 New bootrom 16](#_Toc518890062)

[5.1 Setup 16](#_Toc518890063)

[5.2 ISD Mode (enhance flow) 17](#_Toc518890064)

[5.3 Keil ISD Debugger 18](#_Toc518890065)

[6 Module Test 20](#_Toc518890066)

[6.1 RTC 20](#_Toc518890067)

[6.2 WDT 22](#_Toc518890068)

[6.3 EXINT 24](#_Toc518890069)

[6.4 Dongle 26](#_Toc518890070)

[6.5 ADC/ DAC Test 27](#_Toc518890071)

[6.6 Timer 29](#_Toc518890072)

[6.6.1 Timer0/1 29](#_Toc518890073)

[6.6.2 Timer3 29](#_Toc518890074)

[6.6.3 Timer4 30](#_Toc518890075)

[6.6.4 Timer5 30](#_Toc518890076)

[6.6.5 Timer6 30](#_Toc518890077)

[6.7 LVD 32](#_Toc518890078)

[6.8 OpAmp/Comparator 35](#_Toc518890079)

[6.9 SPI 38](#_Toc518890080)

[6.10 E-Flash mode (Boot from external flash) 42](#_Toc518890081)

[6.11 I2C 44](#_Toc518890082)

[6.12 PWM 48](#_Toc518890083)

[6.13 ROM table 51](#_Toc518890084)

[7 Appendix: 52](#_Toc518890085)

[7.1 MACRO & Sub-routine 52](#_Toc518890086)

# Summary

## TO3 new or fix

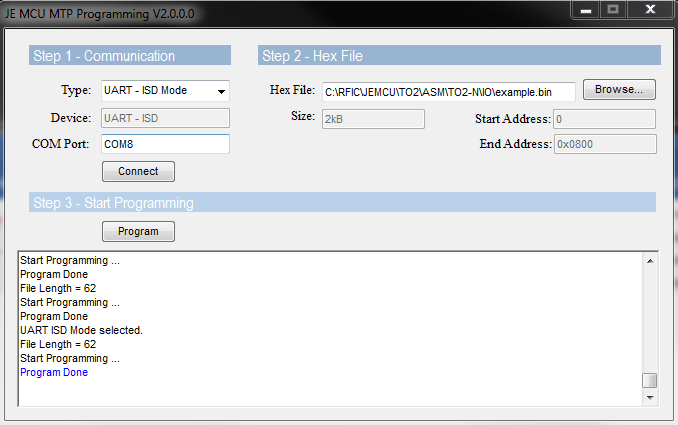
|  |  |  |  |
| --- | --- | --- | --- |
| # | Item | Status | Comment |
| 1 | ISD | OK | Baud rate 4800 bps |
| 2 | In ISD program MTP, after run program in normal mode | OK | Programmer need modified to baud rate to 4800 bps |
| 3 | 48pin can run to 80/40MHz by using internal 80/40Mhz RC oscillator  20pin able to run 40MHz (test just toggle IO) | OK | Oscillator need to configure to 80MHz as default is 40MHz |
| 4 | Enter analog mode successfully | OK | 32K Oscillator, default is 30.4KHz, 40MHz RC Oscillator, measured value is 40.7MHz |
| 5 | Enhance bootrom, ISD mode can use internal RC oscillator as clock source to download program | OK | Set EEPROM(7)=0x00006600 |
| 6 | Connect to Keil ISD debugger | OK | Baud rate change to 4800 |
| 7 | RTC | OK | Can generate interrupt |
| 8 | Watchdog (WDT) | OK | WDT can reset MCU now |
| 9 | EXINT and STOP mode | OK |  |
| 10 | Dongle | OK |  |
| 11 | System Clock Use 32K | OK |  |
| 12 | E-Flash mode functional | OK |  |

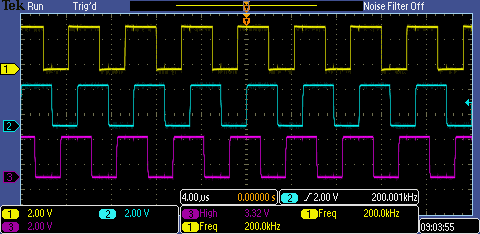
## Module status

|  |  |  |  |
| --- | --- | --- | --- |
| # | **Module** | **TO1 Status** | **TO3 status** |
| 1 | UART(1,2) | Functional OK | Functional OK, ISD can download program |
| 2 | SPI | Functional OK | Functional OK |
| 3 | Timer | Functional OK | T0/1, T3,T4,T5,T6 OK |
| 4 | GPIO | Functional OK | Functional OK, IO can toggle |
| 5 | I2C | Functional OK, Master TX need to set I2CCON.AA | Functional OK |
| 6 | RTC | Design found issue in RTL code | Functional OK |
| 7 | Watchdog | Once watchdog reset, it will repeat reset in bootloader routine due to bootloader time > refresh time. | Functional OK |
| 8 | Dongle | Functional OK, but not match still can access MTP | Functional OK, require match to write MTP, read eeprom |
| 9 | OPAMP & COMP | Functional OK | Functional OK |
| 10 | LDO | Functional OK | Functional OK, chip operate |
| 11 | LVD | Functional OK | Functional OK |
| 12 | 80MHz RC Oscillator | Functional OK | Functional OK |
| 13 | Crystal Oscillator | Functional OK | External oscillator ok |
| 14 | DAC | Functional OK | Functional OK |
| 15 | ADC | Functional OK | Functional OK |
| 16 | RC 32K Oscillator | As RTC issue, can’t test this module | Functional OK, default 30.1KHz |
| 17 | SFR & eSFR | Functional OK | Functional OK |
| 18 | RAM | Functional OK | Functional OK |
| 19 | MTP | Functional OK, loss data when power-up raise time > 20ms | Functional OK, program can write to MTP and copy to RAM in normal mode. |
| 20 | e-Flash | Seems IO can’t support default data rate | Functional OK, external flash is 32-bit address |
| 21 | PWM | Functional OK | Functional OK |
| 22 | Rom table |  | Can read back table value |

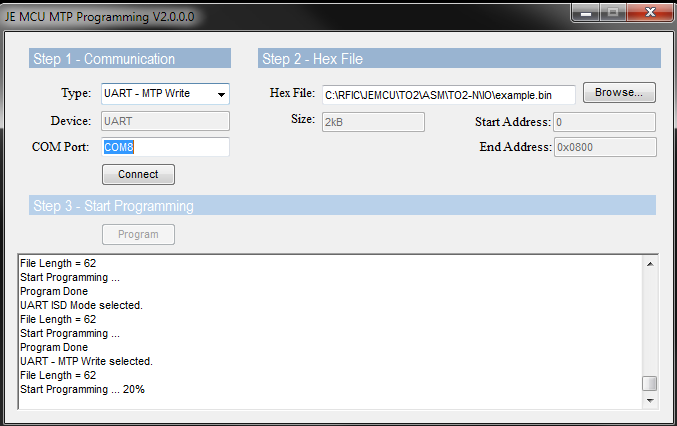
# General

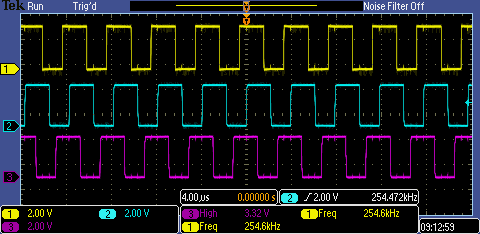
**ISD mode, baud rate 4800bps download to RAM then start immediately (use external 32M oscillator)**





**In ISD mode, program MTP content => Restart/Reset => user program run in Normal mode**



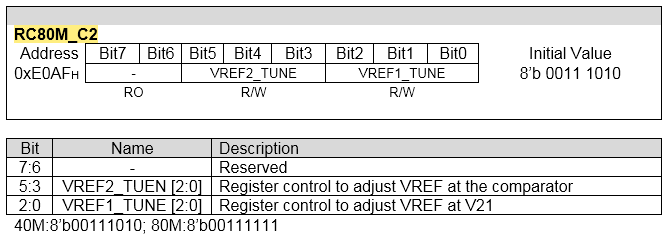


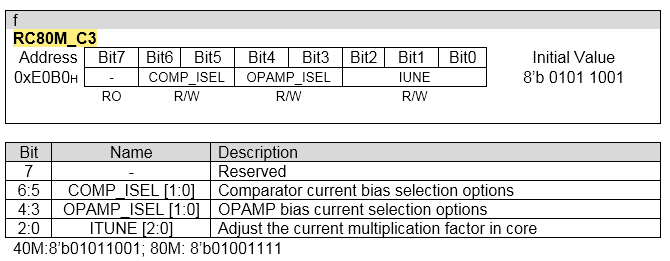
* **ISD mode, Normal mode, 40M RC oscillator, MTP, Bootrom functional ok**

# Internal RC Oscillator

## 80MHz/40MHz

Select 80MHz RC as clock source





**Setup a timer to confirm Fbus is 80MHz**

Test Program:

PROG\_OFFSET EQU 02000h

ORG PROG\_OFFSET

LJM PSTART

ORG PROG\_OFFSET+0Bh ;T0

LJMP T0\_ISR

START:

MOV SP,#50H ;init stack pointer

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;Config 80M RC

**EMOV RC80M\_C2, 3Fh**

**EMOV RC80M\_C3, 4Fh**

**EMOV RC80M\_RES, 60h**

**EMOV RC80M\_CAP, 40h**

EMOV P0\_FN\_H, 00h ;init (P0.7-P0.4)

EMOV P0DD, 0C0h ;(P0.7-P0.6) are output

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

EMOV CLK\_DIV\_L, 00h

EMOV CLK\_DIV\_H, 0h

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

SETB ET0

;SETB ET1

SETB EA ;enable interrupt

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

MOV TMOD,#22H ;Timer0,1 in mode 2

**MOV TH0, #00H**  ;Count from 0 to 255

;

MOV TCON, #00H

;EMOV T01\_DIV\_L,0FFH ;div 256

EMOV T01\_DIV\_L,00h

EMOV T01\_DIV\_H,00h

SETB TR0 ;enable timer0

Main\_loop:

SJMP $

T0\_ISR:

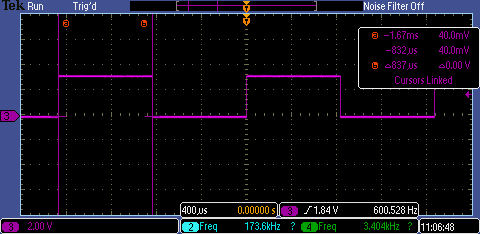
CPL P0.6

RETI

END

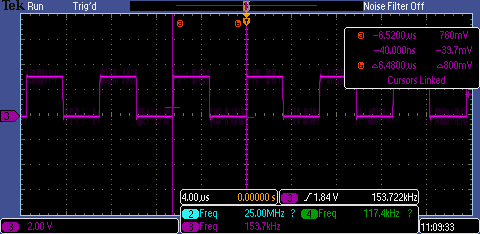
EMOV T01\_DIV\_L,0FFH ;div 256

* **600.528 \* 2 \* 256 \* 256 = 78.7MHz**



EMOV T01\_DIV\_L,00h

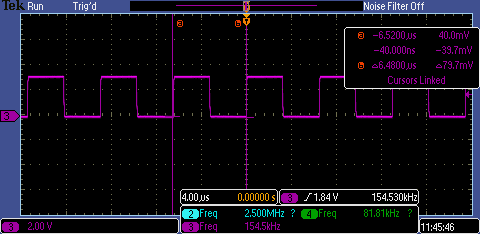
* **153.722K \* 2 \* 256 = 78.7MHz**



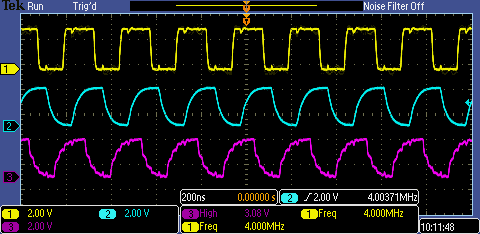
Another chip

EMOV T01\_DIV\_L,00h

* **154.530K \* 2 \* 256 = 79.1MHz**



**Toggle IO, use internal 80MHz RC as clock source**



Test program:

ORG PROG\_OFFSET

LJMP START

START:

MOV SP,#50H ;init s0tack pointer

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;; Clock Selection

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

#ifdef XTAL

EMOV 0E09Eh, 05h

EMOV P0\_FN\_L, 10h ;select external 32M clock source => P0.2 as clock input

; MOV CLK\_SEL\_PD, #05h ;select 32M external clock

MOV CLK\_SEL\_PD, #05h ;select 32M external clock

#endif

#ifdef RC80M

EMOV RC80M\_C2, 3Fh

EMOV RC80M\_C3, 4Fh

#endif

#ifdef RC40M

EMOV RC80M\_C2, 3Ah

EMOV RC80M\_C3, 59h

#endif

EMOV RC80M\_RES, 60h

EMOV RC80M\_CAP, 40h

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

EMOV CLK\_DIV\_L, 8h

EMOV CLK\_DIV\_H, 0h

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

EMOV P0DD, 0FFh ;(P0.7-P0.0) are output

EMOV P1DD, 0FFh ;(P1DD): Input/Output Select, 0=input, 1=output

EMOV P2DD, 0FFh ;(P2DD): P2

CONTINUE:

MOV P0,#00h

MOV P1,#00h

MOV P2,#00h

MOV P0,#0FFh

MOV P1,#0FFh

MOV P2,#0FFh

SJMP CONTINUE

END

## Default RC oscillator frequency

* *Enter Analog test mode:*
* ***Entry method:***

1. *P00, P01 =00*
2. *Press reset*
3. *SPI interface send data*

*Pin2.1 CLK (SCK)*

*Pin2.0 Data (MOSI*

*Reset (SS)*

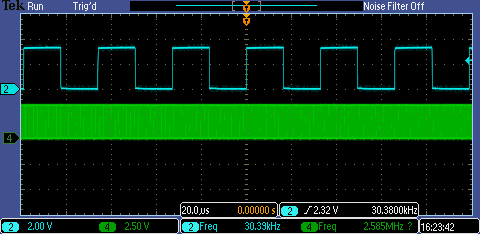
*Send 0xFE (3’b110)*

32K RC oscillator (P1.2)

* 30.39KHz

HSRC Out (P0.3)

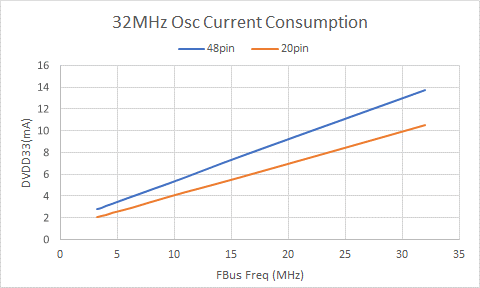
* 2.54MHz (DIV16) => 40.64 MHz



# Clock source, frequency and current

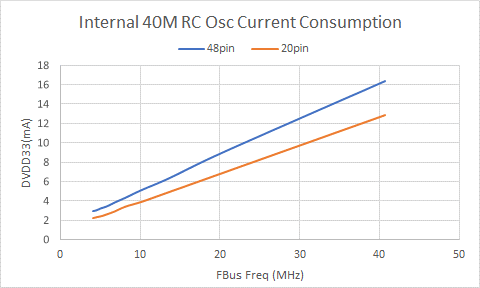
## External 32M Oscillator

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  | 48pin | | 20pin | |
| CLK\_DIV\_L | Fbus | IO toggle | AVDD33  (mA) | DVDD33  (mA) | AVDD33  (mA) | DVDD33  (mA) |
| 0, 1 | 32M | 1.6 M | 2.449 | 13.7 | 2.3 | 10.5 |
| 2 | 16M | 800K | 2.449 | 7.7 | 2.3 | 5.8 |
| 3 | 10.7M | 533.3K | 2.449 | 5.6 | 2.3 | 4.3 |
| 4 | 8M | 400K | 2.449 | 4.6 | 2.3 | 3.5 |
| 5 | 6.4M | 320K | 2.449 | 4.0 | 2.3 | 3 |
| 6 | 5.3M | 266.7K | 2.449 | 3.6 | 2.3 | 2.7 |
| 7 | 4.6M | 228.6K | 2.449 | 3.3 | 2.3 | 2.5 |
| 8 | 4.0M | 200K | 2.449 | 3.1 | 2.3 | 2.3 |
| 9 | 3.6M | 177.8K | 2.449 | 2.9 | 2.3 | 2.2 |
| 10 | 3.2M | 160K | 2.449 | 2.8 | 2.3 | 2.1 |



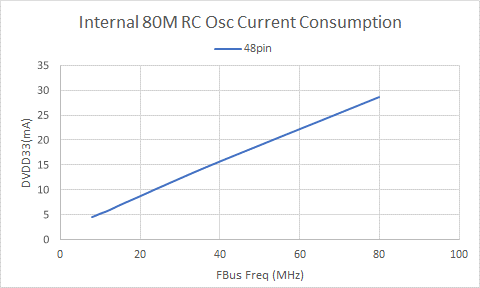
## Internal 40M RC Oscillator

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  | 48pin | | 20pin | |
| CLK\_DIV\_L | Fbus | IO toggle | AVDD33  (mA) | DVDD33  (mA) | AVDD33  (mA) | DVDD33  (mA) |
| 0, 1 | 40.7M | 2.04M | 2.449 | 16.4 | 2.3 | 12.9 |
| 2 | 20.4M | 1.02M | 2.449 | 9.0 | 2.3 | 6.9 |
| 3 | 13.6 | 678.7K | 2.449 | 6.3 | 2.3 | 4.9 |
| 4 | 10.2 | 509.0K | 2.449 | 5.1 | 2.3 | 3.9 |
| 5 | 8.1 | 407.2K | 2.449 | 4.3 | 2.3 | 3.4 |
| 6 | 6.8 | 339.3K | 2.449 | 3.8 | 2.3 | 2.9 |
| 7 | 5.8 | 290.9K | 2.449 | 3.4 | 2.3 | 2.6 |
| 8 | 5.1 | 254.5K | 2.449 | 3.2 | 2.3 | 2.4 |
| 9 | 4.5 | 226.2 | 2.449 | 3.0 | 2.3 | 2.3 |
| 10 | 4.1 | 203.6K | 2.449 | 2.9 | 2.3 | 2.2 |



## Internal 80M RC Oscillator (48pin only)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| CLK\_DIV\_L | Fbus | IO toggle | AVDD33(mA) | DVDD33(mA) |
| 0, 1 | 80M | 4.0M | 3.641 | 28.7 |
| 2 | 40M | 2.0M | 3.645 | 15.7 |
| 3 | 26.7M | 1.33M | 3.645 | 11.1 |
| 4 | 20M | 1.0M | 3.646 | 8.7 |
| 5 | 16M | 800.1K | 3.645 | 7.3 |
| 6 | 13.3M | 667.3K | 3.645 | 6.3 |
| 7 | 11.4M | 572.1K | 3.646 | 5.6 |
| 8 | 10M | 500.6K | 3.646 | 5.19 |
| 9 | 8.9 | 444.9 | 3.646 | 4.8 |
| 10 (Default) | 8.9 | 400.4 | 3.646 | 4.5 |



## 32K RC Oscillator

Use internal 32k RC Oscillator

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  | 48pin (mA) | | 20pin (mA) | |
| CLK\_DIV\_L | Fbus | IO toggle | AVDD33 | DVDD33 | DVDD33 | AVDD33 |
| 0, 1 | 32K | 1.52k | 2.4 | 1.26 | 0.61 | 2.3 |
| 2 | 16K | 760.8Hz | 2.4 | 1.24 | 0.61 | 2.3 |
| 10 | 3.2K | 152.1Hz | 2.4 | 0.84 | 0.63 | 2.3 |

## Lowest current consumption

Use 32K RC Oscillator

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Power Down module (default sysclk div= 1 | 48pin (mA) | | 20pin (mA) | |
| AVDD33 | DVDD33 | DVDD33 | AVDD33 |
| 80M/40M RC OSC  (80M\_PD=1, (32M\_PD=1 or 0 no change) |  |  | 0.47 **\*\*** | 0.976 |
| + ADC/DAC(default PD)/LVD/OpAmp(Default PD) |  |  | 0.442 | 0.931 |
| + Power down 2.5V LDO |  |  | 0.442 | 0.777 |
| + Power down Temperature sensor |  |  | 0.442 | 0.539 |
| + Power down MTP | **0.555mA** | **0.423mA** | **0.224** | **0.538** |

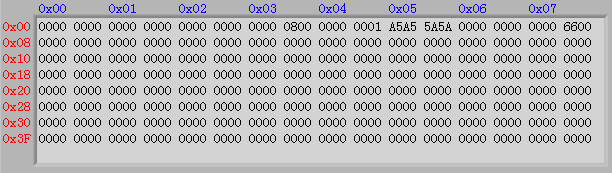
* + 80M/40M RC Oscillator consume 1.46mA \*\*
    - 1.32mA from AVDD33, 0.14mA from DVDD33
  + 2.5V LDO use 0.154mA
  + Temp sensor use 0.234mA
  + MTP use 0.218mA
* **Less than 1mA**

# New bootrom

## Setup

ISD/MTP ok

Enable new bootrom flow by set eeprom(7)=0x00006600



Program to set eeprom(7) to 0x00006600, in ISD mode, download following program to change the eeprom content.

ORG PROG\_OFFSET

MOV R1,#00h ;bit 0-7

MOV R2,#66h ;bit 8-15

MOV R3,#00h ;bit16-24

MOV R4,#00h ;bit25-31

MOV SP,#50H ;init stack pointer

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;Select external 32MHz clock

; EMOV P0\_FN\_L, 10h ;select external 32M clock source

; MOV CLK\_PD\_CON, #05h

; EMOV CLK\_DIV\_L, 01h

EMOV CLK\_DIV\_L, 04h

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;Port 1

EMOV P1\_FN\_L, 00h ;(P1\_FN\_L): P1.0-P1.3 mux selection

EMOV P1\_FN\_H, 50h ;(P1\_FN\_H): P1.4-P1.7 mux selection

EMOV P1DD, 0FFh ;(P1DD): Input/Output Select, 0=input, 1=output

main:

mov dptr, #0h

mov a,#0

movc A, @A+DPTR

;; EEPROM Write

mov A, #0x53

mov dptr, #MTPCON2

movx @dptr, A

UP\_ReadDone:

; Writing EEPROM

mov dps, #0

; Writing Destination Address

mov A, #0x00

mov dptr, #mtpsadd2

movx @dptr, A

mov R5,#0x07 ;EEPROM Address

mov b,#1

mov dps, #0

mov A, R1

mov dptr, #mtpdata1

movx @dptr, A

mov A, R2

mov dptr, #mtpdata2

movx @dptr, A

mov A, R3

mov dptr, #mtpdata3

movx @dptr, A

mov A, R4

mov dptr, #mtpdata4

movx @dptr, A

mov A, #11

mov dptr, #mtpcon1

movx @dptr, A ; write operation start

UP\_loopEEPROM:

mov dptr, #mtpcon1

movx a, @dptr

anl a, #0x20

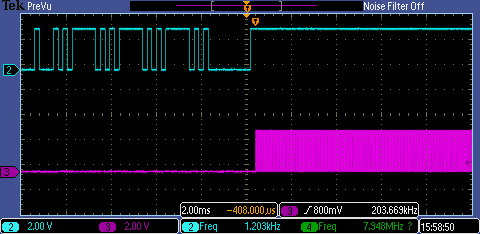
jz UP\_loopEEPROM ; Loop until writing done

jmp $

end

## ISD Mode (enhance flow)

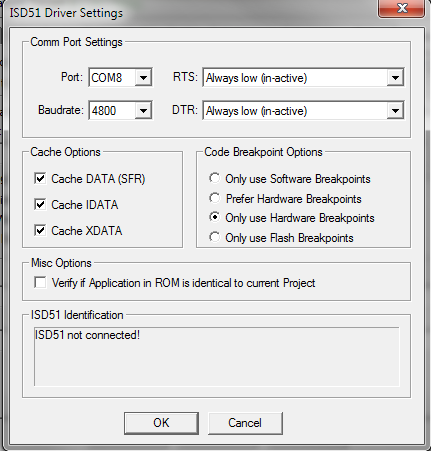
* It’s ok to disable on-board 32M oscillator in enhance flow. MCU will use internal 40M RC Oscillator as clock source.



ISD download to RAM

Program run on RAM

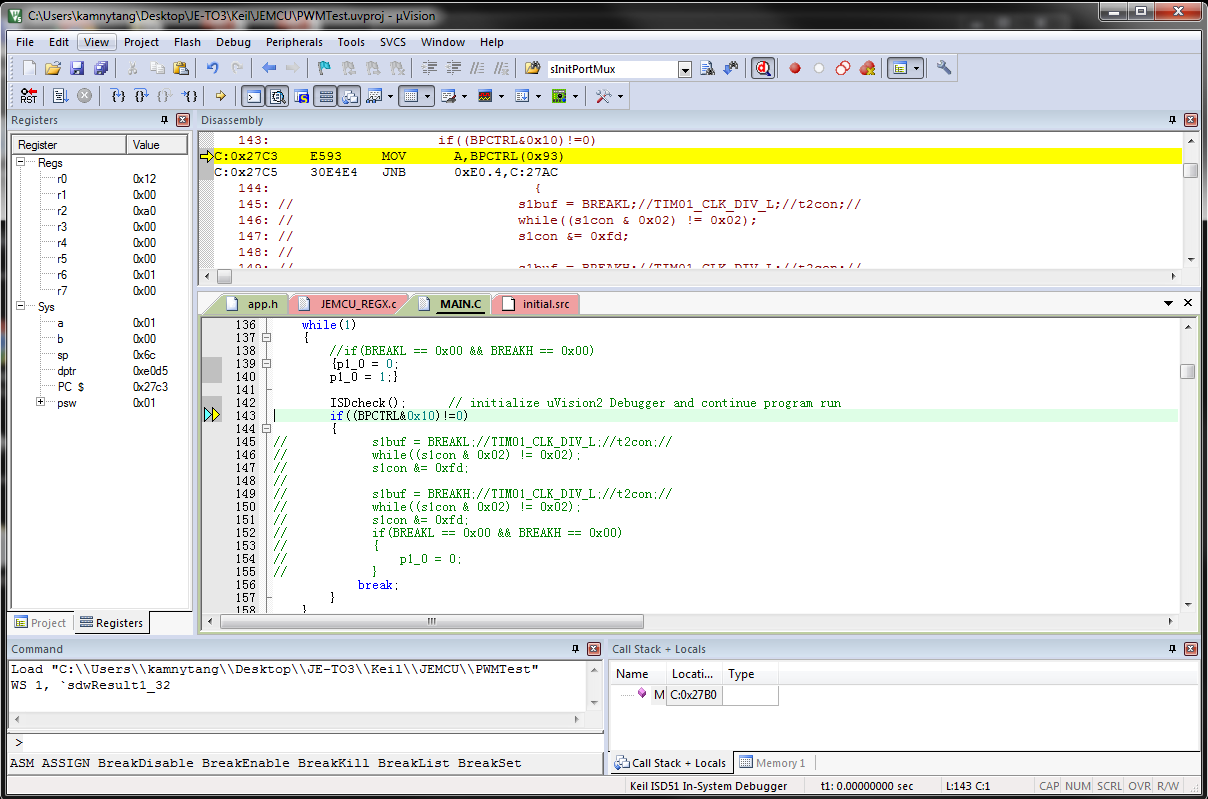
## Keil ISD Debugger



* Change baud rate to 4800
* Set Port number to match your PC

Procedure:

* Compile in Keil
* Download bin file to MCU in ISD mode
* Press Keil debug button



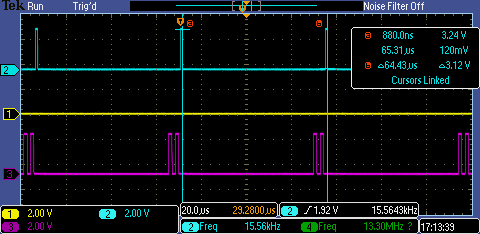
Debug button

# Module Test

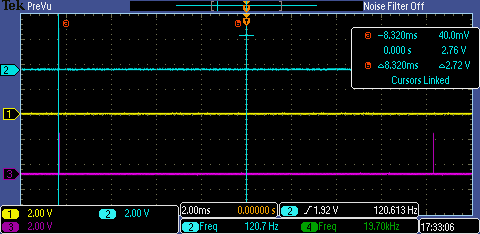
## RTC

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Clock Source | RTCMODH | RTCMODL | RTCPS | Time Rep |  |
| 32M Osc | 00 | 01 | 0 | 6443us |  |
| 32M Osc | 00 | 01 | 1 | 128.8us |  |
| 32M Osc | 00 | 01 | F | 1.03ms |  |
| 32K RC Osc | 00 | 01 | 0 | 8.32ms |  |
| 32K RC Osc | 00 | 01 | F | 133ms |  |
| 32K RC Osc | 00 | 02 | 0 | 16.6ms |  |

Use 32MHz Oscillator as clock source



Use 32K RC Oscillator as clock source



Test Program:

ORG PROG\_OFFSET

LJMP START

ORG PROG\_OFFSET + 06Bh ;11. IE6 (RTC, LVD18, LVD33, T4, HWBP)

LJMP RTC\_ISR

RETI

START:

MOV SP,#50H ;init stack pointer=

MOV 8Fh, #05h ;select 32M external clock

EMOV P0DD, 0FFh

MOV P0,#00h

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

Start1:

MOV IEN0,#80h ;bit8-All,

MOV IEN1,#20h ;bit5-RTC, bit4-SPI

EMOV RTCMODH,00H

EMOV RTCMODL,02H

EMOV RTCCTRL,40H ;32K RC

; EMOV RTCCTRL,50H ;32M Ext clock

; EMOV RTCCTRL,60H ;32K RC

SJMP $

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

RTC\_ISR:

NOP

NOP

NOP

CLR P0.0

NOP

NOP

NOP

SETB P0.0

NOP

NOP

NOP

CLR P0.0

NOP

NOP

NOP

SETB P0.0

NOP

NOP

NOP

CLR P0.0

SETB P0.7

NOP

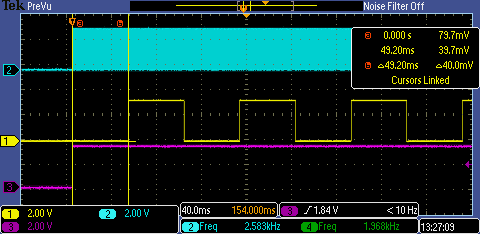
CPL P0.7

RETI

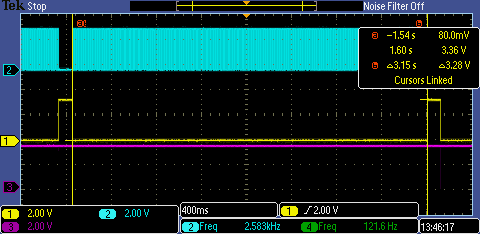
* Summary of RTC, functional is0 ok, source can use 32K RC Osc and 32MHz oscillator.

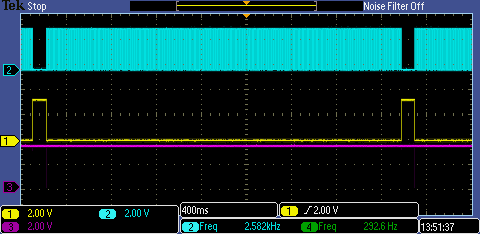
## WDT

Periodic refresh the watchdog (yellow line, signal line 1), signal 2 toggle as normal.



Without refresh the watchdog, signal 2(blue) will reset around each 3s.





Test program

ORG PROG\_OFFSET+ 100h

START:

EMOV P2\_FN\_L, 00h ;(P2\_FN\_L): init (P2.3-P2.0)

EMOV P2\_FN\_H, 00h ;(P2\_FN\_H): init (P2.7-P2.4)

EMOV P2DD, 0FFh ;(P2DD):

MOV P2, #00h

SETB EA ;enable interrupt

#if ENABLE\_WDT

;enable Watchdog

MOV WDTREL, #80h ; 6xx ms

; MOV WDTREL, #0FFh ; 5.x ms

SETB P2.7

MOV R3,#0FFh

ORL IEN0,#40h ;WDT

ORL IEN1,#40h ;SWDT

#endif

LOOP\_Test:

DJNZ R3, CONT\_TO

#ifdef ENABLE\_WDT\_REFRESH

MOV R3,#0FFh

CPL P2.5

ORL IEN0,#40h ;WDT

ORL IEN1,#40h ;SWDT

#endif

CONT\_TO:

CPL P2.0

MOV R2,#0FFh

XLOOP: NOP

DJNZ R2,XLOOP

SJMP LOOP\_Test

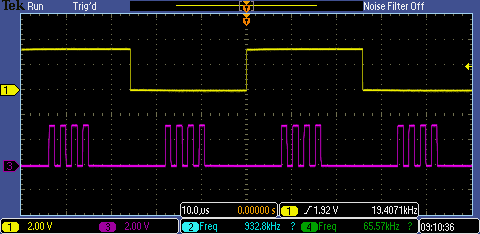
END

* Summary of WDT, functional is ok, it can reset the MCU without refresh the wdt flag.

## EXINT

* Level trigger ok
* Rising edge ok
* Falling edge ok
* Both edge ok
* STOP mode can wake-up by EXINT level trigger (LOW)
  + \*\* Enter STOP mode current may different each time
  + STOP 0.8mA /1.2mA, Running 1.8mA

INT4 generated



Test Program

;#define RISING 1

;#define FALLING 1

#define BOTH\_EDGE 1

;#define LOW\_LEVEL 1

ORG PROG\_OFFSET

LJMP START

ORG PROG\_OFFSET + 003h ;1. IE0 (EXINT1,2,3,4)

LJMP IE0\_ISR

RETI

START:

SETB IEN0.0

SETB IEN0.7

MOV SP,#50H ;init stack pointer=

; EMOV P0\_FN\_L, 00h ;init (P0.3-P0.6)

EMOV P0\_FN\_H, 50h ;init (P0.7-P0.4)

EMOV P0DD, 0FFh ;(P0.7-P0.6) are output

EMOV P1\_FN\_L, 00h ;init (P1.3-P1.0)

EMOV P1\_FN\_H, 00h

EMOV P1DD, 0FFh ;(P1.7-P1.0) are output

EMOV P2\_FN\_L, 00h ;(P2\_FN\_L): init (P2.3-P2.0)

EMOV P2\_FN\_H, 00h ;(P2\_FN\_H): init (P2.7-P2.4)

EMOV P2DD, 0FCh ;(P2DD): P2 all output

MOV P2,#0FFh

MOV P0,#00h

MOV P1,#00h

MOV P2,#00h

EMOV P2\_FN\_L, 05h ;EXINT1

; EMOV EXINT\_EN, 01h

EMOV EXINT\_EN, 0Fh

#ifdef FALLING

SETB IT0 ; IT0=TCON.0, fall edge

EMOV EXINT\_CTRL,00h ;EXINT1 falling

#endif

; EMOV EXINT\_CTRL,01h ;EXINT1 Rising ede

#ifdef RISING

SETB IT0 ; IT0=TCON.0, fall edge

EMOV EXINT\_CTRL,55h ;EXINT1 Rising ede

#endif

; EMOV EXINT\_CTRL,02h ;EXINT1 Both edge

#ifdef BOTH\_EDGE

SETB IT0 ; IT0=TCON.0, fall edge

EMOV EXINT\_CTRL,0AAh ;EXINT1 Both edge

#endif

; EMOV EXINT\_CTRL,03h ;EXINT1 Low level

#ifdef LOW\_LEVEL

CLR IT0

EMOV EXINT\_CTRL,0FFh ;EXINT1 Low level

#endif

Main\_loop:

; CPL P0.0

; ORL PCON,#02h ;STOP MODE wake-up test

SJMP Main\_loop

IE0\_ISR:

CLR EA

CLR IEN0.0

CPL P1.1

MOV R2, INT\_REG1;

MOV INT\_REG1,#00h;

EXINT1\_CHK:

MOV A,R2

ANL A,#01h

JZ EXINT2\_CHK

IND\_PULSE 1

EXINT2\_CHK:

MOV A,R2

ANL A,#02h

JZ EXINT3\_CHK

IND\_PULSE 2

EXINT3\_CHK:

MOV A,R2

ANL A,#04h

JZ EXINT4\_CHK

IND\_PULSE 3

EXINT4\_CHK:

MOV A,R2

ANL A,#08h

JZ EXINT4\_EXIT

IND\_PULSE 4

EXINT4\_EXIT:

MOV A,INT\_REG1

ANL A,#0F0h

MOV INT\_REG1,A

SETB IEN0.0

SETB EA

RETI

END

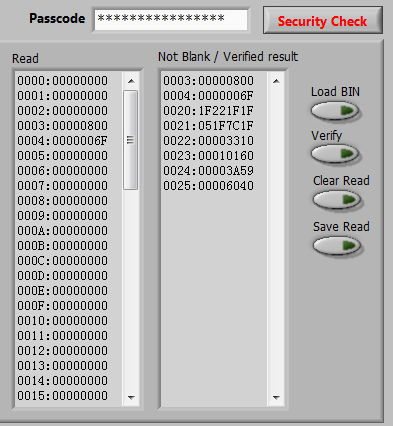
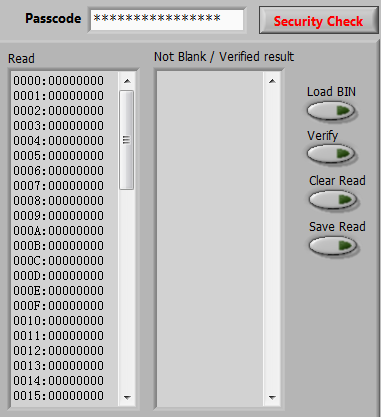
* Summary of EXINT, functional is ok, it can wake-up the MCU in STOP mode (Level trigger).

## Dongle

If password match, by SPI interface

Data can program and read current program data.

EEPROM data can write/read



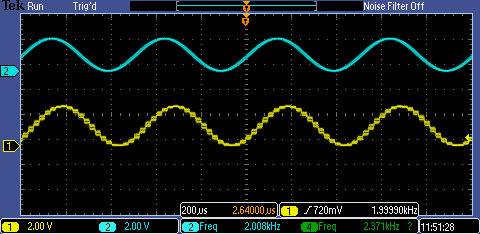
After match Dongle password EEPROM data are able to read/write

Incorrect Dongle password EEPROM data will not display

* Dongle function ok.

## ADC/ DAC Test

ADC connect a signal generator and convert to digital and write to DAC output



DAC output

ADC input

Test Program:

#define ADCin1 1

//#define ADCin2 1

//#define ADCin3 1

ORG 2000h

LJMP START

START:

MOV SP,#50H ;init stack pointer

EMOV P0DD,0FFh

EMOV AD\_PD, 00h ;Power-on ADC

EMOV 0E0ECh, 0Ah ;enable ADC clk

#ifdef ADCin1

EMOV ADCHS, 08h ;select Ch4 => ADCin1

#endif

#ifdef ADCin2

EMOV ADCHS, 10h ;select Ch5 => ADCin2

#endif

#ifdef ADCin3

EMOV ADCHS, 20h ;select Ch6 => ADCin3

#endif

;; EMOV DAC\_CON, 09h ;enable DAC1 Vref 2.5V

EMOV DAC\_CON, 00h ;enable DAC1,2 Vref 3.3V

; EMOV DAC\_CON, 0ah ;enable DAC1,2, Vref 2.5V

; EMOV DAC\_CON, 0ch ;enable DAC2, Vref 3.3V

EMOV ADCTL, 20h ;software trigger

Main\_loop:

EMOVR ADCTL

ANL A, #40h

JNZ Main\_loop ;ADC busy ?

EMOV ADCTL, 20h ;software trigger

Wait\_Conv:

EMOVR ADFLG

ANL A,#01h

JZ Wait\_Conv

#ifdef ADCin1

EMOVR AD4OUT\_L

#endif

#ifdef ADCin2

EMOVR AD5OUT\_L

#endif

#ifdef ADCin3

EMOVR AD6OUT\_L

#endif

;11bit -> 8bit

ANL A,#0F8h

RR A

RR A

RR A

MOV R0,A

#ifdef ADCin1

EMOVR AD4OUT\_H

#endif

#ifdef ADCin2

EMOVR AD5OUT\_H

#endif

#ifdef ADCin3

EMOVR AD6OUT\_H

#endif

ANL A,#07h

RL A

RL A

RL A

RL A

RL A

ORL A,R0

NOP

;8bit ADC out to DAC in

MOV DPTR, #DAC1\_DAT

MOVX @DPTR, A

MOV DPTR, #DAC2\_DAT

MOVX @DPTR, A

SJMP Main\_loop

* ADC/DAC functional ok.

## Timer

### Timer0/1

Use 32MHz crystal as clock source

MOV TMOD, #22H ;Timer0,1 in mode 2

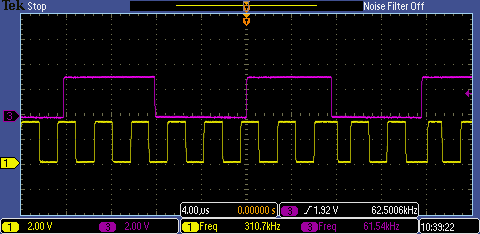
MOV TH0, #00H ;div 256

MOV TH1, #0CCH

MOV TCON, #00H

EMOV T01\_DIV\_L,00h

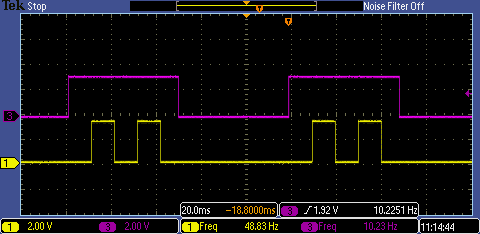
EMOV T01\_DIV\_H,00h



* T0 = 256/32M = 8us
* T1 = 52/32M = 1.6us

### Timer3

When P0.6= high, enable T3 interrupt

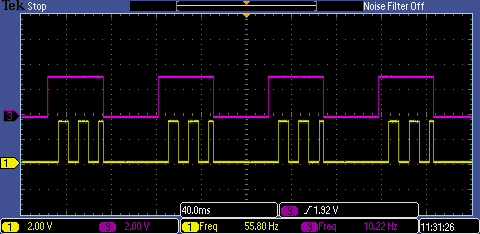


P0.7

P0.6

### Timer4

When P0.6= high, enable T4 interrupt

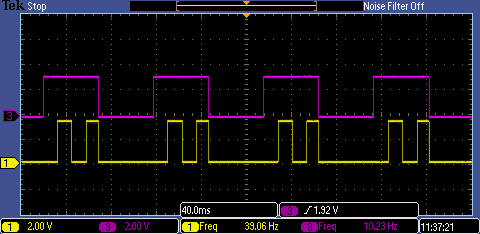


P0.7

P0.6

### Timer5

When P0.6= high, enable T5 interrupt

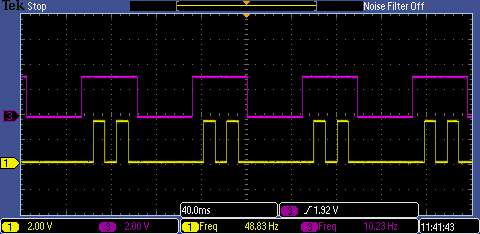


P0.7

P0.6

### Timer6

When P0.6= high, enable T6 interrupt



**Test program (Timer3)**

ORG 02000h

LJMP START

ORG PROG\_OFFSET + 013h ;3. IE3 (Comparator1, T3)

LJMP IE3\_ISR

ORG PROG\_OFFSET + 01Bh ;4. Timer1

RETI

ORG PROG\_OFFSET + 06Bh ;11. IE6 (RTC, LVD18, LVD33, T4, HWBP)

RETI

START:

MOV SP,#40H ;init stack pointer

EMOV P0\_FN\_L, 10h ;P0.2 as clock input

MOV A,CLK\_SEL\_PD

ANL A,#11101000b ;turn on 32M

ORL A,#00000101b ;Select 32M as clock source

MOV CLK\_SEL\_PD, A

EMOV P0\_FN\_H, 00h ;init (P0.7-P0.4)

EMOV P0DD, 0C0h ;(P0.7-P0.6) are output

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

EMOV CLK\_DIV\_L, 00h

EMOV CLK\_DIV\_H, 0h

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

MOV A,CLK\_SEL\_PD

ORL A,#00001000b ; 80M\_PD=1 turn off Int 40M RCOSC

MOV CLK\_SEL\_PD, A

Init\_Timer3:

CLR TCON.3

EMOV T3CON, 0A0h ;Clear Interrupt

EMOV T3PS,09h

EMOV T3RC\_L,00h

EMOV T3RC\_H,80h

SETB EX1

SETB EA

EMOV T3CTR\_L,00h

EMOV T3CTR\_H,80h

Toggle\_P0\_6\_IO:

P0\_6\_ON:

SETB P0.6

EMOV T3CON, 024h ;Enable T3, Enable Interrupt

NOP

CALL LOOP2

CALL LOOP2

CALL LOOP2

CALL LOOP2

P0\_6\_OFF:

CLR P0.6

EMOV T3CON, 00h ;Disable T3, Enable Interrupt

EMOV T3CTR\_L,00h

EMOV T3CTR\_H,80h

CLR P0.7

CALL LOOP2

CALL LOOP2

CALL LOOP2

CALL LOOP2

SJMP Toggle\_P0\_6\_IO

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

IE3\_ISR:

EMOVR T3CON

ANL A,#80h

JZ IE3\_ISR\_EXIT

CLR TCON.3 ;clear interrupt

EMOV T3CON,24h

CPL P0.7

IE3\_ISR\_EXIT:

RETI

## LVD

AVDD18 = 1.893V

DVDD33 = 3.287V

LVD33, interrupt generated when DVDD33 drop to 2.7V

LVD18, interrupt generated when 1.8V drop to 1.38V. (DVDD33 drop to 1.69V)

\*\* Enable LVD18 and LVD33, LVD18 will not generate interrupt. Only enable LVD18, interrupt for LVD18 will generate.

Test program:

ORG PROG\_OFFSET

LJMP START

ORG PROG\_OFFSET + 06Bh ;11. IE6 (RTC, LVD18, LVD33, T4, HWBP)

RETI

START:

EMOV P0\_FN\_L, 00h

EMOV P0\_FN\_H, 00h

EMOV P0DD, 0E1h

MOV P0,#00h

EMOV P0\_FN\_L, 10h ; P0.2 connect 32M clock source

MOV A,CLK\_SEL\_PD

ANL A,#11101000b ;turn on 32M

ORL A,#00000101b ;Select 32M as clock source

MOV CLK\_SEL\_PD, A

; EMOV LDO\_CON, 7Dh ;power down 1.8V

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

EMOV CLK\_DIV\_L, 20h

EMOV CLK\_DIV\_H, 0h

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

#if 1

MOV A,CLK\_SEL\_PD

ORL A,#00001000b ; 80M\_PD=1 turn off Int 40M RCOSC

MOV CLK\_SEL\_PD, A

#endif

; EMOV LVD\_CON, 33h ;Power on LVD

; EMOV LVD\_CON, 3Fh ;Power on LVD18

EMOV LVD\_CON, 0F3h ;Power on LVD33

EMOV LVD\_INT\_EN, 03h ;enable LVD33,LVD18 (LVD\_INT\_EN)

SETB IEN1.5

SETB EA

Main:

SETB P0.0

NOP

MOV P2, INT\_REG4

NOP

NOP

CLR P0.0

NOP

NOP

SJMP Main

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

IE11\_ISR:

CPL P0.5

CHK\_LVD18:

MOV A,INT\_REG4

ANL A,#02h

JNZ LVD18

SJMP CHK\_LVD33

LVD18:

SETB P0.6

NOP

NOP

NOP

CLR P0.6

CHK\_LVD33:

MOV A,INT\_REG4

ANL A,#04h

JNZ LVD33

SJMP LVD\_ISR\_EXIT

LVD33:

SETB P0.6

NOP

NOP

NOP

NOP

CLR P0.6

NOP

NOP

NOP

NOP

SETB P0.6

NOP

NOP

NOP

NOP

CLR P0.6

NOP

LVD\_ISR\_EXIT:

; CPL P0.6

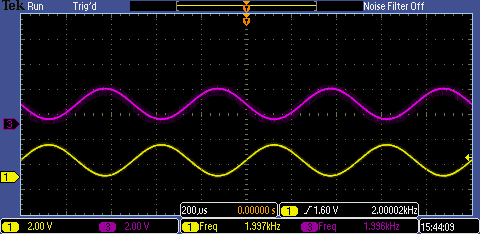
RETI

END

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

## OpAmp/Comparator

EMOV OPAMP\_CON, 07Eh ;OpAmp mode, => inverting amplifier

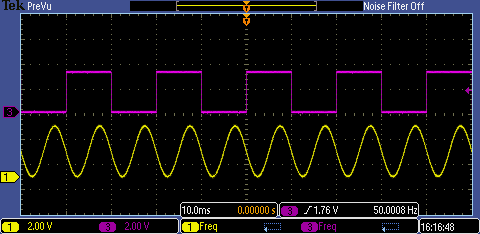


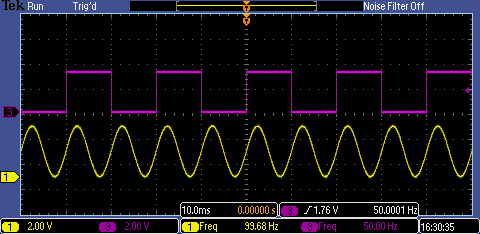
Output

INN

MOV OPAMP\_CON, 7Dh ;Comparator mode, interrupt generated,

Can be rising or falling.





**Test program:**

ORG 02000h

LJMP START

ORG PROG\_OFFSET + 013h ;3. IE1 (Comparator1, T3)

LJMP IE1\_ISR

ORG PROG\_OFFSET + 05Bh ;9. IE4 (Comparator2,3,4)

LJMP IE4\_ISR

START:

EMOV P0\_FN\_L, 00h

EMOV P0\_FN\_H, 00h

EMOV P0DD, 0FFh

Init\_Comp:

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;; Power up COMP

; EMOV OPAMP\_CON, 07Eh ;power on OPAMP1

; EMOV OPAMP\_CON, 07Bh ;power on OPAMP2

; EMOV OPAMP\_CON, 06Fh ;power on OPAMP3

EMOV OPAMP\_CON, 7Dh ;power on COMP1

; EMOV OPAMP\_CON, 77h ;power on COMP2

; EMOV OPAMP\_CON, 5Fh ;power on COMP3

; EMOV OPAMP\_CON, 6Fh ;power on COMP4

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;; Interrupt Select

EMOV COMP\_INT\_EN, 01h ;enable COMP1 interrupt

; EMOV COMP\_INT\_EN, 02h ;enable COMP2 interrupt

; EMOV COMP\_INT\_EN, 04h ;enable COMP3 interrupt

; EMOV COMP\_INT\_EN, 08h ;enable COMP4 interrupt

;;; COMP trigger type, raising or failling

; EMOV COMP\_INT\_SEL, 0AAh ;COMP failling/raising edge INT trigger

EMOV COMP\_INT\_SEL, 00h ;COMP failling edge INT trigger

; EMOV COMP\_INT\_SEL, 55h ;COMP raising edge INT trigger

SETB EX1 ;INT of T3 and COMP1

; SETB IEN1.3 ;INT of COMP2,3,4

; SETB TCON.2 ;fall edge

SETB EA

; EMOV COMP\_INT\_EN, 1Eh ;enable COMP2,3,4 interrupt

EMOV COMP\_INT\_EN, 01h ;enable COMP1 interrupt

SJMP $

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

IE1\_ISR:

EMOVR T4CON

ANL A,#80h

JZ No\_overflow

EMOV T4CON,24h

CPL P0.7

SJMP EXIT\_IE1\_ISR

No\_overflow:

EMOVR COMP\_INT

ANL A,#01h ;COMP1 interrupt occur ?

JZ EXIT\_IE1\_ISR

CPL P0.0

EMOV COMP\_INT, 00h

EXIT\_IE1\_ISR:

RETI

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

IE4\_ISR:

EMOVR COMP\_INT

MOV R1,A

COMP2\_CHK:

ANL A,#02h ;COMP2 interrupt occur ?

JZ COMP3\_CHK

SETB P0.0

NOP

NOP

CLR P0.0

NOP

NOP

COMP3\_CHK:

MOV A,R1

ANL A,#04h ;COMP3 interrupt occur ?

JZ COMP4\_CHK

SETB P0.0

NOP

NOP

CLR P0.0

NOP

NOP

SETB P0.0

NOP

NOP

CLR P0.0

NOP

NOP

COMP4\_CHK:

MOV A,R1

ANL A,#08h ;COMP4 interrupt occur ?

JZ EXIT\_IE4\_ISR

SETB P0.0

NOP

NOP

CLR P0.0

NOP

NOP

SETB P0.0

NOP

NOP

CLR P0.0

NOP

NOP

SETB P0.0

NOP

NOP

CLR P0.0

NOP

NOP

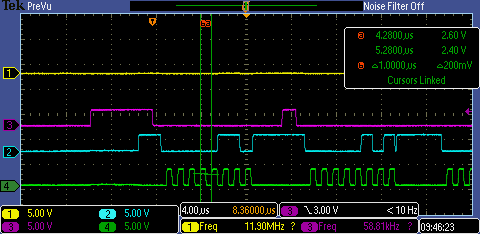
EXIT\_IE4\_ISR:

EMOV COMP\_INT, 00h

RETI

END

## SPI

**Use on chip SPI to module to program external flash**

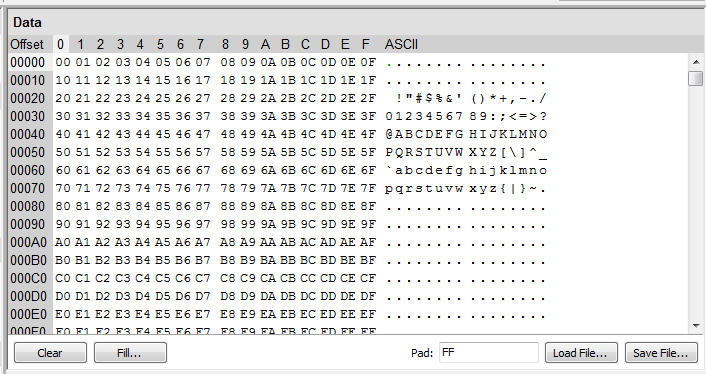
MISO

CSB

MOSI

SCK

Use other programmer to read SPI flash, data was programmed.



J27 J28

* MISO and MOSI in EVB requre to cross connection

**Test program,**

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

#define BULK\_ERASE 1

#define READ\_ID 1

#define PAGE\_PROGRAM 1

PROG\_OFFSET EQU 02000h

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; Flash Instruction

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

WREN EQU 06h ;Write enable

WRDI EQU 04h ;Write disable

RDID EQU 09Fh ;Read ID

RDSR EQU 05h ;Read status register

WRSR EQU 01h ;Write status register

READ EQU 03h ;Read data bytes

FAST\_READ EQU 0Bh ;Read data bytes at higher speed

PP EQU 02h ;Page program

SE EQU 0DBh ;Sector erase

BE EQU 0C7h ;Bulk erase

DP EQU 0B9h ;Deep power-down

RES EQU 0ABh ;Release from deep power-down

DUMMY EQU 00h ;Dummy

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; MACRO Definition

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

NOPX MACRO NUM

LOCAL NOP\_LOOP

MOV R4,#NUM

NOP\_LOOP:

NOP

DJNZ R4,NOP\_LOOP

ENDM

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

CSB\_CLR MACRO

NOP

NOP

SETB P1.4

ENDM

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

CSB\_SET MACRO

NOP

NOP

CLR P1.4

ENDM

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; Reset and Interrupt Vector

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

ORG PROG\_OFFSET

LJMP START

ORG PROG\_OFFSET + 063h ;10. IE5

LJMP SPI\_ISR

; RETI

START:

EMOV P0\_FN\_L, 10h ;select external 32M clock source

MOV A,CLK\_SEL\_PD

ANL A,#11101000b ;turn on 32M

ORL A,#00000101b ;Select 32M as clock source

MOV CLK\_SEL\_PD, A

EMOV P0\_FN\_H, 00h

EMOV P0DD, 0FBh

EMOV P1\_FN\_L, 50h ;P1.3-SPI\_MOSI, P1.2-SPI\_MISO,

EMOV P1\_FN\_H, 04h ;P1.7-UTX1, P1.6-RX1, P1.5-SPI\_SCK, P1.4-SPI\_CSB

EMOV P1DD,10h

SETB P1.4

EMOV P2\_FN\_L, 00h ;Set P2.0-7 output

EMOV P2\_FN\_H, 00h

EMOV P2DD, 0FFh

MOV IEN0,#80h

; MOV IEN1,#10h

EMOV SPCON, 0C2h ;1Mhz sck, master

SPI\_START:

#if BULK\_ERASE

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;Bulk Erase

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

MOV A,#WREN ;Write enable

CALL SPI\_MS\_TX

CSB\_CLR

MOV A,#RDSR ;Check Status

CALL SPI\_MS\_TX

MOV A,#DUMMY

CALL SPI\_MS\_TX

BE\_STATUS\_CHK1:

MOV A,#DUMMY

CALL SPI\_MS\_TX

MOV A,B

ANL A,#02h

JZ BE\_STATUS\_CHK1 ;Write enable ?

CSB\_CLR

MOV A,#BE ;Bulk Erase

CALL SPI\_MS\_TX

CSB\_CLR

MOV A,#RDSR ;Write enable

CALL SPI\_MS\_TX

MOV A,#DUMMY

CALL SPI\_MS\_TX

BE\_STATUS\_CHK2:

MOV A,#DUMMY

CALL SPI\_MS\_TX

MOV A,B

JNZ BE\_STATUS\_CHK2 ;Bulk erase finished ?

CSB\_CLR

#endif

#if READ\_ID

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;Read ID

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

MOV A,#RDID

CALL SPI\_MS\_TX

MOV A,#DUMMY

CALL SPI\_MS\_TX

MOV A,#DUMMY

CALL SPI\_MS\_TX

MOV A,#DUMMY

CALL SPI\_MS\_TX

MOV A,#DUMMY

CALL SPI\_MS\_TX

CSB\_CLR

#endif

#if PAGE\_PROGRAM

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;Page Program

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

MOV A,#WREN ;Write enable

CALL SPI\_MS\_TX

CSB\_CLR

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;Read Status

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

MOV A,#RDSR ;Check Status

CALL SPI\_MS\_TX

MOV A,#DUMMY

CALL SPI\_MS\_TX

PP\_STATUS\_CHK1:

MOV A,#DUMMY

CALL SPI\_MS\_TX

MOV A,B

ANL A,#02h

JZ PP\_STATUS\_CHK1 ;Write enable ?

CSB\_CLR

MOV R0,#00h

MOV R1,#00h

MOV A,#PP ;page program

CALL SPI\_MS\_TX

MOV A,#00h

CALL SPI\_MS\_TX

MOV A,#00h

CALL SPI\_MS\_TX

MOV A,#00h

CALL SPI\_MS\_TX

Data\_INC:

MOV A,R1

CALL SPI\_MS\_TX

INC R1

DJNZ R0,Data\_INC

CSB\_CLR

MOV A,#WRDI ;Write disable

CALL SPI\_MS\_TX

CSB\_CLR

#endif

SJMP $

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;SPI\_MS\_TX sub-routine

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

SPI\_MS\_TX:

CSB\_SET ;CS active

MOV DPTR,#SPDAT

MOVX @DPTR, A ;SPI Tx

MOVX A,@DPTR ;Read data shift in

MOV B,A

MOV DPTR,#SPSTA

SPI\_MS\_SPIF\_CHK:

MOVX A,@DPTR ;Read Status register

MOV P2,A ;Move SPSTA value to P2

ANL A,#80h

JZ SPI\_MS\_SPIF\_CHK ;wait until SPIF set

RET

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

SPI\_ISR:

CPL P0.7

NOP

NOP

NOP

NOP

NOP

CPL P0.7

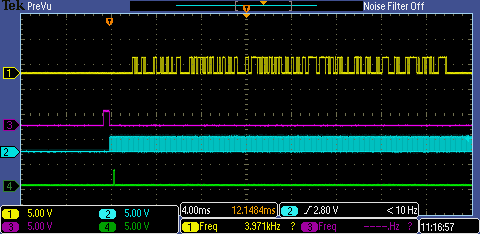
RETI

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

END

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

## E-Flash mode (Boot from external flash)



MISO

CSB

SCK

MOSI

J27 J28

* P2[7:4] reserve for SPI clock setting
* P1[5:2] SPI interface (SCLK, CSB, SDO, SDI)
* MISO and MOSI in EVB requre to cross connection

Data format: 32-bit ie, (byte3, byte2, byte1, byte0, byte7, byte6, byte5, byte5,…….)

Test program:

ORG PROG\_OFFSET

LJMP START

START:

MOV SP,#50H ;init stack pointer

EMOV P0\_FN\_L, 00h ;init (P0.3-P0.0)

EMOV P0\_FN\_H, 00h ;init (P0.7-P0.4)

EMOV P0DD,0FFH

EMOV P1\_FN\_L, 00h ;init (P0.3-P0.0)

EMOV P1\_FN\_H, 00h ;init (P0.7-P0.4)

EMOV P1DD,0FFH

EMOV P2\_FN\_L, 00h ;init (P0.3-P0.0)

EMOV P2\_FN\_H, 00h ;init (P0.7-P0.4)

EMOV P2DD,0FFH

CONTINUE:

MOV P0,#00h

MOV P1,#00h

MOV P2,#00h

CALL LOOP3

MOV P0,#0FFh

MOV P1,#0FFh

MOV P2,#0FFh

CALL LOOP3

SJMP CONTINUE

Compile source code to generate hex file, must 32-bit format in E-Flash mode.

**(Original 8-bit format)**

02 20 03 75 81 50 90 E0 D5 74 00 F0 90 E0 D4 74

00 F0 90 E0 D6 74 FF F0 90 E0 DB 74 00 F0 90 E0

DA 74 00 F0 90 E0 DC 74 FF F0 90 E0 E1 74 00 F0

90 E0 E0 74 00 F0 90 E0 E2 74 FF F0 75 80 00 75

90 00 75 A0 00 12 20 56 75 80 FF 75 90 FF 75 A0

FF 12 20 56 80 E6 7A 0A 79 FF 78 FF 00 00 00 00

D8 FA D9 F6 DA F2 22

**Test Program : (32-bit format)**

75 03 20 02 E0 90 50 81 F0 00 74 D5 74 D4 E0 90

E0 90 F0 00 F0 FF 74 D6 74 DB E0 90 E0 90 F0 00

F0 00 74 DA 74 DC E0 90 E0 90 F0 FF F0 00 74 E1

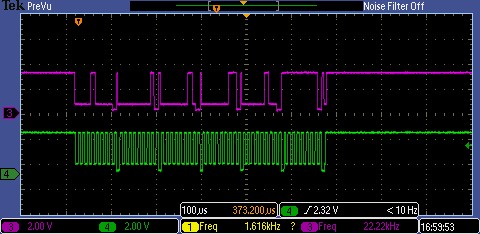
74 E0 E0 90 E0 90 F0 00 F0 FF 74 E2 75 00 80 75

A0 75 00 90 56 20 12 00 75 FF 80 75 A0 75 FF 90

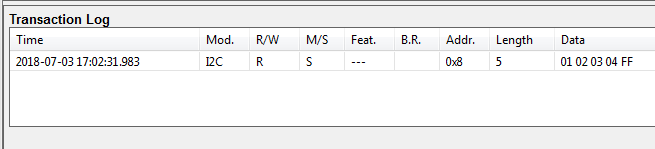
56 20 12 FF 0A 7A E6 80 FF 78 FF 79 00 00 00 00

F6 D9 FA D8 00 22 F2 DA

## I2C



MCU as master send 0x01, 0x02, 0x03, 0x04, 0xFF

****

Use Total phase, AARDVARK I2C/SPI adaptor can receive those data

Test routine:

MOV A,#08h ;address

RL A

MOV DPTR,#I2CDATA ;Send data

MOVX @DPTR,A

EMOV I2CCON, 60h ;I2CCON, enable I2C Master mode

Check\_bit I2CCON, Bit\_SI ;Check SI bit, Send completed

EMOV I2CCON, 45h

I2C\_TX 01h ;1st Byte Data

I2C\_TX 02h ;2nd Byte Data

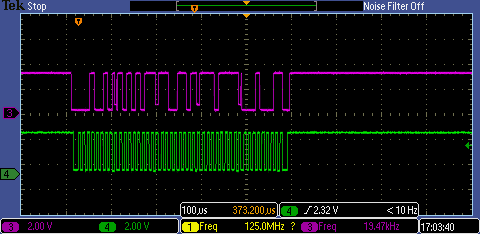
I2C\_TX 03h ;3rd Byte Data

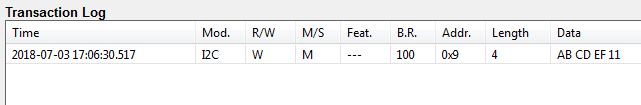
I2C\_TX 04h ;4th Byte Data

I2C\_TX\_LAST 0FFh ;Last Byte

MCU as slave, I2C adaptor send 0xAB, 0xCD, 0xEF, 0x11 to MCU,

MCU acknowledge the adaptor as address match (address 0x09)



****

**Test program:**

#define Clear\_ACK 11111110b

#define Clear\_SI 11110111b

#define Bit\_ACK 00000001b

#define Bit\_SI 00001000b

;=========================================================================

; Select Master TX or Master RX or Slave (include TX and RX)

;=========================================================================

;#define Master\_TX 1

;#define Master\_RX 1

#define Slave 1

;=========================================================================

#ifdef Master\_TX

#define Master 1

#endif

#ifdef Master\_RX

#define Master 1

#endif

;=========================================================================

ORG 2000h

START:

EMOV P0\_FN\_L, 10h ;select external 32M clock source => P0.2 as clock input

MOV A,CLK\_SEL\_PD

ANL A,#11101000b ;turn on 32M

ORL A,#00000101b ;Select 32M as clock source

MOV CLK\_SEL\_PD, A

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

EMOV CLK\_DIV\_L, 4h

EMOV CLK\_DIV\_H, 0h

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

EMOV P1\_FN\_L, 05h ;init I2C port mux

I2C\_init:

EMOV P2\_FN\_L, 00h ;(P2\_FN\_L): init (P2.3-P2.0)

EMOV P2\_FN\_H, 00h ;(P2\_FN\_H): init (P2.7-P2.4)

EMOV P2DD, 0FFh ;(P2DD): P2 all output

MOV P2,#0FFh

EMOV I2CCLK, 13h ;clock 100K

; EMOV I2CCLK, 09h ;clock 200K

; EMOV I2CCLK, 04h ;clock 400K

;=========================================================================

; Master

;=========================================================================

#ifdef Master

;=========================================================================

Start\_master:

EMOV I2CADDR, 12h ;device address (address=0x09)

;============= Write to Slave ===============================

#ifdef Master\_TX

MOV A,#08h

RL A

MOV DPTR,#I2CDATA ;Send data

MOVX @DPTR,A

EMOV I2CCON, 60h ;I2CCON, enable I2C Master mode

Check\_bit I2CCON, Bit\_SI ;Check SI bit, Send completed

EMOV I2CCON, 45h

I2C\_TX 01h ;1st Byte Data

I2C\_TX 02h ;2nd Byte Data

I2C\_TX 03h ;3rd Byte Data

I2C\_TX 04h ;4th Byte Data

; Check\_bit I2CCON, Bit\_SI ;Check SI bit, Send completed

; EMOV I2CCON, 50h

I2C\_TX\_LAST 0FFh ;Last Byte

#endif

;============= Read from Slave ===============================

#ifdef Master\_RX

MOV A,#08h

RL A

ORL A,#01h ;Read from slave

MOV DPTR,#I2CDATA ;Send data

MOVX @DPTR,A

EMOV I2CCON, 60h ;I2CCON, enable I2C Master mode

Check\_bit I2CCON, Bit\_SI ;Check SI bit, Send completed

EMOV I2CCON, 45h

I2C\_RX ;Rx 1st byte

I2C\_RX ;Rx 2nd byte

I2C\_RX ;Rx 3rd byte

I2C\_RX ;Rx 4th byte

I2C\_RX ;Rx 5th byte

I2C\_RX ;Rx 6th byte

EMOV I2CDATA,00h ;End dummy byte

Check\_bit I2CCON, Bit\_SI ;Check SI bit, Send completed

EMOV I2CCON, 50h ;End master read

#endif

;==================================================================

SJMP $

#endif

;=========================================================================

; Slave

;=========================================================================

#ifdef Slave

#define Slave\_TX 1

#define Slave\_RX 1

;=========================================================================

Start\_slave:

MOV R7,#0Fh

CLEAR\_BIT I2CCON, Clear\_SI

EMOV I2CSTATUS,00h

EMOV I2CADDR, 12h ;device address (address=0x09)

EMOV I2CCON, 44h

Check\_bit I2CCON, Bit\_SI ;Data receive ?

Check\_bit I2CSTATUS, 00000010b ;Check address match or not?

EMOV I2CSTATUS,00h

Check\_Data:

EMOVR I2CTIMER ;check direction rx/tx

ANL A,#08h

JZ \_Slave\_RX

JMP \_Slave\_TX

;==================== Slave Recive (Master Send) ==============================

#ifdef Slave\_RX

\_Slave\_RX:

I2C\_SRX ;1st receive byte

I2C\_SRX ;2nd receive byte

I2C\_SRX ;3rd receive byte

I2C\_SRX ;4th receive byte

EMOV I2CCON, 40h ;No ACK

Check\_bit I2CCON, Bit\_SI ;Check SI bit

#endif

;==================== Slave Send (Master Receive) ==============================

#ifdef Slave\_TX

\_Slave\_TX:

EMOV I2CDATA,00h ;1st byte send

EMOV I2CCON, 44h

Check\_bit I2CCON, Bit\_SI ;Completed ?

Check\_bit I2CCON, Bit\_ACK ;Completed ?

Check\_REG I2CSTATUS,P2

EMOV I2CSTATUS,00h

I2C\_STX 01h

I2C\_STX 23h

I2C\_STX 45h

I2C\_STX 67h

I2C\_STX\_LAST 0AAh

#endif

;================================================================================

SJMP $

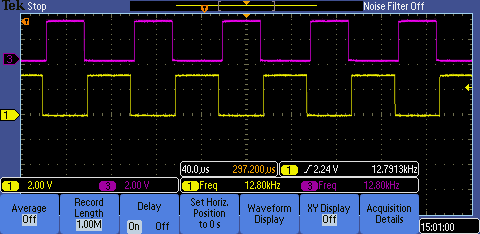
#endif

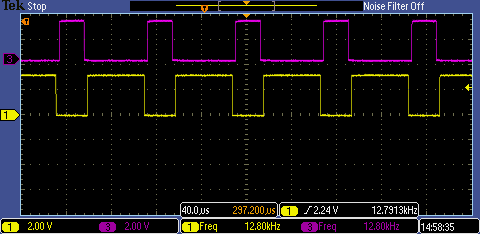
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

END

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

## PWM





PWM1A (Yellow) PWM1B(Cyan)

Test Program

DEAD\_TIME\_L EQU 0x64;

DEAD\_TIME\_H EQU 0x00;

PRD\_TIME\_L EQU 0xe2;

PRD\_TIME\_H EQU 0x04;

;-------------------------------------------------------------------------------

ORG 02000h

LJMP START

START:

EMOV CLK\_DIV\_L, 00h

EMOV CLK\_DIV\_H, 00h

; system config

EMOV P0\_FN\_L,00h

EMOV P0\_FN\_H,00h

EMOV P2\_FN\_H,55h

EMOV P2\_FN\_L,50h

EMOV P0DD, 0FFh

EMOV PCLKCR, 1Eh

; Set counter mode = up/down

EMOV P1TBCTL\_L, 02h

; Set CLK ratio = 1, TB counter = run

EMOV P1TBCTL\_H, 80h

; Set period = 1250

mov P1TBPRD\_L, #PRD\_TIME\_L ;P1 period 1250

mov P1TBPRD\_H, #PRD\_TIME\_H

;CMP load from CTR = Zero or CTR = PRD

EMOV P1CMPCTL, 0Ah

; Disable PWM1

EMOV P1AQCTLA\_L, 50h

EMOV P1AQCTLB\_L, 0A0h

;DB control

EMOV P1DBCTL, 2Bh

; Rising Edge Delay Count

EMOV P1DBRED\_L, DEAD\_TIME\_L ;write

EMOV P1DBRED\_H, DEAD\_TIME\_H ;write

; Falling Edge Delay Count

EMOV P1DBFED\_L, DEAD\_TIME\_L ;write

EMOV P1DBFED\_H, DEAD\_TIME\_H ;write

;TZ control, enable OSHT

EMOV P1TZSEL, 01h

;TZ force PWMxA & PWMxB to a low state

EMOV P1TZCTL, 0Ah

;Enable TZ INT

EMOV P1TZEINT, 04h

;Enable ET INT

EMOV P1ETSEL\_L, 09h

;Enable ET SOCA & SOCB

EMOV P1ETSEL\_H, 0AFh

; Generate INT on the 3 event

EMOV P1ETPS\_L, 03h

; Generate SOCA & SOCB on the 3 event

EMOV P1ETPS\_H, 33h

;ENABLE Epwm 1-4, Start all the TB clocks

EMOV PCLKCR, 1Fh

; Set CMPA

mov P1CMPA\_L, #0x0f4 ;write

mov P1CMPA\_H, #0x001 ;write

mov P1CMPB\_L, #0x001 ;write

mov P1CMPB\_H, #0x000 ;write

mov a, #0x0ff;

; Disable PWM

mov dptr, #P1AQCTLA\_L

mov a, #0x050

movx @dptr, a

mov dptr, #P1AQCTLB\_L

mov a, #0x0a0

movx @dptr, a

; Enable PWM

mov dptr, #P1AQCTLA\_L

mov a, #0x090

movx @dptr, a

mov dptr, #P1AQCTLB\_L

mov a, #0x090

movx @dptr, a

mov P1ETCLR, #0x001

CALL LOOP0

; Set CMPA

mov p1, #0x002

mov P1CMPA\_L, #0x000 ;write 1-255

mov P1CMPA\_H, #0x000 ;write

mov P1CMPB\_L, #0x000 ;write

mov P1CMPB\_H, #0x000 ;write

dly3:

mov r0, #0x000;

mov r1, #0x000;

dly0:

mov a, P1ETFLG

cjne a, #0x00d, dly0

mov P1ETCLR, #0x001

mov dptr, #0x00e05b

movx a, @dptr

mov r0, P1CMPA\_L

inc r0

mov P1CMPA\_L, r0 ;write 1-255

mov P1CMPA\_H, r1 ;write

mov P1CMPB\_L, r0 ;write

mov P1CMPB\_H, r1 ;write

cjne r0, #0x0ff, dly0

inc r1

cjne r1, #0x004, dly0

dly1:

mov a, P1ETFLG

cjne a, #0x00d, dly1

mov P1ETCLR, #0x001

mov dptr, #0x00e05b

movx a, @dptr

mov r0, P1CMPA\_L

inc r0

mov P1CMPA\_L, r0 ;write

mov P1CMPA\_H, r1 ;write

mov P1CMPB\_L, r0 ;write

mov P1CMPB\_H, r1 ;write

cjne r0, #0x0e2, dly1

inc r1

cjne r1, #0x004, dly3

SJMP $

## ROM table

(first 96 bytes only)

00 00 00 C9 01 92 02 5B 03 23 03 EC 04 B5 05 7D

06 45 07 0D 07 D5 08 9C 09 64 0A 2A 0A F1 0B B6

0C 7C 0D 41 0E 05 0E C9 0F 8C 10 4F 11 11 11 D3

12 93 13 54 14 13 14 D1 15 8F 16 4C 17 08 17 C3

18 7D 19 37 19 EF 1A A6 1B 5D 1C 12 1C C6 1D 79

1E 2B 1E DC 1F 8B 20 39 20 E7 21 92 22 3D 22 E6

# Appendix:

## MACRO & Sub-routine

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

EMOVA MACRO Addr

MOV DPTR,#Addr

MOVX @DPTR, A

ENDM

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

EMOV MACRO Addr, Data

MOV DPTR,#Addr

MOV A,#Data

MOVX @DPTR, A

ENDM

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

EMOVR MACRO Addr

MOV DPTR,#Addr

MOVX A,@DPTR

ENDM

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

IND\_PULSE MACRO Number

LOCAL IND, EXIT

MOV R3,#Number

IND:

SETB P1.0

NOP

NOP

CLR P1.0

DJNZ R3,IND

ENDM

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

LOOP3: MOV R2, #0AH

LOOP2: MOV R1, #0FFH

LOOP1: MOV R0, #0FFH

LOOP0: NOP

NOP

NOP

NOP

DJNZ R0,LOOP0

DJNZ R1,LOOP1

DJNZ R2,LOOP2

RET

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;