University of British Columbia
Electrical and Computer Engineering
ELEC291/ELEC292

Project 1 – EFM8 board, FSM, NVMEM, and tips

Dr. Jesús Calviño-Fraga P.Eng.
Department of Electrical and Computer Engineering, UBC
Office: KAIS 3024
E-mail: jesusc@ece.ubc.ca
Phone: (604)-827-5387

February 9, 2024

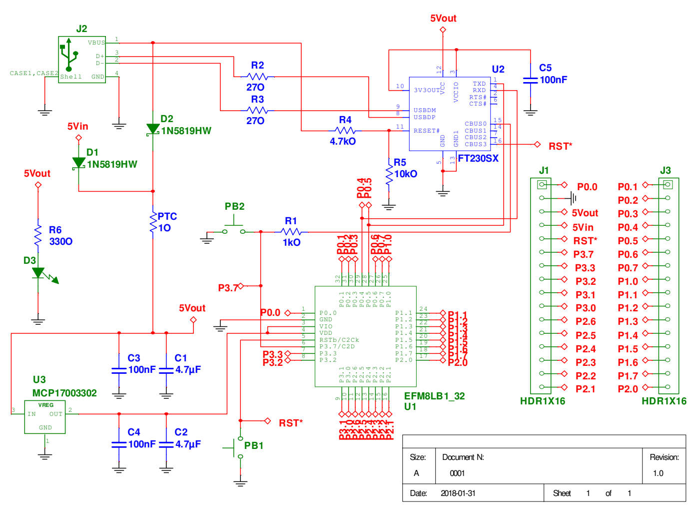
Objectives

- Introduction to the EFM8 board.
- Programming using Finite State Machines (FSMs) in assembly language.
- Using non-volatile memory (flash) for variable storage and initialization.
- Extra project tips.

The EFM8 Board

- Each student should have a EFM8 board for the second half of the course.
- Each student should assemble (or try to) a EFM8 board. Stencil + Solder Paste + SMDs + TH + Testing.
- The EFM8 board needs to be soldered in an reflow oven. You need a reflow oven controller!

EFM8 circuit



EFM8 Bill of Materials (BOM)

Qty	Supplier's#	Reference	Man's #	Description
1	768-1135-1-ND	U2	FT230XS-R	IC USB SERIAL BASIC UART 16SSOP
1	MCP1700T3302ETTCT-ND	U3	MCP1700T-3302E/TT	IC REG LDO 3.3V 0.25A SOT23-3
1	336-3736-ND	U1	EFM8LB12F64E-B-QFP32	IC MCU 8BIT 64KB FLASH 32QFP
2	450-1759-1-ND	PB1, PB2	FSM4JSMATR	SWITCH TACTILE SPST-NO 0.05A 24V
2	A26509-16-ND	J1, J3	4-103741-0-16	CONN HEADR BRKWAY .100 16POS STR
1	ED2983-ND	J2	USB-B1HSB6	CONN USB TYPE B R/A BLACK
2	1N5819HW-FDICT-ND	D1, D2	1N5819HW-7-F	DIODE SCHOTTKY 40V 1A SOD123
3	399-1170-1-ND	C3, C4, C5	C0805C104K5RACTU	CAP CER 0.1UF 50V X7R 0805
2	311-22ARCT-ND	R2, R3	RC0805JR-0722RL	RES SMD 22 OHM 5% 1/8W 0805
1	160-1179-1-ND	D3	LTST-C170GKT	LED GREEN CLEAR 0805 SMD
1	311-330ARCT-ND	R6	RC0805JR-07330RL	RES SMD 330 OHM 5% 1/8W 0805
1	311-1.0KARCT-ND	R1	RC0805JR-071KL	RES SMD 1K OHM 5% 1/8W 0805
1	311-4.7KARCT-ND	R4	RC0805JR-074K7L	RES SMD 4.7K OHM 5% 1/8W 0805
2	478-8125-1-ND	C1, C2	F921A475MPA	CAP TANT 4.7UF 10V 20% 0805
1	507-1797-1-ND	PTC	0ZCJ0020FF2E	PTC RESTTBLE 0.20A 30V CHIP 1206
1	311-10KARCT-ND	R5	RC0805JR-0710KL	RES SMD 10K OHM 5% 1/8W 0805

Steps Assembling a PCB with SMDs.

- Step 1: Apply solder paste to the PCB. You will use a Mylar stencil. (The most critical step in the whole process!)
- Step 2: Place the SMT components into the PCB.
- Step 3: Reflow soldering. You will be using a toaster oven with a controller of your own design.
- Step 4: Hand soldering of TH (thru hole) components.

Testing the EFM8 Board

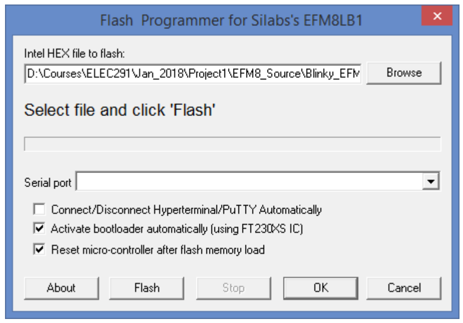
- Write a "blinky.asm" for the EFM8. Some things to take into account compared to the N76E003:
  - The default oscillator frequency is 6.000MHz. It can be configured for 12MHz, 24MHz, 48MHZ, and 72MHz… or many different values in between!
  - The number cycles per instruction is different.
  - The registers used to configure the ports are different. Check the datasheet!

blinky_EFM8.asm

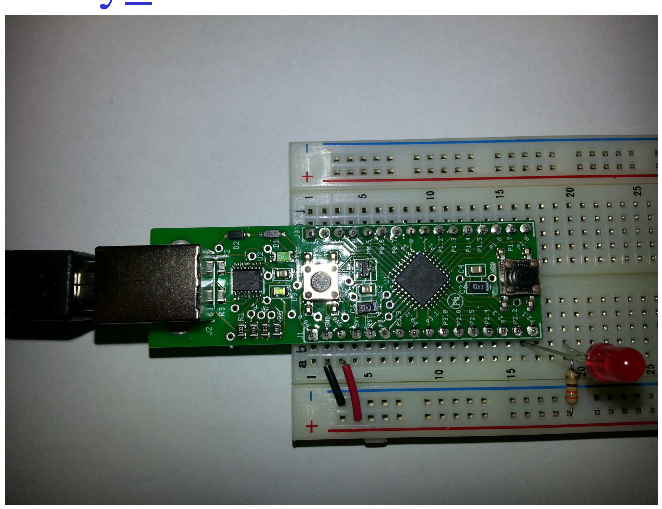
```
$MODEFM8LB1

CSEG at 0H
    ljmp main
Wait_half_second:
    ;For a 6MHz clock one machine cycle takes 1/6.0000MHz=166.666ns
    mov R2, #25
L3: mov R1, #250
L2: mov R0, #120
L1: djnz R0, L1 ; 4 machine cycles-> 4*166.666ns*120=80us
    djnz R1, L2 ; 80us*250=0.02s
    djnz R2, L3 ; 0.02s*25=0.5s
    ret
main:
    ; DISABLE WDT: provide Watchdog disable keys
    mov WDTCN, #0xDE ; First key
    mov WDTCN, #0xAD ; Second key
    mov SP, #7FH
    ; Enable crossbar and weak pull-ups
    mov XBR0, #0x00
    mov XBR1, #0x00
    mov XBR2, #0x40
    mov P2MDOUT, #0x02 ; make LED pin (P2.1) output push-pull
M0: cpl P2.1 ; Led off/on
    lcall Wait_half_second
    sjmp M0
end
```

Flashing HEX file into EFM8 Board

- In CrossIDE click fLash->Silabs EFM8LB1. Select the correct HEX file, make sure settings are like shown below, and then click ‘Flash’.



Testing the board with blinky_EFM8.asm in breadboard.

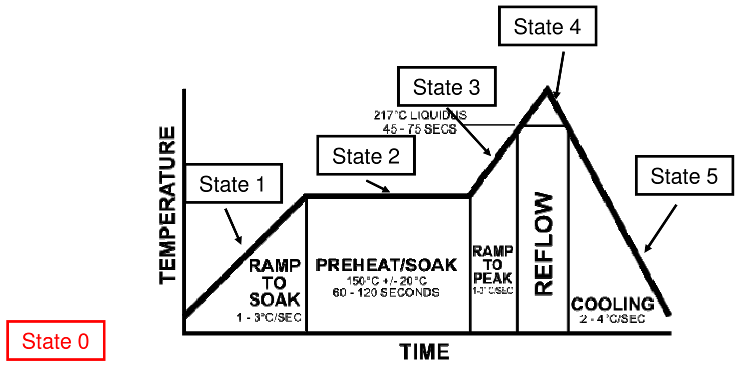


Finite State Machines in Assembly Language

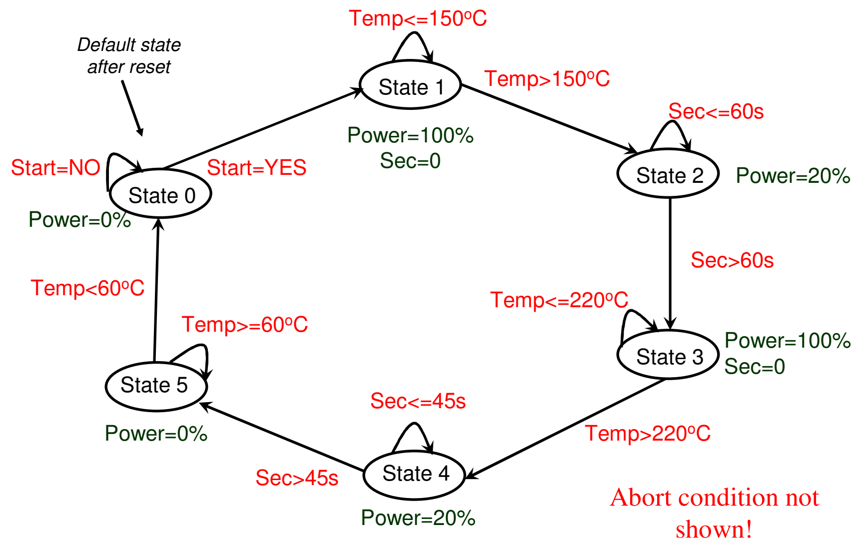
- A finite state machine (FSM) is a programming abstraction method that can be represented using a graph structure.
- We can draw the states as circles and the transitions as arrows.
- There is a finite number of states. The active state is called the current state.
- FSMs are easily implemented in assembly language!
- Many FMS can be run “concurrently”. (One after another really!)
- FSM are in principle non-blocking.

Reflow Profile States

http://en.wikipedia.org/wiki/Reflow_soldering



Reflow Profile FSM



Abort condition not shown!

FSM In assembly (some states only!)

```
FSM1:
    mov a, FSM1_state

FSM1_state0:
    cjne a, #0, FSM1_state1
    mov pwm, #0
    jb PB6, FSM1_state0_done
    jnb PB6, $ ; Wait for key release
    mov FSM1_state, #1
FSM1_state0_done:
    ljmp FSM1_FSM2

FSM1_state1:
    cjne a, #1, FSM1_state2
    mov pwm, #100
    mov sec, #0
    mov a, #150
    clr c
    subb a, temp
    jnc FSM1_state1_done
    mov FSM1_state, #2
FSM1_state1_done:
    ljmp FSM2
```

```
FSM1_state2:
    cjne a, #2, FSM1_state3
    mov pwm, #20
    mov a, #60
    clr c
    subb a, sec
    jnc FSM1_state2_done
    mov FSM1_state, #3
FSM1_state2_done:
    ljmp FSM2
         .
         .
         .
         .
         .
         .
         .
```

In assembly (some states only!) using variables…

```
FSM1:
    mov a, FSM1_state

FSM1_state0:
    cjne a, #0, FSM1_state1
    mov pwm, #0
    jb PB6, FSM1_state0_done
    jnb PB6, $ ; Wait for key release
    mov FSM1_state, #1
FSM1_state0_done:
    ljmp FSM2

FSM1_state1:
    cjne a, #1, FSM2_state2
    mov pwm, #100
    mov sec, #0
    mov a, temp_soak
    clr c
    subb a, temp
    jnc FSM1_state1_done
    mov FSM1_state, #2
FSM1_state1_done:
    ljmp FSM2
```

```
FSM1_state2:
    cjne a, #2, FSM1_state3
    mov pwm, #20
    mov a, time_soak
    clr c
    subb a, sec
    jnc FSM1_state2_done
    mov FSM1_state, #3
FSM1_state2_done:
    ljmp FSM2
        .
        .
        .
        .
        .
DSEG ; Before the state machine!
state: ds 1
temp_soak: ds 1
Time_soak: ds 1
Temp_refl: ds 1
Time_refl: ds 1
```

Variables need to be initialized!

About Variables

- Initialize variables before using them!
- It is easy to work with binary (8-bit) variables. Use “inc”, “dec”, to increment/decrement and ‘subb’ to compare.
- Small variables are easy to save and retrieve from non-volatile memory such as FLASH or EEPROM.
- If temperature measurements are too “noisy”, make several measurements and take the average!
- To convert 8-bit binary variable to decimal use either HEX2BCD (in the math32 library) or one of these smaller/faster 8051 subroutines:

Binary to decimal conversion of 8-bit numbers in the 8051

```
; Send eight bit number via serial port, passed in 'a'.
SendToSerialPort:
        mov b, #100
        div ab
        orl a, #0x30 ; Convert hundreds to ASCII
        lcall putchar ; Send to PuTTY/Python/Matlab
        mov a, b     ; Remainder is in register b
        mov b, #10
        div ab
        orl a, #0x30 ; Convert tens to ASCII
        lcall putchar ; Send to PuTTY/Python/Matlab
        mov a, b
        orl a, #0x30 ; Convert units to ASCII
        lcall putchar ; Send to PuTTY/Python/Matlab
        ret
```


Binary to decimal conversion of 8-bit numbers in the 8051

```
; Eight bit number to display passed in 'a'.
; Sends result to LCD
SendToLCD:
        mov b, #100
        div ab
        orl a, #0x30 ; Convert hundreds to ASCII
        lcall ?WriteData ; Send to LCD
        mov a, b     ; Remainder is in register b
        mov b, #10
        div ab
        orl a, #0x30 ; Convert tens to ASCII
        lcall ?WriteData; Send to LCD
        mov a, b
        orl a, #0x30 ; Convert units to ASCII
        lcall ?WriteData; Send to LCD
        ret
```

DIV AB

DIV AB

Function: Divide

Description: DIV AB divides the unsigned eight-bit integer in the accumulator by the unsigned eight-bit integer in register B. The accumulator receives the integer part of the quotient; register B receives the integer remainder. The carry and OV flags will be cleared. *Exception*: If B had originally contained 00H, the values returned in the accumulator and B register will be undefined and the overflow flag will be set. The carry flag is cleared in any case.

Example: The accumulator contains 251 (0FBH or 11111011B) and B contains 18 (12H or 00010010B). The instruction DIV AB will leave 13 in the accumulator (0DH or 00001101 B) and the value 17 (11H or 00010001B) in B, since 251 = (13x18) + 17. Carry and OV will both be cleared.

Operation: **DIV AB**
(A), (B) ¬ (A) / (B)

Encoding:

1	0	0	0	0	1	0	0

Non-Volatile Memory: N76E003 Flash Memory as Data Storage

21.3 Using Flash Memory as Data Storage

In general application, there is a need of data storage, which is non-volatile so that it remains its content even after the power is off. Therefore, in general application user can read back or update the data, which rules as parameters or constants for system control. The Flash Memory array of the N76E003 supports IAP function and any byte in the Flash Memory array may be read using the MOVC instruction and thus is suitable for use as non-volatile data storage. IAP provides erase and program function that makes it easy for one or more bytes within a page to be erased and programmed in a routine. IAP performs in the application under the control of the microcontroller's firmware. Be aware of Flash Memory writing endurance of 100,000 cycles. A demo is illustrated as follows.

Assembly demo code:

```
;*********************************************************************************
;      This code illustrates how to use IAP to make APROM 201h as a byte of
```

Aug. 03, 2020 Page **222** of **274** Rev. 1.09

Why non-volatile memory?

- To save your reflow oven controller parameters so they are available automatically the next time you use it.
- To store other useful information.

Example: Writing Project Data to the N76E003 Flash Memory

```
;*****************************************************************************
;       This code illustrates how to use IAP to make APROM 201h as a byte of
```

Aug. 03, 2020 Page 222 of 274 Rev. 1.09

nuvoTon N76E003 Datasheet

```
;       Data Flash when user code is executed in APROM.
;*****************************************************************************
PAGE_ERASE_AP               EQU         00100010b
BYTE_PROGRAM_AP             EQU         00100001b

        ORG     0000h

        MOV     TA,#0Aah                ;CHPCON is TA protected
        MOV     TA,#55h
        ORL     CHPCON,#00000001b       ;IAPEN = 1, enable IAP mode

        MOV     TA,#0Aah                ;IAPUEN is TA protected
        MOV     TA,#55h
        ORL     IAPUEN,#00000001b       ;APUEN = 1, enable APROM update

        MOV     IAPCN,#PAGE_ERASE_AP    ;Erase page 200h~27Fh
```

Don't use memory location 201H. Your program is likely to be there and you'll corrupt it! Check my example posted on Canvas.

Extra Tips…

- Are you using macros yet?

```
Change_8bit_Variable MAC
    jb %0, %2
    Wait_Milli_Seconds(#50) ; de-bounce
    jb %0, %2
    jnb %0, $
    jb SHIFT_BUTTON, skip%Mb
    dec %1
    sjmp skip%Ma
skip%Mb:
    inc %1
skip%Ma:
ENDMAC
```

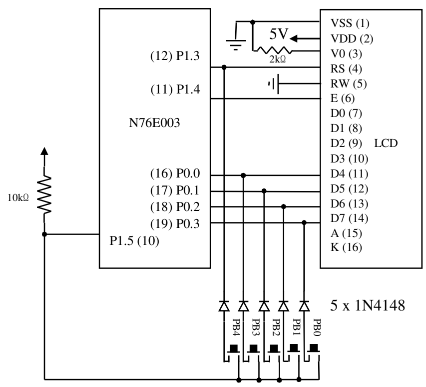
```
    Change_8bit_Variable(MY_VARIABLE_BUTTON, my_variable, loop_c)
    Set_Cursor(2, 14)
    mov a, my_variable
    lcall SendToLCD
    lcall Save_Configuration
loop_c:
```


Extra tips…

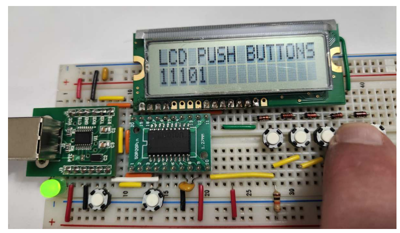
- 'Noisy' measurements? Average!

```
Average_ADC:
    Load_x(0)
    mov R5, #100
Sum_loop0:
    lcall Read_ADC
    mov y+3, #0
    mov y+2, #0
    mov y+1, R1
    mov y+0, R0
    lcall add32
    djnz R5, Sum_loop0
    load_y(100)
    lcall div32
    ret
```

Need More Pushbuttons? Multiplex them using the control signals of the LCD.



Using this technique you can read up to five pushbuttons using only one extra input. Example source code provided on Canvas.



Extra tips…

- Op-amp has to much offset? Zero it! (Should not be needed for OP07)

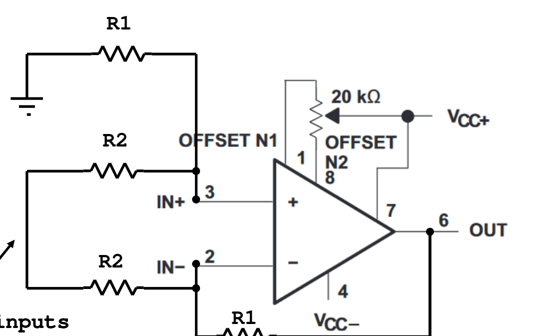


Figure 2. Input Offset-Voltage Null Circuit

Adjust pot until 'OUT' is near zero mV.

Connect inputs of difference amplifier

Extra tip

- There is ‘magic’ value of gain that will give you the temperature of the thermocouple (minus cold junction) directly when reading from the ADC!

The smallest component we are using is ‘0805’ size