University of British Columbia
Electrical and Computer Engineering
EECE281/EECE282

Project 1 – EFM8 board, FSM, SPI EEPROM, 7-Seg Disp

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 2, 2018

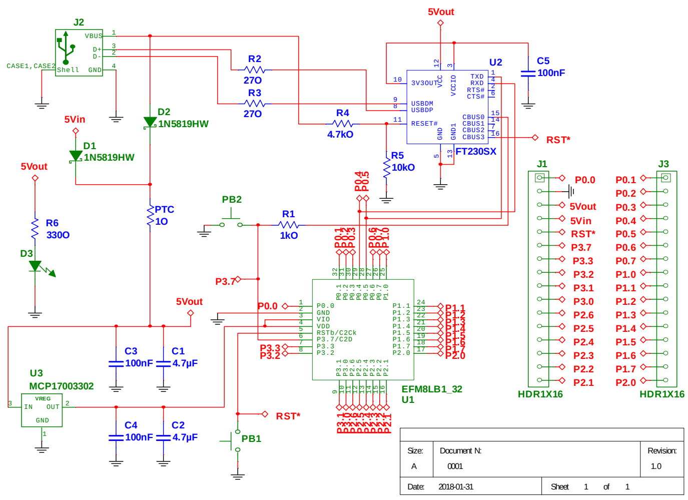
Objectives

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

The EFM8 Board

- Each student should have a F38x 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. (I personally believe this is 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 AT89LP51RC2:
  - 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

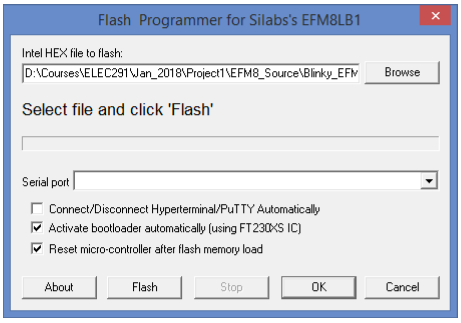
```
$MODEFM8LB1

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
```

MODEFM8LB1: Special Function Register definitions for EFM8LB1

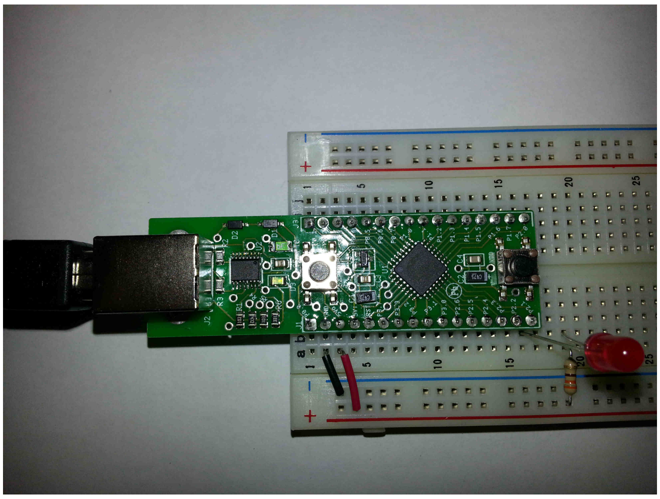
- Not included with CrossIDE. Download from Connect.
- Copy to Call51/define.

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.

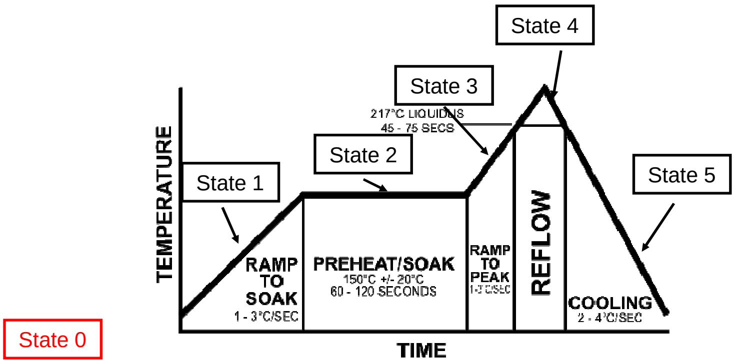


Finite State Machines in Assembly Language

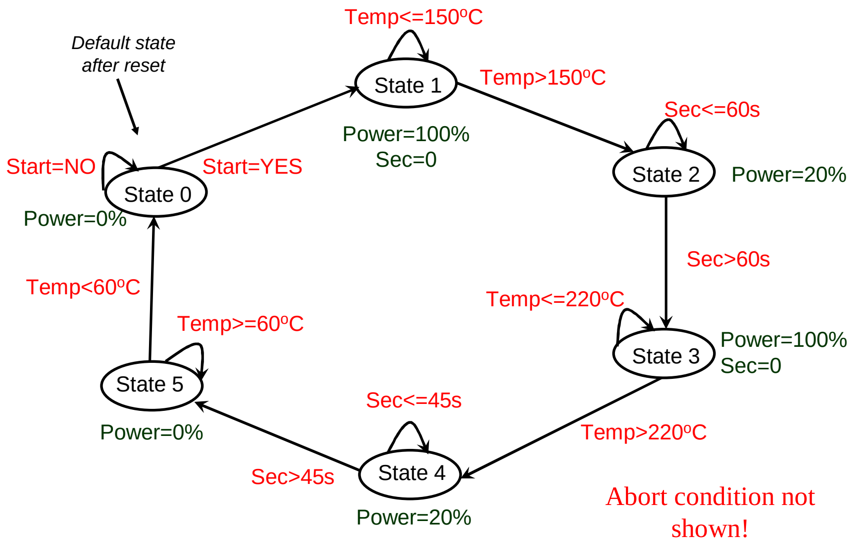
- A finite state machine (FSM) is a programming abstraction method that can be represented using a graph structure.
- We can draw the states as circles and the transitions as arrows.
- There is a finite number of states. The active state is called the current state.
- FSMs are easily implemented in assembly 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!

In assembly (some states only!)

```
    mov a, state

state0:
    cjne a, #0, state1
    mov pwm, #0
    jb KEY.3, state0_done
    jnb KEY.3, $ ; Wait for key release
    mov state, #1
state0_done:
    ljmp forever

state1:
    cjne a, #1, state2
    mov pwm, #100
    mov sec, #0
    mov a, #150
    clr c
    subb a, temp
    jnc state1_done
    mov state, #2
state1_done:
    ljmp forever

state2:
    cjne a, #2, state3
    mov pwm, #20
    mov a, #60
    clr c
    subb a, sec
    jnc state2_done
    mov state, #3
state2_done:
    ljmp forever
        .
        .
        .
        .
        .
        .
        .
```


In assembly (some states only!) using variables…

```
    mov a, state

state0:
    cjne a, #0, state1
    mov pwm, #0
    jb KEY.3, state0_done
    jnb KEY.3, $ ; Wait for key release
    mov state, #1
state0_done:
    ljmp forever

state1:
    cjne a, #1, state2
    mov pwm, #100
    mov sec, #0
    mov a, temp_soak
    clr c
    subb a, temp
    jnc state1_done
    mov state, #2
state1_done:
    ljmp forever

state2:
    cjne a, #2, state3
    mov pwm, #20
    mov a, time_soak
    clr c
    subb a, sec
    jnc state2_done
    mov state, #3
state2_done:
    ljmp forever
        .
        .
        .
        .
        .
        .
        .
DSEG ; Before the state machine!
temp_soak: ds 1
Time_soak: ds 1
Temp_refl: ds 1
Time_refl: ds 1
```

About Variables

- 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 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 this simple subroutine:

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

```
; Eight bit number to display passed in 'a'.
PuTTy_Accumulator:
       mov b, #100
       div ab
       orl a, #0x30 ; Convert hundreds to ASCII
       lcall putchar ; Send
       mov a, b     ; Remainder is in register b
       mov b, #10
       div ab
       orl a, #0x30 ; Convert tens to ASCII
       lcall putchar ; Send
       mov a, b
       orl a, #0x30 ; Convert units to ASCII
       lcall putchar ; Send
       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: 93C66

FMD
Fremont Micro Devices **93C46/A, 93C56/A, 93C66/A**

3-Wire Serial EEPROM
1K, 2K and 4Kbit (8-bit or 16-bit wide)

FEATURES

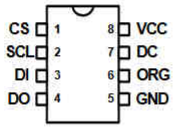
- Standard Voltage and Low Voltage Operation:
  - FT93C46/56/66: V_{CC} = 2.5V to 5.5V
  - FT93C46A/56A/66A: V_{CC} = 1.8V to 5.5V
- User Selectable Internal Organization:
  - FT93C46: 128 x 8 or 64 x 16
  - FT93C56: 256 x 8 or 128 x 16
  - FT93C66: 512 x 8 or 256 x 16
- 2 MHz Clock Rate (5V) Compatibility.
- Industry Standard 3-wire Serial Interface.
- Self-Timed ERASE/WRITE Cycles (5ms max including auto-erase).
- Automatic ERAL before WRAL.
- Sequential READ Function.
- High Reliability: Typical 1 Million Erase/Write Cycle Endurance.
- 100 Years Data Retention.
- Industrial Temperature Range (-40^{o} C to 85^{o} C).
- Standard 8-pin PDIP/SOIC/TSSOP Pb-free Packages.

Non-Volatile Memory: 93C66

PIN CONFIGURATION

Pin Name	Pin Function
CS	Chip Select
SCL	Serial Clock
DI	Serial Data Input
DO	Serial Data Output
ORG	Internal Organization
DC	Don't Connect
VCC	Power Supply
GND	Ground

All these packaging types come in Pb-free certified.

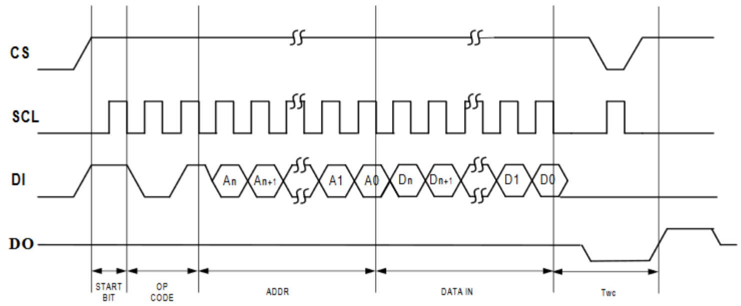


8L PDIP

Non-Volatile Memory: 93C66

SPI!

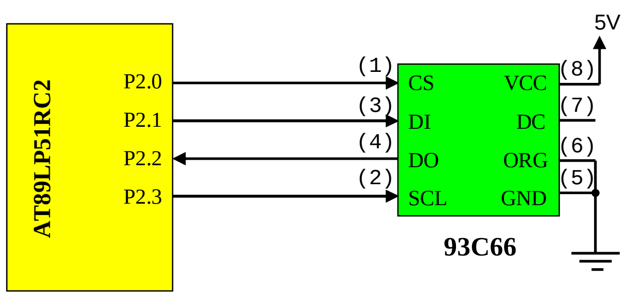
WRITE Timing



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, such as the last reflow profile.

Connecting the 93C66 to the AT89LP51RC2



WARNING: P2.0 was also used to enable/disable the MCP3008 ADC!

93C66 Instruction Set

INSTRUCTION SET for the FT93C56 and FT93C66

Instruction	SB	Op Code	Address		Data		Comments
			x 8	x 16	x 8	x 16	
READ	1	10	A_8- A_0	A_7- A_0			Reads data stored in memory, at specified address.
EWEN	1	00	11xxxxxxx	11xxxxxx			Write enable must precede all programming modes.
EWDS	1	00	00xxxxxxx	00xxxxxx			Disables all programming instructions.
ERASE	1	11	A_8- A_0	A_7- A_0			Erase memory location A_n - A_0.
WRITE	1	01	A_8- A_0	A_7- A_0	D_7 - D_0	D_{15} - D_0	Writes memory location A_n - A_0.
ERAL	1	00	10xxxxxxx	10xxxxxx			Erases all memory locations.
WRAL	1	00	01xxxxxxx	01xxxxxx	D_7 - D_0	D_{15} - D_0	Writes all memory locations.

93C66 Instruction Set (code)

```
; Address to read passed in dptr (dpl and dph)
; Value read, returned in accumulator
; READ: 110[A8] [A7 downto A0]
FT93C66_Read:
    setb FT93C66_CE  ; Activate the EEPROM.
    lcall SmallDelay
    mov a, #1100B
    orl a, dph
    mov R0, a ; Send start bit, op code, and MSB of address
    lcall DO_SPI_G
    mov R0, dpl ; Send LSB of address
    lcall DO_SPI_G
    mov R0, #11111111B ; Dummy value to receive data
    lcall DO_SPI_G
    mov a, R1
    clr FT93C66_CE ;  De-activate the EEPROM.
    ret
```

93C66 Functions Provided

- FT93C66_Write_Enable
- FT93C66_Write_Disable
- FT93C66_Read
- FT93C66_Erase
- FT93C66_Erase_All
- FT93C66_Write
- FT93C66_Write_All

93C66 Functions Example

```
lcall FT93C66_Write_Enable

mov dptr, #0x10 ; Random memory location to test
mov a, 0x55 ; Value to write at location
lcall FT93C66_Write
lcall FT93C66_Read
cjne a, #0x55, it_failed ; Read back and check
```

Example code in web page: EEPROM_test.asm

Example: Writing Project Data to EEPROM

```
Save_Configuration:
    lcall FT93C66_Write_Enable
    mov DPTR, #0
    ; Save variables
    mov a, temp_soak
    lcall FT93C66_Write
    inc DPTR
    mov a, time_soak
    lcall FT93C66_Write
    inc DPTR
    mov a, temp_refl
    lcall FT93C66_Write
    inc DPTR
    mov a, time_refl
    lcall FT93C66_Write
    inc DPTR
    mov a, #0x55 ; First key value
    lcall FT93C66_Write
    inc DPTR
    mov a, #0xAA ; Second key value
    lcall FT93C66_Write
     lcall FT93C66_Write_Disable
    ret
```


Example: Read Project Data From EEPROM; check keys first!

```
Load_Configuration:
   mov dptr, #0x0004 ;First key value location.  Must be 0x55
   lcall FT93C66_Read
   cjne a, #0x55, Load_Defaults
   inc dptr ; Second key value location.  Must be 0xaa
   lcall FT93C66_Read
   cjne a, #0xaa, Load_Defaults
```

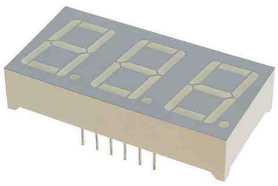
Example: Read From Flash; Load Saved Values

```
; Keys are good.  Load saved values.
mov dptr, #0x0000
lcall FT93C66_Read
mov temp_soak, a
inc dptr
lcall FT93C66_Read
mov time_soak, a
inc dptr
lcall FT93C66_Read
mov temp_refl, a
inc dptr
lcall FT93C66_Read
mov time_refl, a
ret
```

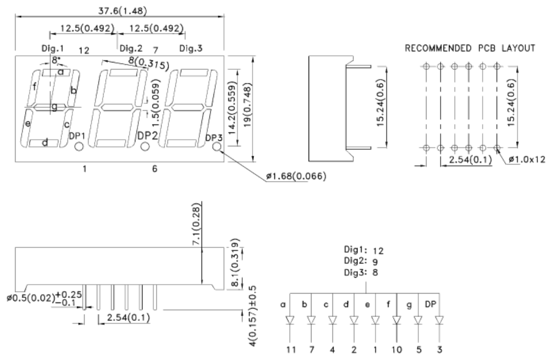

Example: Read From Flash; Load Default Values

```
Load_Defaults: ; Load defaults if keys are incorrect
   mov temp_soak, #150
   mov time_soak, #45
   mov temp_refl, #225
   mov time_refl, #30
   ret
```

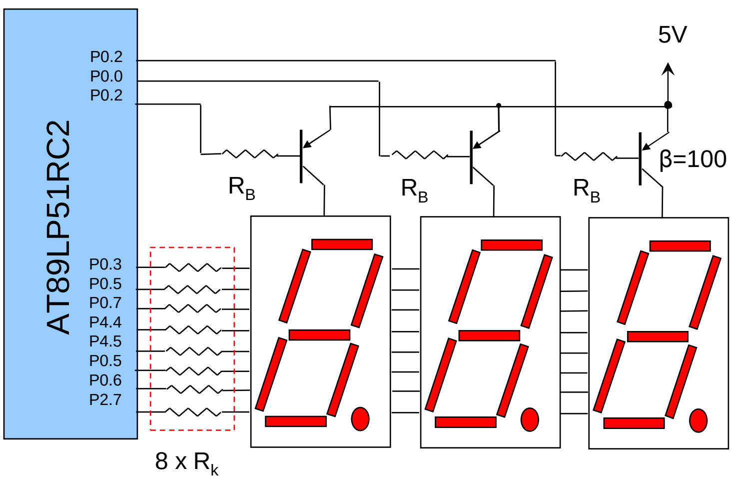
Multiplexed Displays



BA56-12SRWA



Multiplexed Displays



Multiplexed Displays: Calculating the Resistors

Electrical / Optical Characteristics at TA=25°C

Symbol	Parameter	Emitting Color	Typ.	Max.	Units	Test Conditions
λpeak	Peak Wavelength	Super Bright Red	655		nm	I_F=10mA
λD [1]	Dominant Wavelength	Super Bright Red	640		nm	I_F=10mA
Δλ1/2	Spectral Line Half-width	Super Bright Red	20		nm	I_F=10mA
C	Capacitance	Super Bright Red	45		pF	V_F=0V;f=1MHz
V_F [2]	Forward Voltage	Super Bright Red	1.8	2.5	V	I_F=10mA
I_R	Reverse Current	Super Bright Red		10	uA	V_R=5V

Notes:
1. Wavelength: +/-1nm.
2. Forward Voltage: +/-0.1V.
3. Wavelength value is traceable to CIE127-2007 standards.
4. Excess driving current and / or operating temperature higher than recommended conditions may result in severe light degradation or premature failure.

10mA per segment sounds reasonable.

Multiplexed Displays: Calculating the Resistors

Worst case collector current (all LEDs on):

$$I_C = 8 \times 10mA = 80mA$$

Worst case base current:

$$I_B = \frac{I_C}{\beta} = \frac{80mA}{100} = 0.8mA$$

Maximum allowed base resistor R_B:

$$R_B = \frac{V_{CC} - V_{EB}}{I_B} = \frac{5V - 0.7V}{0.8mA} = 5.3k\Omega$$

Multiplexed Displays: Calculating the Resistors

Minimum LED Cathode resistance R_K:

$$R_K = \frac{V_{CC} - V_{SAT} - V_F}{I_F} = \frac{5.0 - 0.2 - 1.8}{10mA} = 300\Omega$$

$$\boxed{\begin{aligned} R_B &= 1k\Omega \\ R_K &= 330\Omega \end{aligned}}$$

Multiplexed Displays: Code

```
; For the 7-segment display
SEGA equ P0.3
SEGB equ P0.5
SEGC equ P0.7
SEGD equ P4.4
SEGE equ P4.5
SEGF equ P0.4
SEGG equ P0.6
SEGP equ P2.7
CA1  equ P0.2
CA2  equ P0.0
CA3  equ P0.1

 dseg at 0x30
 .
 .
 Disp1:  ds 1
 Disp2:  ds 1
 Disp3:  ds 1
 state:  ds 1
```

```
; Pattern to load passed in acc
load_segments:
       mov c, acc.0
       mov SEGA, c
       mov c, acc.1
       mov SEGB, c
       mov c, acc.2
       mov SEGC, c
       mov c, acc.3
       mov SEGD, c
       mov c, acc.4
       mov SEGE, c
       mov c, acc.5
       mov SEGF, c
       mov c, acc.6
       mov SEGG, c
       mov c, acc.7
       mov SEGP, c
       ret
```

Using P4.4 as General Purpose I/O

AUXR = 8EH — Reset Value = 0000 10X0B

Not Bit Addressable

	DPU	WS1[1]	WS0	XRS2	XRS1	XRS0	EXTRAM	AO
Bit	7	6	5	4	3	2	1	0

Symbol	Function
DPU	Disable Weak Pull-up. When DPU = 0 all I/O ports in quasi-bidirectional mode have their weak pull-up enabled. When DPU = 1 all I/O ports in quasi-bidirectional mode have their weak pull-up disabled to reduce power consumption.
WS_{1-0}	Wait State Select. Determines the number of wait states inserted into external memory accesses.
XRS_{2-0}	XRAM Size. Selects the size of the on-chip extra RAM (EDATA)
EXTRAM	External RAM Enable. When EXTRAM = 0, MOVX instructions can access the internally mapped portions of the address space (Extra RAM). Accesses to addresses above internally mapped memory will access external memory. Set EXTRAM = 1 to bypass the internal memory and map the entire 64KB address space to external memory. The default state of EXTRAM is set by a user configuration fuse. See Section 24.2 on page 188.
DISALE	ALE Output. When AO = 0 the ALE pulse is active at 1/3 of the system clock frequency in Compatibility mode and 1/2 of the system clock frequency in Fast mode. When AO = 1 the ALE is inactive (high) unless an external memory access occurs. AO must be set to use P4.4 as a general I/O.

WS1	WS0	Wait States	$\overline{RD}$ / $\overline{WR}$ Strobe Width	ALE to $\overline{RD}$ / $\overline{WR}$ Setup
0	0	0	1 x t_{CYC} (Fast); 3 x t_{CYC} (Compatibility)	1 x t_{CYC} (Fast); 1.5 x t_{CYC} (Compatibility)
0	1	1	2 x t_{CYC} (Fast); 15 x t_{CYC} (Compatibility)	1 x t_{CYC} (Fast); 1.5 x t_{CYC} (Compatibility)
1	0	2	2 x t_{CYC} (Fast)	2 x t_{CYC} (Fast)
1	1	3	3 x t_{CYC} (Fast)	2 x t_{CYC} (Fast)

XRS2	XRS1	XRS0	EDATA Size (bytes)	Address Range
0	0	0	256	0000H–00FFH
0	0	1	512	0000H–01FFH
0	1	0	768 (default)	0000H–02FFH
0	1	1	1024	0000H–03FFH
1	0	0	1152	0000H–047FH
1	0	1	Reserved	
1	1	–	Reserved	

```
mov AUXR, #00010001B
```

Typo!

Multiplexed Displays: code

```
;;;  State machine for 7-segment displays starts here
        ; Turn all displays off
        setb CA1
        setb CA2
        setb CA3

        mov a, state
state0:
        cjne a, #0, state1
        mov a, disp1
        lcall load_segments
        clr CA1
        inc state
        sjmp state_done
state1:
        cjne a, #1, state2
        mov a, disp2
        lcall load_segments
        clr CA2
        inc state
        sjmp state_done
state2:
        cjne a, #2, state_reset
        mov a, disp3
        lcall load_segments
        clr CA3
        mov state, #0
        sjmp state_done
state_reset:
        mov state, #0
state_done:
;;;  State machine for 7-segment displays ends here
```

Put this code in a timer ISR

Multiplexed Displays: code

```
HEX_7SEG: DB 0xC0, 0xF9, 0xA4, 0xB0, 0x99,
          DB 0x92, 0x82, 0xF8, 0x80, 0x90
```

Use like this:

```
mov dptr, #HEX_7SEG
mov a, BCD_counter
anl a, #0x0f
movc a, @a+dptr
mov disp1, a
mov a, BCD_counter
swap a
anl a, #0x0f
movc a, @a+dptr
mov disp2, a
mov disp3, #0xff
```


Extra Tips…

- Are you using macros yet?

```
Change_8bit_Variable MAC
    jb %0, %2
    Wait_Milli_Seconds(#50)
    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 Display_Accumulator
    lcall Save_Configuration
loop_c:
```

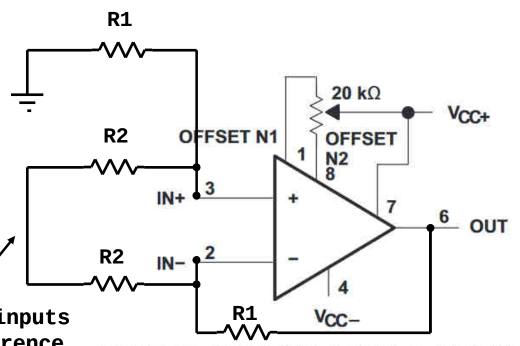
Extra tips…

- ‘Noisy’ measurements? Average!

```
Average_ADC_Channel MAC
    mov b, #%0
    lcall ?Average_ADC_Channel
ENDMAC

?Average_ADC_Channel:
    Load_x(0)
    mov R5, #100
Sum_loop0:
    lcall Read_ADC_Channel
    mov y+3, #0
    mov y+2, #0
    mov y+1, R7
    mov y+0, R6
    lcall add32
    djnz R5, Sum_loop0
    load_y(100)
    lcall div32
    ret
```


Extra tips…

- Op-amp has to much offset? Zero it!



Connect inputs of difference amplifier

Adjust pot until ‘OUT’ is near zero mV.

Figure 2. Input Offset-Voltage Null Circuit