Laboratory Worksheet #06
Analog-to-Digital Conversion

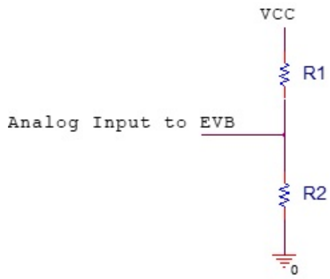


Figure 1: Voltage Divider

The above resistor network has two resistors, R1 and R2 (neither of these are potentiometers), connected in series between 5V and ground (0V). Recall, the total resistance for a series network is $R_{Tot} = R1 + R2$. The voltage across resistors in series is found by using voltage divider concepts. For the circuit shown, the voltage across each resistor is:

$$V_{R1} = \frac{R1}{R1 + R2}(Vcc) \qquad\qquad V_{R2} = \frac{R2}{R1 + R2}(Vcc)$$

In general, for N resistors in series, the voltage across the ith resistor is given by

$$V_{Ri} = \frac{Ri}{\sum_{n=1}^{N} R_n}(Vcc)$$

When answering the following questions, the current flowing through the two resistors is 2.5mA and the voltage input to the EVB is 1.25V when VCC = 5V.

Exercise 1:

1)	What is the total resistance of the two resistors?	2000Ω
2)	What is the voltage across R1?	3.75V
3)	What is the voltage across R2?	1.25V
4)	What is the resistance value of R1?	1500Ω
5)	What is the resistance value of R2?	500Ω

6) Using the closest resistor components you can find, build the small circuit and measure the voltage at the EVB input using the voltmeter. How close is it to the spec? (Put your resistors back when finished.)

3.72V. This value is very close to the spec.

Exercise 2: Software Initialization and A/D conversion

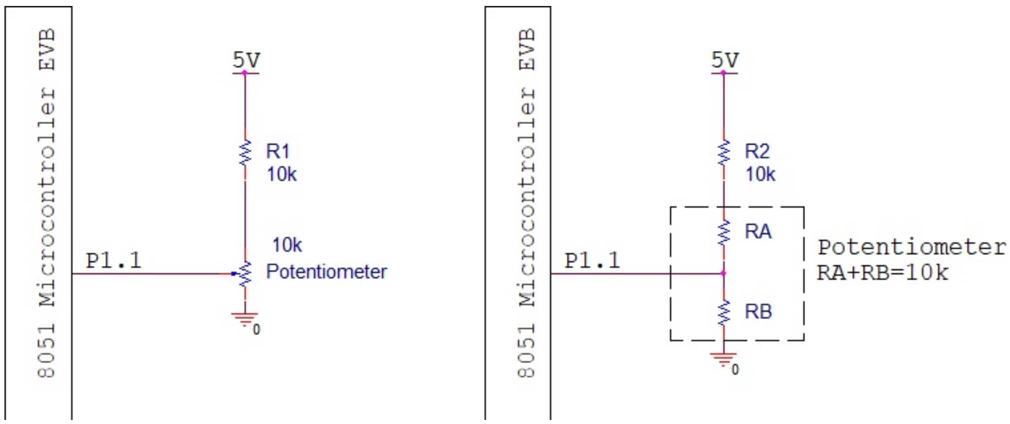


Figure 2: Potentiometer Circuit

In the above circuit, a resistor in series with a potentiometer is shown on the left. The equivalent circuit is shown on the right, with the two variable resistors inside the potentiometer. When determining voltage input to P1.1, you should treat the potentiometer as two resistors in series, making the effective circuit three resistors in series. The voltage at P1.1 is equal to the voltage across the 'bottom' resistor of the potentiometer, RB. The general voltage divider expression from Exercise 1 applies to the circuit. Answer the following questions based on the schematic shown above.

1) Write a function emphvoid Port_Init(void) to configure P1.1 as an analog input. Your function should leave the other bits (0, 2-7) unchanged.

P1MDIN &= ~0x02;	//Set P1.1 as an analog input
P1MDOUT &= ~0x02;	// Set P1.1 as a input port bit
P1 |= 0x02;	// Set P1.1 to a high impedance state

2) Write a function emphvoid ADC_Init(void) to set VREF to use the internal voltage reference of 2.4 V, set the ADC gain to 1, and enable ADC1.

REF0CN = 0x03;	// Configure ADC1 to use VREF
	//
ADC1CF |= 0x01;	// Set a gain of 1
ADC1CN = 0x80;	// Enable ADC1

3) Write a function *void ADC_Test(void)* that performs an ADC (analog to digital conversion) on P1.1 and stores the result in variable *P1_1_Result*.

AMX1SL = 1;	// Set the Port pin number
ADC1CN = ADC1CN & ~0x20;	// Clear the conversion complete flag
ADC1CN = ADC1CN | 0x10;	// Start and A/D conversion
while ((ADC1CN & 0x20) == 0x00);	// Wait for the conversion to be complete
P1_1_Result = ADC1;	// Assign the A/D conversion result

4) When the potentiometer is adjusted such that RA = 6k and RB = 4k, what is the voltage at P1.1? (Apply voltage divider concepts.)

1V

5) Considering parts 3 and 4, what value (unsigned char) will be stored in the variable P1_1_Result?

106

6) If the voltage at P1.1 is now changed to 3.53 V, what value will be stored in P1_1_Result?

255

When complete, include Worksheet 6 with your Laboratory 2 Pre-lab submission.

EVB Pin	
1	2
3	4
5	6
7	8
9	10
11	12
13	14
15	16
17	18
19	20
21	22
23	24
25	26
27	28
29	30
31	32
33	34
35	36
37	38
39	40
41 ⟷	60

	Port Bit	Bit Addresses & Labels
1.		
2.		
3.		
4.		
5.		
6.		
7.		
8.		
9.		
10.		
11.		
12.	P1.0	0x90 POT
13.		
14.		
15.		
16.		
17.		
18.		
19.		
20.		
21.		
22.		
23.		
24.	P2.4	0xA4 SS0
25.	P2.5	0xA5 SS1
26.		
27.	P2.2	0xA2 PB2
28.	P2.3	0xA3 PB3
29.	P2.0	0xA0 PB0
30.	P2.1	0xA1 PB1
31.		
32.		
33.	P3.4	0xB4 BILED0
34.	P3.5	0xB5 BILED1
35.	P3.2	0xB2 LED2
36.	P3.3	0xB3 LED3
37.	P3.1	0xB1 LED1
38.	P3.0	0xB0 LED0
39.		
40.		

Software Initializations

A) Port I/0

```
P1MDIN &= 0xFE;
P1MDOUT &= 0xFE;
P1 |= ~0xFE;
P2MDOUT &= ~0x3F;
P2 |= 0x3F;
P3MDOUT |= 0x3F;
```

B) Timers

```
CKCON |= 0x08;
TMOD &= 0xF0;
TMOD |= 0x02;
TR0 = 0;
TMR0 = 0;
```

C) Interrupts

```
IE |= 0x02;
EA = 1;
```

D) A/D

```
REF0CN = 0x03;
ADC1CN = 0x80;
ADC1CF |= 0x01;
```

E) PCA

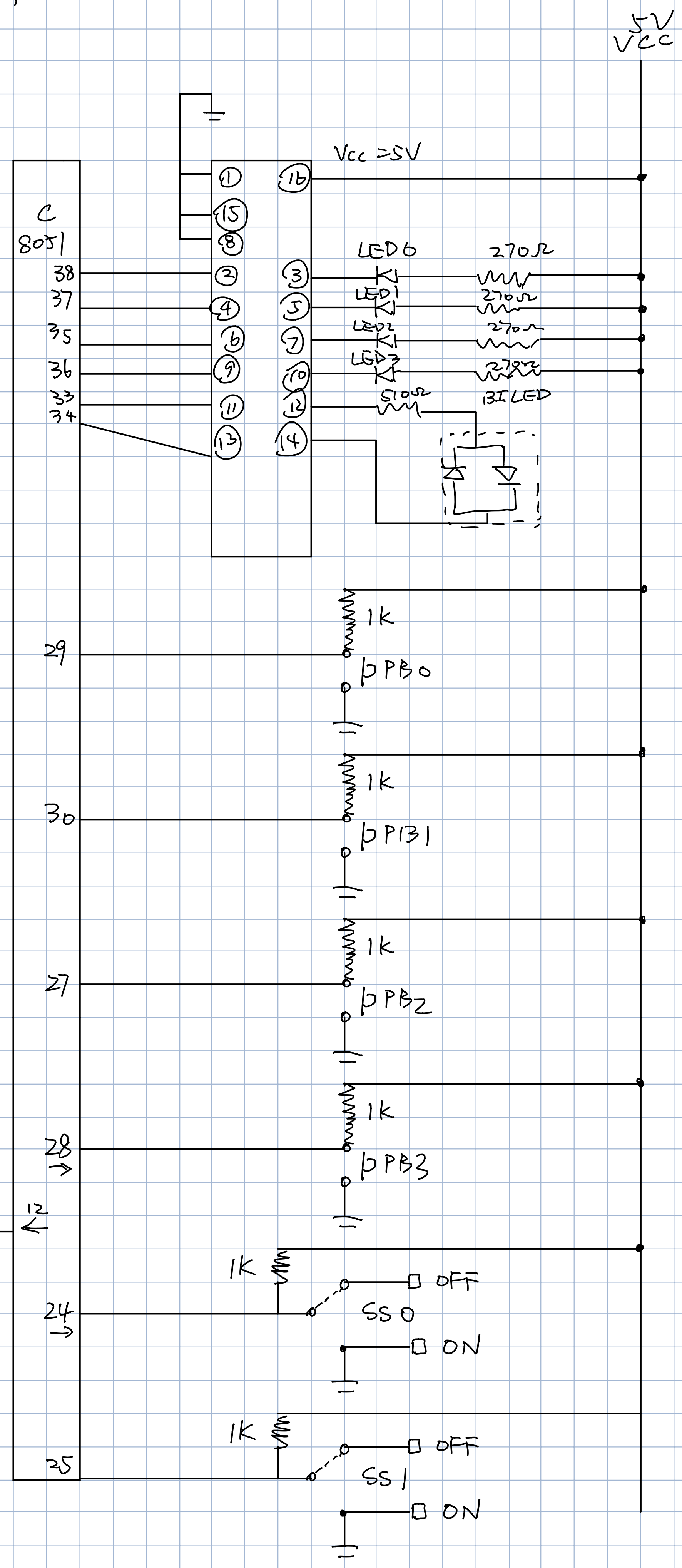
F) XBAR

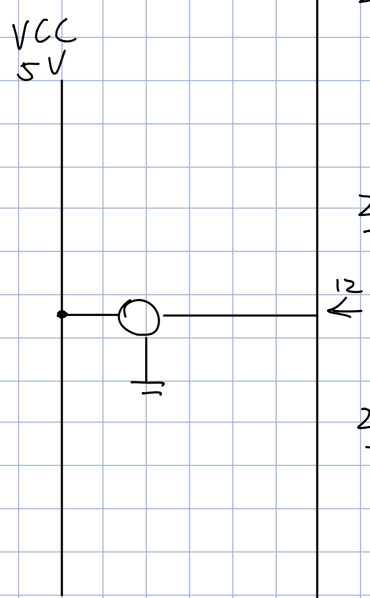
G) I2C

4 pushbuttons
2 SS
4 LEDS
1 BiLED
1 potentiometer
resistors.
1 74365

port 2 ⇒ input
port3 ⇒ output

PB 0	P2.0	# 29
PB 1	P2.1	#30
PB 2	P2.2	# 27
PB 3	P2.3	# 28
SS 0	P2.4	# 24
SS 1	P2.5	# 25
LED 0	P3.0	# 38
LED 1	P3.1	# 37
LED 2	P3.2	# 35
LED 3	P3.3	# 36
BILED0	P3.4	# 33
BILED1	P3.5	# 34
potentiometer	P1.0	#12





```
Compile directives
   #include <c8051_SDCC.h>
   #include <stdio.h>
   #include <stdlib.h>

Function prototypes
   void Port_Init(void);
   void Timer_Init(void);
   void Interrupt_Init(void);
   void ADC_Init(void)
   void Timer0_ISR(void) __interrupt 1;
   void random(void);
   unsigned char read_AD_input(unsigned char pin_number) // read the AD_value
   void Potent_Reader(void); // The function to read the potentiometer and set the on-time;
   void Mode_1(void); // Set the blink pattern for mode 1
      void Sequence_1(void); // Function to control the sequence 1
      void Sequence_2(void); // Function to control the sequence 2
      void Sequence_3(void); // Function to control the sequence 3
   void Mode_2(void); // The function to count the number PB pushed
   void Mode_3(void); // Set the function of mode 3
   void Verifier(void); // The function to check the wire connection

Declare global variables
   sbit LED0, LED1, LED2, LED3, BILED0, BILED1, SS0, SS1, PB0, PB1, PB2, PB3
   unsigned int Counts // Timer 0 counter
   unsigned char Potent_AD // A/D converter result
   unsigned char Current_Random
   unsigned char Previous_Random
   unsigned int Button_Status_Counter

Main function
   declare local variables
      (NONE)
   Initialization functions
      Sys_Init();
      Port_Init();
      Interrupt_Init();
      Timer_Init();
      ADC_Init();
      putchar(' ');
      Turn off all the outputs
      Turn off Timer0
      Begin infinite loop
         Check the button status to determine the mode
         call the designated mode
      End infinite loop
End main function

functions
   void Port_Init(void)

   void Mode_1(void)
      Call the Potent_Reader(void) and get the Potent_AD result.
```

```
   Using the conversion to get the number of Counts required for the Timer0 to stop the required time.
   Call Sequence_1() to blink sequence 1.
   Call Sequence_2() to blink sequence 2.
   Call Sequence_3() to blink sequence_3.
Exit the Mode_1(void)


void Mode_2(void)
   declare local variables
      unsigned int Player_1_Score;
      unsigned int Player_2_Score;
   Turn off all LEDs
   Repeat 3 times
   Call the Potent_Reader(void) and get the Potent_AD result.
   Using the conversion to get the number of Counts required for the Timer0 to stop the required time.
   Turn the left LED on for the given time
      Count the number left button was pushed
         Increment the Player_1_Score
      check to see if the right button is pushed
         if pushed, right player lose the game
   Ensure that the left button is not still pressed using Button_Status_Counter.
   Turn on the right LED on for the given time
      Count the number right button was pushed
         Increment the Player_2_Score
      check to see if the left button is pushed
         if pushed, left player lose the game
   Ensure that the right bbutton is not still pressed using Button_Status_Counter.
   Display the score for both players and compare the value of Player_1_Score and Player_2_Score.
   Light up the BILEd depends in the scores
Exit the Mode_2(void);

void Mode_3(void);
   declare local variables
      unsigned int Player_1_Score;
      unsigned int Player_2_Score
   Turn off all LEDs
   Call the Potent_Reader(void) and get the Potent_AD result.
   Using the conversion to get the number of Counts required for the Timer0 to stop the required time.
   Repeat 5 times
      Set a clockwise blink pattern
         Determining the starting LED using Random_Number from random
         Repeat
            Blink the appropriate LED
            Check id the button is pushed and released during onn-time(one time push)
         If button was correectly pressed
            Add total number of LED blinks to the Player_1_Score
            Exit the clockwise blink pattern
      Set a counter_clockwise blink pattern
         Determining the starting LED using Random_Number from random
         Repeat
            Blink the appropriate LED
            Check id the button is pushed and released during onn-time(one time push)
         If button was correectly pressed
            Add total number of LED blinks to the Player_2_Score
```

```
            Exit the clockwise blink pattern
    Display score and determine winner player based on comparison
    Set the BILED
Exit the Mode_3(void)

void Port_Init(void)
    Set Port P1.0 as *analog inputs to 1
    Set Port P2.0, P2.1, P2.2, P2.3, P2.4, P2.5 as *digital inputs to 0
    Set Port P1.0, P2.0, P2.1, P2.2, P2.3, P2.4, P2.5 as inputs
    Set Port P3.0, P3.1. P3.2, P3.3, P3.4, P3.5 as digital outputs
    Set Port P1.0, P2.0, P2.1, P2.2, P2.3, P2.4, P2.5 to high impedence.
Exit the Port_Init(void)

void Interrupt_Init(void)
    Enable Timer0 Interrupt request (by masking).
    Enable global interrupts (by sbit).

void Timer_Init(void)
    Set Timer0 as stated in the manual, Use 16bit and SYSCLK.
    Stop the Timer0 for now.
    Clear high and low byte of T0

void ADC_Init(void)
    Set Vref to use internal ref. Voltage 2.4V.
    Enable A/D converter.
    Set A/D converter gain to 1.

void Timer0_ISR(void) __interrupt 1
    Increment the Counts and Button_Status_Counter to count the number of overflows

void random(void)
    Begin infinite loop
        generate random numbers
        Exit while the new random number is not the same as the old one
        Set the Previous_Random the same as the Current_Random
    End infinite loop

void Potent_Reader(void)
declare local variables
    unsigned char AD_value
Call the unsigned char read_AD_input(unsigned char pin_number) to get the AD_value
Convert the AD_value to the number of Counts

unsigned char read_AD_input(unsigned char pin_number)
    Set P1.n as the analog input for ADC1.
    Clear the convertion completed flag
    Initiate A/D convertion.
    Wait for the conversion to be completed.
    Return the digital value in ADC1 register.
```