Lab 2: A Microprocessor-Controlled Game – LED/Tracker

Preparation

Reading

Lab Manual:

Chapter 2 - Lab Equipment

Chapter 4 - The Silicon Labs C8051F020 and the EVB (sections relating to Timer 0 and A/D Conversion)

C language reference concepts:

data types, declarations, variables, variable scope, symbolic constants, functions, modular program development, variable passing, arrays

Embedded Control Multimedia Tutorials

Hardware: Multimeter and Logic Probe

Basics: Analog/Digital Conversion

Objectives

General

1. Introduction to analog-to-digital conversion using C8051 and development of a simple interactive game using the C8051 as the controller, and utilizing various switches and LEDs on the protoboard for game I/O.

Hardware

1. Familiarization with the use of the multimeter as a voltmeter.
2. Configuration of a potentiometer to provide variable voltage input to the analog to digital input port of the C8051.
3. Keep the circuit from Laboratory 1-2, adding LEDs, Pushbuttons and more as specified.

Software

1. The speed at which a new random value is generated or other operations may be a function of the digital conversion read from the *game speed potentiometer*.
2. Further refinement of programming skills requiring bit manipulation.
3. Adaptation and re-use of software modules developed in previous lab exercises.
4. Refinement of skills in *top-down* and *modular program development* on a larger program.
5. Add the port initializations for the added hardware components.

Motivation

In the previous labs, you explored the use of *digital* inputs and outputs. However, most of the *"real world"* is *analog*. In this lab, you will explore how the C8051 can convert an *analog* signal into a *digital* value. The skills refined and developed in this lab exercise will be applicable to the following labs—the *Smart Car & Gondola.* Portions of those labs may also require the use of LEDs to indicate the status of various system parameters, the use of switches for input, the use of A/D conversion to determine the game speed or other parameters, and the use of interrupts to keep track of time.

Lab Description & Activities

Description of Game Objective

This game is called *LITEC LED/Tracker*. Two slide switches are used to determine which of the 3 distinct modes of the game will run after displaying instructions on the terminal. Binary values of off & on, on & off, and on & on select 1, 2, or 3 respectively after the pushbutton is pressed. (Students may decide what off & off will do.) The first mode generates a random number from 0 to 4. The generated number lights up none to all 4 LEDs in a line. The LEDs stay on for 0.5 s, then turn off and a timer is started. The player then turns a potentiometer whose voltage value is used to turn on the LEDs in sequence until the same pattern appears. The pushbutton is then pressed to stop the timer and record the LED pattern, which is then compared to that generated by the random number. Scoring is based on correctness and speed of entry. The whole process is repeated 5 times and the final score is displayed on the terminal.

The second mode uses the same hardware, but involves 2 players. The first player adjusts the potentiometer to light some pattern on the LEDs. Then the second player must push the pushbutton (with debouncing) the number of times to match the number of LEDs that are lit within a set time limit (1.5 s). A point is earned for the correct number of presses. If the count is wrong the BiLED is lit red or green if the count is low or high respectively. The LEDs will flash to indicate the end of the round. Again, the process repeats 5 times and then displays the final score on the terminal.

The third mode uses the potentiometer to draw a bar graph on the terminal in real-time, whose length depends on the ADC value. A function `Draw_Bar()` is provided to draw the bar by repeatedly printing a character on the screen. The max ADC value prints a line 63 characters long. The same function from the first mode is used to light the LEDs based on a random number. This time the 5 cases (none, 1, 2, 3, or all 4 on) tell the player how long the bar should be (0%, 25%, 50%, 75%, or 100% of the maximum length). After displaying a pattern on the LEDs, a timer starts. While the time is less than 1.0 s the player must adjust the pot to draw a line with the correct length. At the end of 1 s the length is compared to the desired value (0, 16, 32, 48, or 63 characters) and the LEDs should all flash off, on, & off. If the length equals the desired length 10 points are awarded. If shorter or longer the BiLED should be lit red or green respectively for 1 s and the score is adjusted by deducting 2 points for every character position off down to a minimum score of 0. This is repeated 5 times and then the final score is displayed.

The goal of each mode is for the player to correctly match the random value as quickly as possible or within the preset time limit. A timer in the 8051 measures the response time and scoring depends on speed and accuracy. A game consists of 5 tries of each mode. At the end of a game the program goes back and displays the original instructions asking the player which mode they desire and reprinting the directions on how to proceed.

Description of Game Components

The components of game consists of 1 pushbutton (PB), 1 bi-color LED (BiLED), 2 slide switches (SS0 & SS1), 1 potentiometer, and 4 LEDs (LED0, LED1, LED2 & LED3). The slide switches are used to select the game mode. The "Enter" pushbutton should be pressed to start a game and is used to indicate when a pot value should be read in mode 1 as well as counting presses in mode 2. If a wrong value is entered in any mode, the bi-color LED turns red or green to indicate that the values was low or high. The potentiometer is used to determine how many LEDs to light or how long a bar to draw on the terminal.

Description of Game Play:

At the start of the game, the BiLED and LEDs should be off and a message providing instructions is displayed on the terminal that include how to select a mode (slide switches) and to press the "Enter" pushbutton when ready to begin the game. A message indicating the state of the 2 slide switches should be printed on the terminal continuously (on the same line with \r but no \n) until the pushbutton is pressed.

To summarize the process before the game starts, your program should:

1. Print instructions on the terminal (pick mode by setting the slide switches).
2. While waiting for the PB to be pressed print the status of the slide switches.
3. As soon as the PB pushbutton is pressed, the selected mode starts by explaining how to play that selected mode.

The program switches to the desired mode and each mode should display a message on the terminal explaining how that mode of the game is played. After playing 5 tries the program should go back to the beginning where the initial instructions were given (*At the start of the game*, above) and repeat the infinite loop.

Mode 1 (Random value lights LEDs, player must match pattern by adjusting the potentiometer)

Display brief instructions. For this mode low score wins
Zero the game score
Turn off all LEDs, light BiLED red
Wait for PB to be pressed again to start the game
For 5 tries:
    Indicate start of turn by turning off BiLED
    Generate a random number (repeated random numbers are allowed) from 0 to 4 and light corresponding LEDs for 0.5 s
    Start the timer
    Continuously read the pot A/D value & light LEDs from none to all 4 until PB is pressed
    Stop the timer
    Save the timer *overflow_count* value
    Flash all the LEDs off, on, & off.
    Light the BiLED red or green if answer is low or high.

Calculate the point score based on the following example:

Answer penalty *points* = 10 for incorrect answer (0 otherwise)

Time penalty *points* = 1 for every 500 ms

Ex) overflow_count = 677 (~2 sec), *points* = 677/169 = 4

(Integer arithmetic truncates, total penalty is 4 for correct, 14 for incorrect)

Display the points for the try and total score on the terminal

Delay for ~0.5 second

End of the 5 tries loop

Display the final score and flash the BiLED red and green for ~1 second

Mode 2 (1st player sets pot, 2nd player press pushbuttons to match the count)

Display brief instructions. For this mode high score wins

Zero the game score

Turn off all LEDs

Store 5 timer values in an array: overflows[5] = ~1.5s, ~1.35 s, ~1.22 s, ~1.10 s, ~1.0 s

For 5 tries:

Wait 1.0s while 1st player set the potentiometer, have BiLED lit red

Read the pot value on the ADC and scale value into the range 0 to 4

Turn off BiLED

Light appropriate LEDs from none to all 4

Clear the timer *overflow_count* value

While timer is less than overflows[try] – time is shortened by ~10% every try

Count debounced pushbutton presses

End while

Compare press count to ADC scaled value

Score 10 points for correct

Flash all the LEDs off, on, & off.

If it count is low or high light the BiLED red or green

Display the points for the try and total score on the terminal

Delay for ~1.0 second

Turn off the BILED

Delay for ~0.5 second

End of the 5 tries loop

Display the final score and flash the BiLED red and green for ~1 second

Mode 3 (Random value lights LEDs, player must match pattern by adjusting the potentiometer)

Display brief instructions. For this mode high score wins
Zero the game score
Turn off all LEDs, Light BILED red
For 5 tries:
Generate a random number (repeated random numbers are allowed) from 0 to 4 and light corresponding LEDs for 0.5 s
Turn off BILED
Start the timer
While time is less than ~1 s
Continuously read the pot A/D value & pass value to `Draw_Bar()`
End of ~1 s loop
Save the (ADC value)/4
Flash all the LEDs off, on, & off.
Light the BiLED red or green if answer is low or high.
Calculate the point score based on the following example:
Bar graph is out of 63, Desired values: 0, 16, 32, 48, or 63
Error = abs(Desired value – Bar graph) x 2
Points = (10 – Error) or 0, whichever is larger
Ex) Bar graph = 28, Desired value = 32
Points = 10 – (32 – 28) x 2 = 2
Display the points for the try and total score on the terminal
Delay for ~0.5 second
End of the 5 tries loop
Display the final score and flash the BiLED red and green for ~1 second

List of Tasks and Needed Functions (not necessarily a complete list)

You should look at this list and make team assignment to distribute the load equally.

- Function to read the potentiometer and scale the value to be in the range of 0 to 4 (Mode 1) or 0 to 63 (Mode 3, even though the bar graph function wants the raw ADC value 0 to 255).
- Function to get a random number in the range of 0 to 4 using the random number generator library function.
- Function to compare given answer to desired value and light the BiLED red or green if low or high.
- Function to flash the BiLED red and green for 1 s.
- Function to initialize all the ports used in the game.

- Task to wire up **and verify** the protoboard with slide switches, pushbuttons, BILED, etc. A test program (similar to what was described in Lab 1) must be written that uses the port initialization function above and verifies the operation of all inputs and outputs.

Each player gets 5 tries during their turn. At the end of the game, the program should display the score and then go back to the initial instructions explaining the 2 modes of the game, wait for the player to set the slide switch, adjust the time delay potentiometer, and press the "Enter" pushbutton.

NOTE: The pushbutton must be debounced for proper game operation. Timer 0 should be in 16-bit mode.

Game Enhancements

You are encouraged to change some parameters (wait and response times, start indications, …) to improve the functionality or playability of your game.

Hardware

Figure 1 shows the hardware configuration for this lab. The potentiometer is connected in series with a 10kΩ resistor between +5V and Ground, with the middle pin connected to P1.1. All other port pin connections are up to the team.

Remember

The neatness of wiring is important to debug the hardware.

The placement of LEDs and corresponding push buttons must be considered. Each button should be close to the corresponding LED where appropriate, and they shouldn't be too closely packed. LEDs used as a bar graph should be in a straight line.

Software

You will need to write a C program to perform as described above. You are required to include code for the A/D conversion. At the end of this assignment a C programs provides an example for performing A/D conversion. The A/D reference voltages must be set to be 2.4 Volts and the A/D conversion gain must be set to be 1.

Following is a summary of A/D conversion SFR implementation.

1. In program initialization:
   - Configure analog input pins - set desired A/D pins in P1MDOUT to "0" and P1 to "1" and P1MDIN to "0".

   - Configure reference - set internal reference by clearing REF0CN bit 3 and setting bits 0 & 1:

```
REF0CN=0x03;
```

   - Configure A/D converter gain - set to gain of 1:

```
ADC1CF |= 0x01; ADC1CF &= 0xFD;
```

- Enable converter:

```
ADC1CN = 0x80;
```

2. Conduct A/D conversion:
   - Set pin to convert with AMUX1SL.

```
AMUX1SL = XX; //XX: 0-7
```

   - Clear conversion complete bit

```
ACD1CN &= ~0x20;
```

   - Start conversion:

```
ACD1CN |= 0x10;
```

   - Wait for conversion complete:

```
while((ADC1CN & 0x20) == 0x00);
```

3. Read results:
   - Access results register: *ADresult* = ADC1;

You will configure Timer0 using SYSCLK and 16-bit counting, as in Laboratory 1.2.

Parallel Development

It is **not** necessary to wait until all the circuit hardware is in place before software can be run and debugged. Functions that depend on sbit inputs (switches) or outputs (LEDs) may easily use terminal inputs or outputs to simulate those devices.

Replace statements waiting for any input with a `key = getchar()` that will wait for a terminal keyboard press. If different pushbuttons are expected then `if(key == 'a')` or `if(key == 'b')` conditions can be used to change what happens next. Instead of setting sbits for outputs to your circuit a `putchar('1')` or `putchar('0')` statement to the terminal can simulate turning on and off an LED.

Once the circuit has been built and tested these statements should be commented out and the actual sbit statements enabled.

Demonstration and Verification

1. Complete the psuedo-code that describes the program operation
2. Complete the Pin-out form, labeling the Port bits used, *sbit* variables, and initialization values
3. Use the multimeter to demonstrate that the potentiometer is connected correctly.
4. Run your C program and demonstrate that it performs as stated above.
5. Tell a TA how you created required time delays. Write a calculation in the lab notebook that estimates the worst case error of the actual time lag compared to the equation in the lab description.
6. Your TA may ask you to explain how sections of the C code or circuitry you developed for this exercise work. To do this, you will need to understand the entire system.
7. Also play all 3 modes of the game (all 3 team members) and fill the results of the game in the table (shown below). You may want to modify the program to change the wait times for responses. You may reformat the table if you find a better way to display the results. Use your own judgment for how you want to present your data.
8. Capture the screen of SecureCRT or Putty showing the output of one of the sample rounds from both modes and take a print out. Attach this printout along with your code in the lab notebook.

		Final Score (after 5 tries)		
		Player 1 (Name: _____)	Player 2 (Name: _____)	Player 3 (Name: _____)
Mode 1				
Mode 2				
Mode 3				
Total (2+3-1)		[ ]	[ ]	[ ]

Writing Assignment - Design Report

You and your partners must submit a brief report for the Microprocessor-Controlled Game, according to the rubric for Lab 2 and loosely following the *Embedded Control Design Report format* on page 136. This report should cover the design, development and testing of the game system. *Quality* and *completeness* are the criteria for grading your reports.

Please note that the Report is in addition to your Lab Notebook - the Lab Notebook should still be kept up-to-date with regards to the work you did in this lab. Don't forget to refer to the guidelines in- *Writing Assignment Guidelines* on page 155

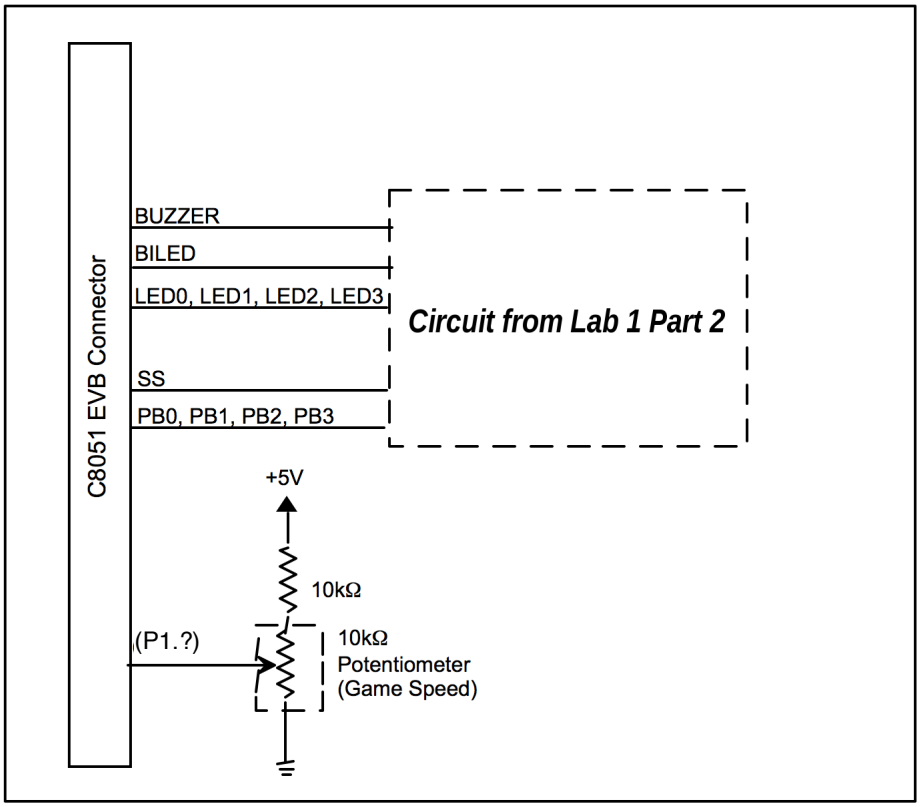

Figure 1 - Suggested hardware configuration for Lab 2 (Some components may not be necessary, others need to be added) Physical rearrangement of component positions on the protoboard may enhance the game experience

Sample C Program for Lab 2

```
/* This program demonstrates how to perform an A/D Conversion */
main()
{
    unsigned char result;

    Sys_Init();                         /* Initialize the C8051 board */
    Port_Init();                        /* Configure P1.4 for analog input */
    ADC_Init();                         /* Initialize A/D conversion */
    putchar(' ');

    while (1)
    {
        result = read_AD_input(4);      /* Read the A/D value on P1.4 */
    }
}


void Port_Init(void)
{
    P1MDIN &= ~0x10;                    /* Set P1.4 for analog input */
    P1MDOUT &= ~0x10;                   /* Set P1.4 to open drain */
    P1 |= 0x10;                         /* Send logic 1 to input pin P1.4 for impedance */
}
```

```
void ADC_Init(void)
{
    REF0CN = 0x03;                      /* Set Vref to use internal reference voltage (2.4
V) */
    ADC1CN = 0x80;                      /* Enable A/D converter (ADC1) */
    ADC1CF |= 0x01;                     /* Set A/D converter gain to 1 */
}

unsigned char read_AD_input(unsigned char n)
{
    AMX1SL = n;                         /* Set P1.n as the analog input for ADC1 */
    ADC1CN = ADC1CN & ~0x20;            /* Clear the "Conversion Completed" flag */
    ADC1CN = ADC1CN | 0x10;             /* Initiate A/D conversion */

    while ((ADC1CN & 0x20) == 0x00);    /* Wait for conversion to complete */

    return ADC1;                        /* Return digital value in ADC1 register */
}

//***************
// Draw a bar graph on terminal based on the unsigned char value passed from 0 to 255.
// Bar will be from 0 to 63 characters ('#') wide (length/4).
// An end marker '|' is printed at the end of the line.

void Draw_Bar(unsigned char length)
{
    char i;

    length = length/4;
    for(i=0; i<length; i++) putchar('#');   // print number of '#'
    length = 63 - length;                   // clear the rest of the line
    for(i=0; i<length; i++) putchar(' ');   // print (63 - length) spaces
    putchar('|');                           // print end mark '|'

    putchar('\r');                          // return to beginning of the line
}
```