Lab 2: A Microprocessor-Controlled Game – Hex⇔Bin

Preparation

Reading

Lab Manual:

Chapter 2 - Lab Equipment

Chapter 4 - The Silicon Labs C8051F020 and the EVB (sections relating to 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

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 and Pushbuttons as specified.

Software

1. The speed at which a new random number is generated will be a function of the digital value 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 next lab sequences—the *Smart Car & Gondola.* Portions of that lab sequence 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, and the use of interrupts to keep track of time.

Lab Description & Activities

Description of Game Objective

This game is called *LITEC Hex↔Bin*. A slide switch is used to determine which of the 2 distinct modes of the game will run after displaying instructions on the terminal. Each mode selects a random number between 0 and 7. The generated number represents a value that will either be displayed as a binary number in 3 LEDs (OFF = 0, ON = 1) or as a hex digit from 0 to 7 on the terminal. In the binary to hexadecimal mode the player must press the correct digit on the keyboard that is represented by the binary number in the LEDs. In the hexadecimal to binary mode the player must press the pushbuttons to display the equivalent binary value for the displayed hexadecimal digit. There will be a total of 4 pushbuttons. One of the 4 buttons represents the "Enter" pushbutton to let the program know when the player has set their binary value for the program to accept. The remaining 3 represent the bits. Pressing a bit pushbutton toggles the corresponding bit LED on or off. The goal of each mode is for the player to correctly match the random value as quickly as possible. A timer in the 8051 measures the response time and scoring depends on speed and accuracy. In the hex to binary mode, the allowed time, and hence the game speed, is controlled by the potentiometer voltage read by the 8051. A game consists of 8 tries of either mode. At the end of a game the program goes back and displays the original instructions asking the player which mode they desire and provides directions on how to proceed.

Description of Game Components

The components of game consists of 3 bit LEDs (LED2, LED1, and LED0), 4 pushbuttons (PBenter, PB2, PB1, and PB0), 1 bi-color LED (BLED), 1 slide switch (SS), 1 potentiometer, and 1 buzzer (BUZZ). The slide switch is used to select the game mode. The "Enter" pushbutton should be pressed to start a game and is used to indicate when a value is ready in the hex to binary mode. The other 3 pushbuttons are used to set (turn ON) or clear (turn OFF) the corresponding bit LEDs by toggling the value each time it is pressed. Once the player is ready the "Enter" button should be pressed. If the wrong value is entered, in either mode, the bi-color LED turns red, otherwise it turns green. The potentiometer is used to determine the speed of game. The buzzer will be used to indicate the start and end of each game.

Description of Game Play:

At the start of the game,

1. All 3 LEDs and the BLED should be off
2. BUZZER should be off
3. A message providing instructions is displayed on the terminal that include how to select a mode (slide switch), a wait time (potentiometer), and to press the "Enter" pushbutton when ready to begin the game.

As soon as the "Enter" pushbutton is pressed, the selected mode starts. The game speed is determined immediately by determining the voltage at the common leg of the potentiometer. This voltage is converted to a *wait_time*, which is the maximum time the program waits for a response before giving 0 points for an answer. The value *wait_time* should be calculated in the approximate number of overflows for Timer 0 in an interval from 0.5 seconds to 5.0 seconds.

wait_time = [(A/D conversion result from the potentiometer) * X+ Y] overflows.

The time will be approximately 0.5 s < *wait_time* **(in overflows)** < 5.0 s. The voltage across the potentiometer varies from 0 V to 2.4 V. A voltage input of 0 V from the potentiometer corresponds to ~0.5 s and a voltage input of 2.4 V corresponds to ~5.0 s. Make sure you set *wait_time* to the number of overflows in the corresponding time period. To summarize the process before the game starts, your program should

1. Print instructions on the terminal.
2. Wait for PBenter to be pressed
3. Read the voltage input from the potentiometer and perform A/D conversion to determine a digital equivalent value (0 to 255).
4. Calculate *wait_time* based on the digital value of the voltage and display the value.
5. Read the slide switch and run the corresponding mode selected.

After the game speed is determining and the program switches to the desired mode, each mode should display a message on the terminal explaining how that mode of the game is played. After playing 8 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 (Binary to Hexadecimal)

```
Display brief instructions
Zero the game score
Turn off all the LEDs and sound the buzzer for ~0.5 second
For 8 tries:
    Generate a random number from 0 to 7
    If bit 0 of the random number is 1, LED0 is lit
    If bit 1 of the random number is 1, LED1 is lit
    If bit 2 of the random number is 1, LED2 is lit
    Clear the timer overflow_count value
    Wait for a keyboard press [input = getchar(); see hw1.c]
    Save the overflow_count value
    If the key matches the random number, BLED is green, otherwise it is red
    Calculate the point score based on the following example:
        If overflow_count > wait_time then points = 0
            Otherwise points = 10 – (10 * overflow_count)/wait_time
        Ex) wait_time = 1000, overflow_count = 650, points = 10 – (10 * 650)/1000 = 4
        (Integer arithmetic truncates 6.5 to 6 so total is 4.)
    Display the points for the try and total score on the terminal
    Delay for 0.5 second
    Turn off the BLED
End of the 8 tries loop
Display the final score and sound the buzzer for ~0.5 second
```

Mode 2 (Hexadecimal to Binary)

```
Display brief instructions
Zero the game score
Turn off all the LEDs and sound the buzzer for ~0.5 second
For 8 tries:
    Generate a random number from 0 to 7
    Display Hex digit on the terminal
    Clear the timer overflow_count value
    While overflow_count < wait_time
        If PB0 is pressed LED0 = !LED0
        If PB1 is pressed LED1 = !LED1
        If PB2 is pressed LED2 = !LED2
    End while
    Calculate binary value from LED2, LED1, LED0
    If it matched the random hex value BLED is green, otherwise it is red
    Points = 10 if correct, otherwise 0
    Display the points for the try and total score on the terminal
    Delay for 0.5 second
    Turn off the BLED
End of the 8 tries loop
Display the final score and sound the buzzer for ~0.5 second
```

Each player gets 8 tries during their turn. At the end of the game, the program should 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: All pushbuttons must be debounced for proper game operation. Timer 0 should be in 16-bit mode.

Game Enhancements

You are encouraged to add additional functionality to your game for extra credits. You could add any additional hardware components or modify the software program to create a more complex, fun game as you wish. For software enhancement, you could make the game speed vary if the player is consistently correct, or keep increasing the speed again for high score players. For hardware enhancement, you could add an additional LED to introduce more complex patterns. You are encouraged to be creative. Talk to your grading TA if you are not sure what to do for game enhancement.

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, and they shouldn't be too closely packed.

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 pin 3:

     ```
     REF0CN=0x03;
     ```

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

     ```
     ADC1CF |=0x01
     ```

   - 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.

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 the *wait_time*. 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 both modes of the game (all 3 players) with different *wait_times* and fill the results of the game in the table (shown below) for 2 to 3 rounds using different values of *wait_time*. You may reformat the table if you find a better way to display the results. Use you own judgment for how you want to present your data.
8. Capture the screen of SecureCRT or Hyperterminal 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.

	wait_time	Final Score (out of 8 tries)		
		Player 1 (Name: ______________)	Player 2 (Name: ___________)	Player 3 (Name: ___________)
Mode 1				
Mode 1				
Mode 2				
Mode 2				
Total		[ ]	[ ]	[ ]

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

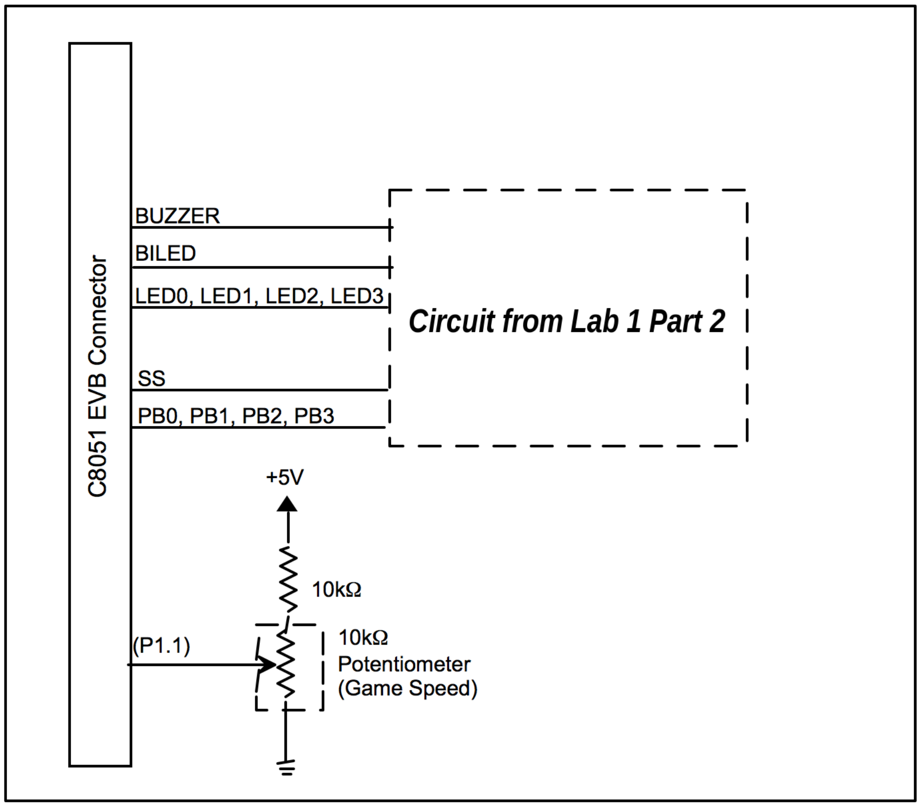

Figure 1 - Suggested hardware configuration for Lab 2 (Some components may not be necessary) Physical rearrangement of component positions on the protoboard may enhance the game experience

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 P1.0 for analog input */
    ADC_Init();                         /* Initialize A/D conversion */

    while (1)
    {
        result = read_AD_input(0);      /* Read the A/D value on P1.0 */
    }
}

void Port_Init(void)
{
    P1MDIN &= ~0x01;                    /* Set P1.0 for analog input */
    P1MDOUT &= ~0x01;                   /* Set P1.0 to open drain */
    P1 |= 0x01;                         /* Send logic 1 to input pin P1.0 */
}

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 */
}
```