}

Lab 1(part 2): Going Further - Timer Overflow Interrupts

Preparation

Reading

Lab Manual:

Chapter 2 - Lab Equipment

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

C language reference concepts:

data types, declarations, variables, variable scope, symbolic constants, functions

Embedded Control Multimedia Tutorials

Hardware: Multimeter

Objectives

General

1. Introduction to timer overflow interrupts.
2. Further familiarization with the hardware and software aspects of digital input and output.

Hardware

1. Familiarization with the use of the multimeter as a voltmeter.
2. Configuration of push buttons to produce a 0 or +5 volt digital output which will serve as an input to the C8051.
3. Addition of a push button to control various digital outputs for the game lab (Lab 2).

Software

1. Further refinement of skills requiring bit manipulation.
2. Keeping track of time using timer overflow interrupts.
3. Use of a random number to determine the color of bi-color LED.

Motivation

In the previous lab, you explored the use of *digital* inputs and outputs. You will continue the use of *digital* input/output, implementing another output LED. The laboratory introduces the use of Timers to measure specific periods of time. Additionally, Interrupts are used to monitor specific events, timer overflows in this lab, and allow the code to act based on those events. This laboratory provides further foundation for *Lab 2: Guitar Hero*.

Lab Description & Activities

In this program, the Slide Switch (SS) represents an ‘on/off’ toggle to control the game. Turning the Slide Switch ‘on’ starts the game. When SS is in the ‘off’ position, the game pauses and will resume once the Slide Switch is turned back ‘on’. Note, during a pause, the timer needs to be disabled. The game itself involves reacting to the random lighting of LEDs. Two LEDs are used, with three possible states

1. Only LED0 is lit
2. Only LED1 is lit
3. Both LED0 and LED1 are lit.

Two pushbuttons are used as inputs to the game. The player reacts by pushing the pushbuttons associated with each LED.

– If LED0 is lit, the player pushes PB0

– If LED1 is lit, the player pushes PB1

– If both LED0 and LED1 are lit, the player pushes both PB0 and PB1

The game uses a random number generator to decide whether to light LED0, LED1 or both. Program code is implemented to generate a random selection of the numbers 0, 1 and 2.

– If the random number is 0, light LED0

– If the random number is 1, light LED1

– If the random number is 2, light both LED0 and LED1

The LEDs are lit for one second. At the end of that one second, the inputs are checked and the program stores whether a correct or incorrect response was made. Then, your program will generate a new random number. To avoid confusion, if the new random number is same as the previous random number, disregard it and continue generating numbers until a different number is generated. Thus, the same LED pattern is not repeated on successive turns.

Furthermore, the BILED is used to indicate correct and incorrect inputs from the pushbuttons. If the player inputs correctly, the BILED should be lit green. If the player inputs incorrectly, the BILED should be lit red.

Your game should consist of 10 turns. After completing the game, your program should print out the number of correct responses. The game is restarted by moving the slide switch to the ‘off’ position and then back to the ‘on’ position.

Hardware

You will need to add one LED to your circuit, with the appropriate wiring and buffer chip connections. *Figure E.2* shows the hardware configuration for this lab. For convenience, the push buttons (PB0 and PB1) should be mounted side by side right below the corresponding LEDs. The buzzer will not be used in this laboratory.

Remember

Do not disassemble the circuitry you built for Lab 1 - it will be used in Lab 2.

Software

You will need to write a C program to perform as described above. Timer overflows are configured as a 16-bit counter using System Clock. Included at the end of this assignment is sample C code providing an foundation to get started on Lab 1 part 2. The code includes functions to initialize the Timer and Interrupt registers. A function that generates random numbers is also included. Some of the code references code that you developed in Lab 1 part 2. Use the sample code to refine the Lab 1 part 1 code for this lab. Another program demonstrates a method for generating a random number in *C*.

The sample code has been provided as a c-file and is available on LMS in Section 4 of course materials.

Lab Check-Off: Demonstration and Verification

1. Complete the entries in your lab notebook as described below and present it to your TA.
2. Run your C program and demonstrate that it performs as stated above.
3. Explain to the TA how Timer 0 and interrupts are used in your code.
4. Your TA may ask you to explain how other sections of the C code or circuitry you developed for this exercise work. To do this, you will need to understand the entire system.

Writing Assignment - Lab Notebook

Be sure to keep your Lab Notebook current by recording you and your lab partner's work and progress for every lab. Don't forget to refer to the guidelines in *Appendix B- Writing Assignment Guidelines 139*.

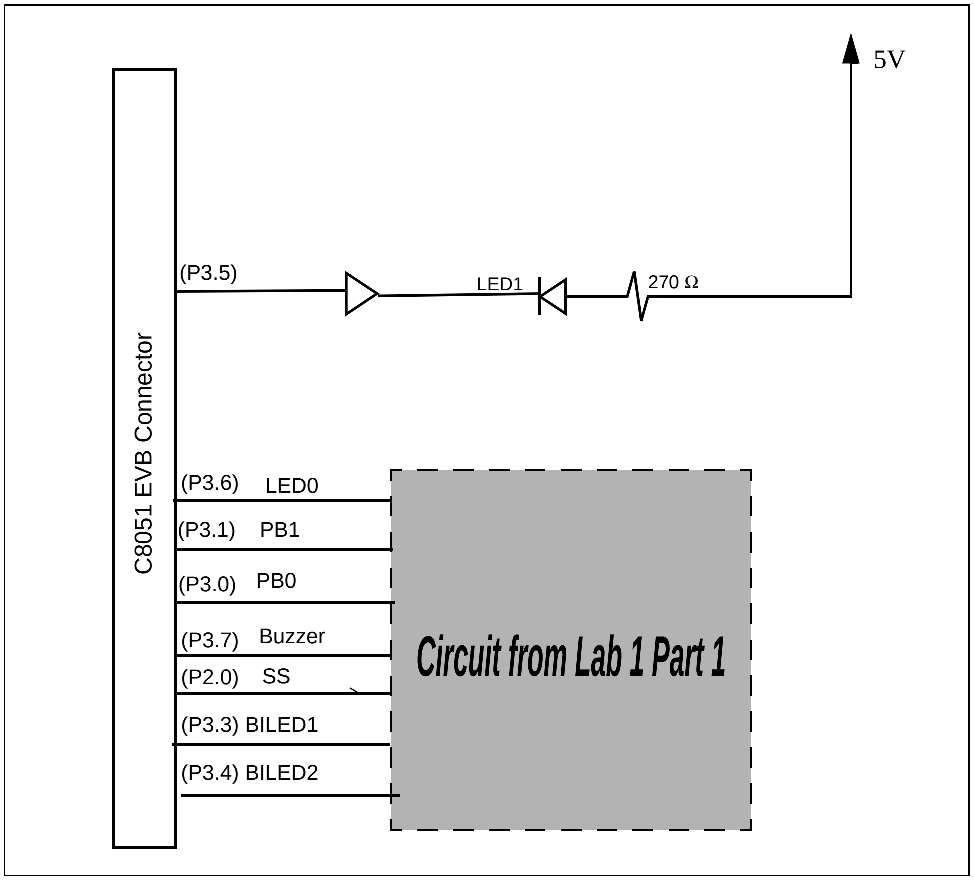

Figure C.3 - Suggested hardware configuration for Lab 1 (part 2)

C Programs for Lab 1 part 2

```
/*This program demonstrates the use of T0 interrupt. The code will count the number of
T0 timer overflows that occur while a slide switch is in ON position*/

#include <c8051_SDCC.h>// include files. This file is available online
#include <stdio.h>
#include <stdlib.h>

//-----------------------------------------------------------------------------
// Function PROTOTYPES
//-----------------------------------------------------------------------------
void Port_Init(void);// Initialize ports for input and output
void Timer_Init(void);// Initialize Timer 0
void Interrupt_Init(void); //Initialize interrupts
void Timer0_ISR (void) interrupt 1;
unsigned char random(void);

//-----------------------------------------------------------------------------
// Global Variables
//-----------------------------------------------------------------------------
sbit at 0xB3 BILED1;
sbit at 0xB4 BILED2;
sbit at 0xB1 PB1;
sbit at 0xA0 SS;
```

```
// sbit settings are incomplete, include those developed in Lab 1.1 and
// add the sbit setting for LED1
unsigned int Counts = 0;


//***************
void main(void)
{
   Sys_Init();// System Initialization
   Port_Init();// Initialize ports 2 and 3
   Interrupt_Init();
   Timer_Init();// Initialize Timer 0

   putchar(' '); //
   printf("Start\r\n");

   while (1) /* the following loop prints the number of overflows that occur
                while the push button is pressed, the BILED is lit while the
                button is pressed */
   {
         BILED1 = 0;// Turn OFF the BILED
         BILED2 = 0;

         while( SS ) // while SS0 is ON (high)
         TR0 = 1;   // Timer 0 enabled

         while (PB1);// wait until PB1 is pressed
         Counts = 0;

         while (!PB1);// wait until PB1 is released

         printf("Number of Overflows = %d\n", Counts);
         BILED1 = 1;// Turn ON the BILED
         BILED2 = 0;

         TR0 = 0;  // Timer 0 disabled

   }
}

//***************
void Port_Init(void)
{
   // use Port configuration from Lab 1.1
   // adding the output bit for LED1

}

void Interrupt_Init(void)
{
   IE |= ____;// enable Timer0 Interrupt request
   EA = ____;// enable global interrupts
}
//***************
void Timer_Init(void)
{

   CKCON |= _____; // Timer0 uses SYSCLK as source
   TMOD &= _____;// clear the 4 least significant bits
   TMOD |= _____; // Timer0 in mode 1
   TR0 = 0;// Stop Timer0
   TL0 = 0; // Clear low byte of register T0
   TH0 = 0;// Clear high byte of register T0
}



//***************
void Timer0_ISR(void) interrupt 1
{
// add interrupt code here, in this lab, the code will increment the
//    global variable ‘counts’
}

/*************************************************************************************/

/* This function demonstrates how to obtain a random integer between 0 and 1 in C, You
may modify and use this code to get a random integer between 0 and N.
*/


/*return a random integer number between 0 and 1*/
unsigned char random(void)
{
   return (rand()%2); /*rand returns a random number between 0 and 32767*/
                    /*the mod command (%) returns the remainder of dividing this */
                    /*value by 2 and returns the result, a value of either 0 or 1*/
}
```