Microprocessor Systems
Lab 2: Interrupt & Timer ISRs

Nick Choi Samuel Deslandes

9/26/16

1 Introduction

The overall goal of this lab is to become familiar with configuring timers with interrupts on the 8051 and utilizing interrupt service routines (ISR) to perform operations based upon the timers interrupts.

The lab is divided into three sections. In the first section, a C program is created to react to an external interrupt 0 generated whenever a pushbutton on the protoboard is pressed. In the second section, two additional C programs are written in order to display elapsed time measured in 0.1 seconds. Each program uses different timer configurations and methods of measuring a tenth of a second; one program will round its calculations in order to approximate 0.1 seconds, while the second program will be exact, with no need for rounding or approximations. In the third section, the programs from the first two sections of the lab are combined to create a reaction based game which records how long it takes a user to respond to prompts which are randomly flashed on the ANSI terminal. The user has the option to reset the terminal every five reactions and an enhancement was added so that the color of the text in the ANSI terminal changes depending on the relative speed of the user's reaction time.

2 Methods

2.1 Software

The code for parts 1, 2 and 3 can be found in the appendix below. All code was uploaded and run on the 8051 through the programming/debugging USB port.

2.1.1 Part 1

The C program for the first section of the lab was a straightforward application of using an external interrupt source, such as the grounding of a pushbutton, to generate an interrupt which would cause text to be displayed on the terminal. External interrupt 0 (/INT0) was used as the interrupt source for this lab; to configure the 8051 for this, interrupts must first be globally enabled by setting bit 7 of the "Interrupt Enable" SFR (IE) as well as bit 0 of the same SFR to enable /INT0. These are bit addressable addresses which correspond to "EA" and "EX0" respectively. The operation mode of /INT0 can then be set to be active logic low triggered or falling edge triggered, by clearing or setting bit 0 of the "Timer Control" SFR (TCON), which is bit addressable as "IT0". In this lab IT0 was set to be triggered by a negative falling edge (IT0 = 1) because it was not desirable to have multiple interrupts generated if the user holds the pushbutton down.

In order to interface the interrupt to the pushbutton, the crossbar must also be configured to route /INT0 to a port pin. This can be done by setting bit 2 of the "XBR1" SFR (XBR1 = 0x04). For the crossbar settings used in this section, /INT0 was routed to pin 2 on port 0, which must be configured as an input. This is done by using the "P0MDOUT" SFR to set P0.2 in open-drain mode, then by using the "P0" SFR to set P0.2 to high impedance mode.

When /INT0 is triggered the program's current operation is preempted by the ISR associated with the interrupt generated. The instructions that take place in the ISR should

be limited to only a small number of fast operations. Rather than executing a lengthy I/O operation such as "printf()" here, a global variable is used as a flag which allows the rest of the program to determine whether an interrupt has occurred. All the ISR has to do in this case is set the flag. When declaring the ISR function, it is important to remember to include the interrupt's priority; /INT0 has a priority level of 0 (the highest priority).

The main function for this section is simple. Before entering the infinite loop, the variable used as the interrupt flag is cleared. In the loop, the program checks if the flag has been set by the ISR and if it has, the desired text is printed to the display and the flag is cleared.

2.1.2 Part 2

The code for section 2 involved utilizing timer interrupts to display elapsed time in multiples of a tenth of a second. This was done using two methods: An inaccurate method using rounding, and an exact method (with no rounding). Both methods operate using the same concepts. Timer0 is used to count from a starting value until it overflows, triggering the timer0 overflow interrupt. In the ISR, the timer is set to its starting value and a global variable used to count the number of overflows is incremented. Since overflows happen at a fixed frequency, by counting the number of overflows, the elapsed time may be measured. For example, in the case of the accurate timer, an overflow happens once every 50 ms. A tenth of a second is therefore equivalent to 2 timer overflows.

Interrupt configuration was performed similarly to section 1, one difference being that instead of setting the bit addressable address "EX0", "ET0" is now set. This enables timer0 interrupts rather than /INT0.

For the inaccurate method timer0 was configured as a 16bit counter with a starting value of 0, using SYSCLK/12 as a base. For this method, SYSCLK used the external oscillator for a frequency of 22.1184 MHz. The calculations for how many overflows correspond to a tenth of a second were as follows:

$$= \frac{22.1184 \times 10^6 \,\text{counts}}{12 \,\text{sec}} * \left(\frac{2^{16} - 1 \,\text{counts}}{\text{overflow}} \right)^{-1} \tag{1}$$

$$= \frac{1\,843\,200 \,\text{counts}}{\text{sec}} * \left(\frac{1 \,\text{overflow}}{65\,535 \,\text{counts}} \right) \tag{2}$$

$$= \frac{28.125 \,\text{overflows}}{\text{sec}} = \frac{2.8125 \,\text{overflow}}{0.1 \,\text{sec}} \tag{3}$$

Since the number of overflows must be an integer value, the 2.8125 was rounded up to 3.

As mentioned above, each time an overflow happened, the ISR incremented an overflow counting variable. In the infinite loop of the main function, whenever this counter had a value of 3 the overflow counter would be reset to 0, a value counting the number of tenths of seconds elapsed would be incremented, and the elapsed time would be displayed. Since a tenth of a second is represented as a floating point data type, "printf_fastf()" had to be used instead of the usual "printf()" function.

For the accurate timing method, timer0 was also configured as a 16bit counter, but had a starting value of 13,696 or 0x3580, and used SYSCLK/48 as a base. For this method

SYSCLK used the external oscillator and the phase-locked loop (PLL) which multiplies its source frequency by a programmable factor. This resulted in a SYSCLK frequency of $22.1184\,\text{MHz} * \left(\frac{9}{4}\right) = 49.7664\,\text{MHz}$. The calculations for how many overflows correspond to a tenth of a second and how to determine the timer's starting value were as follows:

$$\frac{49.7664 \times 10^6\,\text{counts}}{48\,\text{sec}} = \frac{1\,036\,800\,\text{counts}}{\text{sec}} \tag{4}$$

This represents the timer's counting speed. From this, the number of counts necessary for one overflow to happen in a tenth of a second can be calculated.

$$\frac{1\,036\,800\,\text{counts}}{\text{sec}} * \frac{1}{x}\frac{\text{overflow}}{\text{counts}} = \frac{1\,\text{overflow}}{0.1\,\text{sec}} \tag{5}$$

$$x = 103\,680\,\text{counts} \tag{6}$$

This value, however, is too large to store in a 16bit variable. To remedy this, it was halved, requiring $\frac{103680}{2} = 51\,840\,\text{counts/overflow}$ and a tenth of a second required two overflows. Since 51 840 counts are needed per overflow, the starting value of the timer should be $(2^{16} - 1) - 51840 = 13696$, or 0x3580 in hex.

2.1.3 Part 3

This section combined the code from section 1 and the accurate timer of section 2 to make a reaction time game. The external interrupt, port, and timer configurations were exactly as listed in the respective sections.

The game routine takes place in the infinite loop of the main function. First, a random number is generated using "rand()". This number represents the delay in tenths of seconds until the player is sent the signal to press the button. It is modded by 10 to ensure that the player does not wait more than a second between button presses. It is at this point that the timer is started. An empty while loop is used to delay the program by the appropriate number of tenths of seconds determined by rand(). The message "PRESS NOW" is then sent to the terminal to alert the player that the button may be pressed. The timer overflow counting variable is reset and the program waits for the player to press the button using another empty while loop. When the button is pressed, a button press flag is set in the /INT0 ISR, and the program moves on from the while loop.

Next, the elapsed time is calculated using the timer0 overflow counting variable. This value is printed to the terminal with an overall average of the player's response time. In order to accomplish this, the number of rounds the user has played is stored and incremented when the user pushes the button. Likewise, the sum of the player's reaction times are stored in another variable and is updated every time the player presses the button. In order to give additional feedback to the player, the text is displayed in 3 colors, green, yellow, and red, depending on how quickly the user presses the button. If the player reacts within 0.2 s, the text will be green. If pressed within 0.5 s, the text will be yellow. Otherwise, the text will be red. Once the feedback has been printed, the timer is stopped, its value is reset to the

starting value, and the timer0 overflow counting variable, as well as the button press flag, are cleared.

Additionally, every 5 rounds the program will pause to ask the user if they wish to keep playing. Using "getchar()", the user's input can be read. If the user presses 'y' on the keyboard, the game continues. If the user presses 'n', the game is reset and the program restarts. The program will not move on unless one of the two keys is pressed.

2.2 Hardware

Sections 1 and 3 of this lab did were identical in terms of hardware. Section 2 required no hardware other than a serial-to-USB adapter in order to interface with the terminal.

In sections 1 and 3, the pushbutton included on the breadboard was used. It was wired such that the button would be grounded when pressed, with the output connected to pin 18 on the EVB, corresponding to P0.2 of the 8051. A pull-up resistor was used to prevent false external interrupts.

3 Results

By completing section one of the lab, a functioning C program was produced to react to the press of a pushbutton wired on the protoboard. After completing section two of the lab, two new programs were developed to configure timers on the 8051 to display the elapsed time in tenths of a second using an accurate method and an inaccurate method. The final deliverable was a reaction based game which displayed the time that it took the user to press a pushbutton in response to ANSI terminal prompts. The program allowed the user to reset the terminal every 5 responses. To further enhance the final deliverable, the program was modified so that the text color of the ANSI terminal changed between red, yellow and green based upon the reaction time.

4 Conclusion

The end results of the lab matched with the initial goals however there were numerous instances where the system did not behave as expected or calculations needed to be repeated. The initial intention for the labs enhancement was to add an additional button to pause the game. However, due to the prioritization structure of the 8051s external interrupts, this feature did not work as planned with the code that was written. Thus, a design trade-off was made to not massively alter the code structure and to design a different enhancement. This enhancement resulted in the varying text colors of the ANSI terminal's output. This was achieved by incorporating several conditional statements, in conjunction with ANSI escape sequences, into the pre-existing code structure.

If more time was given to complete this lab assignment, additional conditional statements could be added so that the program ignores premature button presses. This would improve the robustness of the code and would make the game much fairer for all users. Additionally, if the second pushbutton could be integrated into the program, the user would not have to respond to a terminal prompt every five rounds.

5 Appendices

5.1 Modified putget.h

```
//-----------------------------------------------------------------------------
// putget.h
//-----------------------------------------------------------------------------
// Title:                Microcontroller Development: putchar() & getchar() functions.
// Author:               Dan Burke
// Date Created:         03.25.2006
// Date Last Modified:   03.25.2006
//
// Description:          http://chaokhun.kmitl.ac.th/~kswichit/easy1/easy1_3.html
//
//
// Target:               C8051F120
// Tool Chain:           KEIL C51

//-----------------------------------------------------------------------------
// putchar()
//-----------------------------------------------------------------------------
void putchar(char c)
{
    while(!TI0);
    TI0=0;
    SBUF0 = c;
}

//-----------------------------------------------------------------------------
// getchar()
//-----------------------------------------------------------------------------
char getchar(void)
{
    char c;
    while(!RI0);
    RI0 =0;
    c = SBUF0;
// Echoing the get character back to the terminal is not normally part of getchar()
//    putchar(c);    // echo to terminal
    return SBUF0;
}
```

5.2 Circuit Schematic for sections 1 and 3

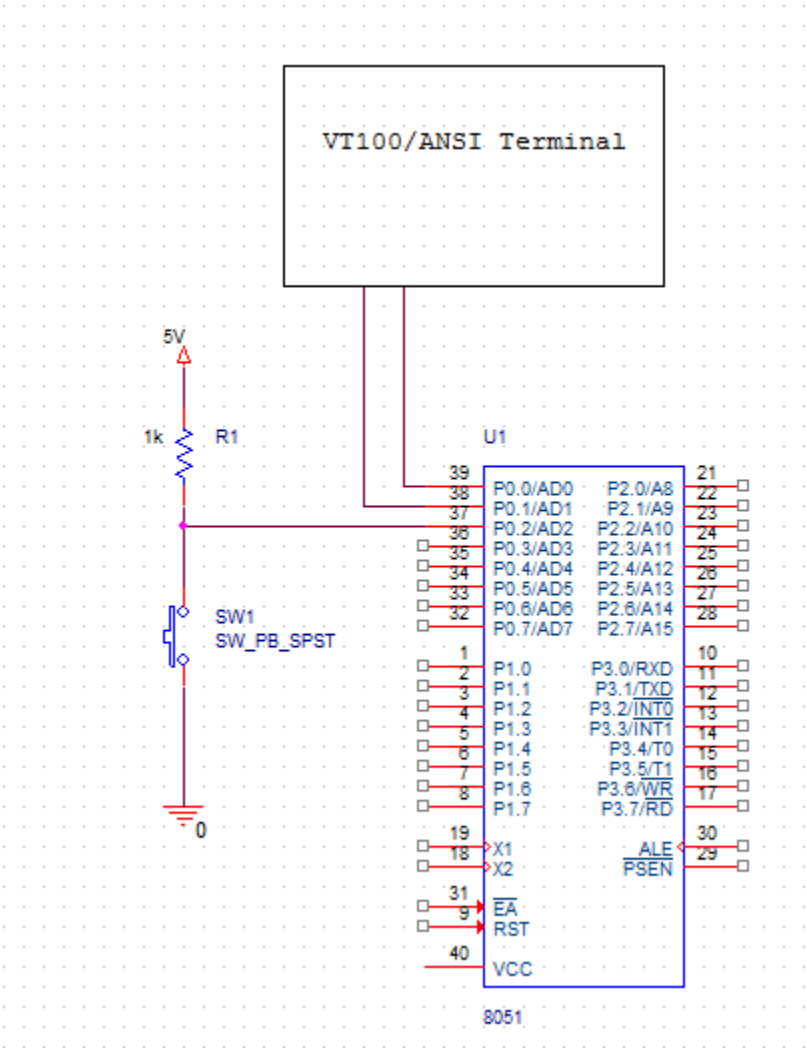

Figure 1: Circuit schematic for parts 1 and 3

5.3 Part 1

5.3.1 Code

```
//-----------------------------------------------------------------------
// Includes
//-----------------------------------------------------------------------
#include <c8051f120.h>
#include <stdio.h>
#include "putget.h"

//-----------------------------------------------------------------------
// Global CONSTANTS
//-----------------------------------------------------------------------

#define EXTCLK      22118400    // External oscillator frequency in Hz
#define SYSCLK      49766400    // Output of PLL derived from (EXTCLK * 9/4)
#define BAUDRATE    115200      // UART baud rate in bps
```

```
char butpress;

//-----------------------------------------------------------------
// Function PROTOTYPES
//-----------------------------------------------------------------
void main(void);
void PORT_INIT(void);
void SYSCLK_INIT(void);
void UART0_INIT(void);
void SW2_ISR (void) __interrupt 0;

//-----------------------------------------------------------------
// Main Function
//-----------------------------------------------------------------

void main(void){
  SFRPAGE = CONFIG_PAGE;

  PORT_INIT();
  SYSCLK_INIT();
  UART0_INIT();

  SFRPAGE = LEGACY_PAGE;
  IT0 = 1;      // /INT0 triggered on negative falling edge

  printf("\033[2J");
  printf("MPS Interrupt Switch Test \n\n\r");
  printf("Ground /INT0 on P0.2 to generate an interrupt. \n\n\r");

  SFRPAGE = CONFIG_PAGE;
  EX0 = 1;    // Enable external interrupts

  SFRPAGE = UART0_PAGE;

  butpress = 0; //  clear button flag
  while(1){
    if(butpress){ // if button flag is set
      printf("/INT0 grounded! \n\n\r");
      butpress = 0;
    }
  }
}
//-----------------------------------------
// Interrupts
//-----------------------------------------

void SW2_ISR (void) __interrupt 0{
  butpress = 1;  // set button flag
}

//-----------------------------------------------------------------
// PORT_Init
//-----------------------------------------------------------------
// Configure the Crossbar and GPIO ports

void PORT_INIT(void){
  char SFRPAGE_SAVE;

  SFRPAGE_SAVE = SFRPAGE;      // Save Current SFR page.

  SFRPAGE = CONFIG_PAGE;
  WDTCN    = 0xDE;             // Disable watchdog timer.
  WDTCN    = 0xAD;
  EA       = 1;                // Enable interrupts as selected.

  XBR0     = 0x04;             // Enable UART0.
  XBR1     = 0x04;             // /INT0 routed to port pin.
  XBR2     = 0x40;             // Enable Crossbar and weak pull-ups.
```

```
  P0MDOUT = 0x01;             // P0.0 (TX0) is configured as Push-Pull for output
  // P0.1 (RX0) is configure as Open-Drain input.
  // P0.2 (pushbutton) is configured as Open_Drain for input.
  P0      = 0x06;             // Additionally, set P0.0=0, P0.1=1, and P0.2=1.
  SFRPAGE = SFRPAGE_SAVE;     // Restore SFR page.
}

//-----------------------------------------------------------
// SYSCLK_Init
//-----------------------------------------------------------

// Initialize the system clock

void SYSCLK_INIT(void){
  int i;

  char SFRPAGE_SAVE;

  SFRPAGE_SAVE = SFRPAGE;     // Save Current SFR page.

  SFRPAGE = CONFIG_PAGE;
  OSCXCN  = 0x67;             // Start external oscillator
  for(i=0; i < 256; i++);     // Wait for the oscillator to start up.
  while(!(OSCXCN & 0x80));    // Check to see if the Crystal Oscillator Valid Flag is set.
  CLKSEL  = 0x01;             // SYSCLK derived from the External Oscillator circuit.
  OSCICN  = 0x00;             // Disable the internal oscillator.

  SFRPAGE = CONFIG_PAGE;    // Configure PLL
  PLL0CN  = 0x04;
  SFRPAGE = LEGACY_PAGE;
  FLSCL   = 0x10;
  SFRPAGE = CONFIG_PAGE;
  PLL0CN |= 0x01;
  PLL0DIV = 0x04;
  PLL0FLT = 0x01;
  PLL0MUL = 0x09;
  for(i=0; i < 256; i++);
  PLL0CN |= 0x02;
  while(!(PLL0CN & 0x10));
  CLKSEL  = 0x02;             // SYSCLK derived from the PLL.

  SFRPAGE = SFRPAGE_SAVE;     // Restore SFR page.
}

//-----------------------------------------------------------
// UART0_Init
//-----------------------------------------------------------

// Configure the UART0 using Timer1, for <baudrate> and 8-N-1.

void UART0_INIT(void){
  char SFRPAGE_SAVE;

  SFRPAGE_SAVE = SFRPAGE;     // Save Current SFR page.

  SFRPAGE = TIMER01_PAGE;
  TMOD   &= ~0xF0;
  TMOD   |=  0x20;            // Timer1, Mode 2: 8-bit counter/timer with auto-reload.
  TH1     = (unsigned char)-(SYSCLK/BAUDRATE/16); // Set Timer1 reload value for baudrate
  CKCON  |= 0x10;       // Timer1 uses SYSCLK as time base.
  TL1     = TH1;
  TR1     = 1;                // Start Timer1.

  SFRPAGE = UART0_PAGE;
  SCON0   = 0x50;             // Set Mode 1: 8-Bit UART
  SSTA0   = 0x10;             // UART0 baud rate divide-by-two disabled (SMOD0 = 1).
  TI0     = 1;                // Indicate TX0 ready.
```

```
  SFRPAGE = SFRPAGE_SAVE;      // Restore SFR page
}
```

5.4 Part 2

5.4.1 Inaccurate timer code

```
//-----------------------------------------------------------------
// Includes
//-----------------------------------------------------------------
#include <c8051f120.h>
#include <stdio.h>
#include "putget.h"

//-----------------------------------------------------------------------
// Global CONSTANTS
//-----------------------------------------------------------------------

#define EXTCLK       22118400    // External oscillator frequency in Hz
#define SYSCLK       49766400    // Output of PLL derived from (EXTCLK * 9/4)
#define BAUDRATE     115200      // UART baud rate in bps
char timer0_flag = 0;
//-----------------------------------------------------------------
// Function PROTOTYPES
//-----------------------------------------------------------------
void main(void);
void PORT_INIT(void);
void SYSCLK_INIT(void);
void UART0_INIT(void);
void TIMER0_INIT(void);
void TIMER0_ISR(void) __interrupt 1;

//-----------------------------------------------------------------
// Main Function
//-----------------------------------------------------------------

void main(void){
  unsigned int tenths = 0;

  SFRPAGE = CONFIG_PAGE;

  PORT_INIT();
  TIMER0_INIT();
  SYSCLK_INIT();
  UART0_INIT();

  SFRPAGE = LEGACY_PAGE;
  IT0 = 1;     // /INT0 triggered on negative falling edge

  printf("\033[2J");
  printf("MPS Interrupt Timer Test \n\n\r");

  SFRPAGE = CONFIG_PAGE;
  EX0 = 1;     // Enable external interrupt

  SFRPAGE = UART0_PAGE;


  while(1){
    if(timer0_flag == 3){ // Wait for 3 overflows, print elapsed time and reset flag
      tenths+=1;
      printf_fastf("Elapsed Time: %.2f\n\r", tenths*0.1);
      timer0_flag = 0;
    }
  }
}
//-----------------------------------------------------------------
```

```
// Interrupts
//-----------------------------------

void TIMER0_ISR(void) __interrupt 1{
  // Reset timer0 value and increment flag
  TH0 = 0x00;
  TL0 = 0x00;
  timer0_flag += 1;
}

//-------------------------------------------------------
// PORT_Init
//-------------------------------------------------------
// Configure the Crossbar and GPIO ports

void PORT_INIT(void){
  char SFRPAGE_SAVE;

  SFRPAGE_SAVE = SFRPAGE;      // Save Current SFR page.

  SFRPAGE = CONFIG_PAGE;
  WDTCN    = 0xDE;             // Disable watchdog timer.
  WDTCN    = 0xAD;
  EA       = 1;                // Enable interrupts as selected.
  XBR0     = 0x04;             // Enable UART0.
  XBR1     = 0x04;             // /INT0 routed to port pin.
  XBR2     = 0x40;             // Enable Crossbar and weak pull-ups.
  P0MDOUT = 0x01;              // P0.0 (TX0) is configured as Push-Pull for output
  SFRPAGE = SFRPAGE_SAVE;      // Restore SFR page.
}

//-------------------------------------------------
// SYSCLK_Init
//-------------------------------------------------

// Initialize the system clock 22.1184Mhz

void SYSCLK_INIT(void){
  int i;

  char SFRPAGE_SAVE;

  SFRPAGE_SAVE = SFRPAGE;      // Save Current SFR page.

  SFRPAGE = CONFIG_PAGE;
  OSCXCN  = 0x67;              // Start external oscillator
  for(i=0; i < 256; i++);      // Wait for the oscillator to start up.
  while(!(OSCXCN & 0x80));     // Check to see if the Crystal Oscillator Valid Flag is set.
  CLKSEL  = 0x01;              // SYSCLK derived from the External Oscillator circuit.
  OSCICN  = 0x00;              // Disable the internal oscillator.

  CLKSEL  = 0x01;              // SYSCLK derived from external oscillator.

  SFRPAGE = SFRPAGE_SAVE;      // Restore SFR page.
}

//-----------------------------------------------------
// UART0_Init
//-----------------------------------------------------

// Configure the UART0 using Timer1, for <baudrate> and 8-N-1.

void UART0_INIT(void){
  char SFRPAGE_SAVE;

  SFRPAGE_SAVE = SFRPAGE;      // Save Current SFR page.

  SFRPAGE = TIMER01_PAGE;
```

```
  TMOD   &= ~0xF0;
  TMOD   |=  0x20;              // Timer1, Mode 2: 8-bit counter/timer with auto-reload.
  TH1     = (unsigned char)-(EXTCLK/BAUDRATE/16); // Set Timer1 reload value for baudrate
  CKCON  |= 0x10;          // Timer1 uses SYSCLK as time base.
  TL1     = TH1;
  TR1     = 1;                  // Start Timer1.

  SFRPAGE = UART0_PAGE;
  SCON0   = 0x50;               // Set Mode 1: 8-Bit UART
  SSTA0   = 0x10;               // UART0 baud rate divide-by-two disabled (SMOD0 = 1).
  TI0     = 1;                  // Indicate TX0 ready.

  SFRPAGE = SFRPAGE_SAVE;       // Restore SFR page
}

// Timer init
void TIMER0_INIT(void){
  char SFRPAGE_SAVE;
  SFRPAGE_SAVE = SFRPAGE;

  SFRPAGE = TIMER01_PAGE;

  TMOD &= 0xF0;           // Timer0, Mode 1: 16-bit counter/timer.
  TMOD |= 0x01;
  TH0 = 0x00;           // Set high byte to 0
  CKCON &= ~0x0B;         // Timer0 uses SYSCLK/12 as base
  TL0 = 0x00;           // Set low byte to 0
  TR0 = 1;            // Start timer0

  SFRPAGE = CONFIG_PAGE;
  ET0 = 1;            // Enable timer0 interrupt

  SFRPAGE = SFRPAGE_SAVE;
}
```

5.4.2 Accurate timer code

```
//-----------------------------------------------------------------
// Includes
//-----------------------------------------------------------------
#include <c8051f120.h>
#include <stdio.h>
#include "putget.h"

//-----------------------------------------------------------------
// Global CONSTANTS
//-----------------------------------------------------------------

#define EXTCLK       22118400    // External oscillator frequency in Hz
#define SYSCLK       49766400    // Output of PLL derived from (EXTCLK * 9/4)
#define BAUDRATE     115200      // UART baud rate in bps
char timer0_flag = 0;
//-----------------------------------------------------------------
// Function PROTOTYPES
//-----------------------------------------------------------------
void main(void);
void PORT_INIT(void);
void SYSCLK_INIT(void);
void UART0_INIT(void);
void TIMER0_INIT(void);
void TIMER0_ISR(void) __interrupt 1;

//-----------------------------------------------------------------
// Main Function
//-----------------------------------------------------------------

void main(void){
```

```
  __bit restart = 0;
  unsigned int tenths = 0;

  SFRPAGE = CONFIG_PAGE;

  PORT_INIT();
  TIMER0_INIT();
  SYSCLK_INIT();
  UART0_INIT();

  printf("\033[2J");
  printf("MPS Interrupt Switch Test \n\n\r");
  printf("Ground /INT0 on P0.2 to generate an interrupt. \n\n\r");

  SFRPAGE = CONFIG_PAGE;
  EX0 = 1;

  SFRPAGE = UART0_PAGE;


  while(1){
    if(timer0_flag == 2){  // Wait for 2 overflows, print elapsed time and reset flag
      tenths+=1;
      printf("Elapsed Time: %u\n\r", tenths);
      timer0_flag = 0;
    }
  }
}
//-------------------------------------
// Interrupts
//-------------------------------------

void TIMER0_ISR(void) __interrupt 1{  // reset timer to 0x3580 and increment flag
  TH0 = 0x35;
  TL0 = 0x80;
  timer0_flag += 1;
}

//-------------------------------------------------------
// PORT_Init
//-------------------------------------------------------
// Configure the Crossbar and GPIO ports

void PORT_INIT(void){
  char SFRPAGE_SAVE;

  SFRPAGE_SAVE = SFRPAGE;      // Save Current SFR page.

  SFRPAGE = CONFIG_PAGE;
  WDTCN    = 0xDE;             // Disable watchdog timer.
  WDTCN    = 0xAD;
  EA       = 1;                // Enable interrupts as selected.
  XBR0     = 0x04;             // Enable UART0.
  XBR1     = 0x04;             // /INT0 routed to port pin.
  XBR2     = 0x40;             // Enable Crossbar and weak pull-ups.
  P0MDOUT = 0x01;              // P0.0 (TX0) is configured as Push-Pull for output
  SFRPAGE = SFRPAGE_SAVE;      // Restore SFR page.
}

//-------------------------------------------------
// SYSCLK_Init
//-------------------------------------------------

void SYSCLK_INIT(void){
  int i;

  char SFRPAGE_SAVE;
```

```
  SFRPAGE_SAVE = SFRPAGE;      // Save Current SFR page.

  SFRPAGE = CONFIG_PAGE;
  OSCXCN  = 0x67;              // Start external oscillator
  for(i=0; i < 256; i++);      // Wait for the oscillator to start up.
  while(!(OSCXCN & 0x80));     // Check to see if the Crystal Oscillator Valid Flag is set.
  CLKSEL  = 0x01;              // SYSCLK derived from the External Oscillator circuit.
  OSCICN  = 0x00;              // Disable the internal oscillator.

  SFRPAGE = CONFIG_PAGE;       // Set PLL to multiply external oscillator by (9/4)
  PLL0CN  = 0x04;
  SFRPAGE = LEGACY_PAGE;
  FLSCL   = 0x10;
  SFRPAGE = CONFIG_PAGE;
  PLL0CN |= 0x01;
  PLL0DIV = 0x04;
  PLL0FLT = 0x01;
  PLL0MUL = 0x09;
  for(i=0; i < 256; i++);
  PLL0CN |= 0x02;
  while(!(PLL0CN & 0x10));
  CLKSEL  = 0x02;              // SYSCLK derived from the PLL.

  SFRPAGE = SFRPAGE_SAVE;      // Restore SFR page.
}

//-----------------------------------------------------------------
// UART0_Init
//-----------------------------------------------------------------

// Configure the UART0 using Timer1, for <baudrate> and 8-N-1.

void UART0_INIT(void){
  char SFRPAGE_SAVE;

  SFRPAGE_SAVE = SFRPAGE;      // Save Current SFR page.

  SFRPAGE = TIMER01_PAGE;
  TMOD   &= ~0xF0;
  TMOD   |=  0x20;             // Timer1, Mode 2: 8-bit counter/timer with auto-reload.
  TH1     = (unsigned char)-(SYSCLK/BAUDRATE/16); // Set Timer1 reload value for baudrate
  CKCON  |= 0x10;        // Timer1 uses SYSCLK as time base.
  TL1     = TH1;
  TR1     = 1;                 // Start Timer1.

  SFRPAGE = UART0_PAGE;
  SCON0   = 0x50;              // Set Mode 1: 8-Bit UART
  SSTA0   = 0x10;              // UART0 baud rate divide-by-two disabled (SMOD0 = 1).
  TI0     = 1;                 // Indicate TX0 ready.

  SFRPAGE = SFRPAGE_SAVE;      // Restore SFR page
}

// Timer init
void TIMER0_INIT(void){
  char SFRPAGE_SAVE;
  SFRPAGE_SAVE = SFRPAGE;

  SFRPAGE = TIMER01_PAGE;

  TMOD &= 0xF0;        // Timer0, Mode 1: 16-bit counter/timer.
  TMOD |= 0x01;
  TH0 = 0x35;          // Set high byte such that timer0 starts at 0x3580
  CKCON &= ~0x09;
  CKCON |= 0x02;         // Timer0 uses SYSCLK/48 as base
  TL0 = 0x80;          // Set high byte such that timer0 starts at 0x3580
  TR0 = 1;           // Start timer0
```

```
  SFRPAGE = CONFIG_PAGE;
  ET0 = 1;          // Enable timer0 interrupt

  SFRPAGE = SFRPAGE_SAVE;
}
```

5.5 Part 3

5.5.1 Code

```
//-----------------------------------------------------------------
// Includes
//-----------------------------------------------------------------
#include <c8051f120.h>
#include <stdio.h>
#include <stdlib.h>
//#include <time.h>
#include "putget.h"

//-----------------------------------------------------------------
// Global CONSTANTS
//-----------------------------------------------------------------

#define EXTCLK      22118400    // External oscillator frequency in Hz
#define SYSCLK      49766400    // Output of PLL derived from (EXTCLK * 9/4)
#define BAUDRATE    115200      // UART baud rate in bps
char timer0_flag = 0;
__bit reactPress = 0;
char react_flag;

//-----------------------------------------------------------------
// Function PROTOTYPES
//-----------------------------------------------------------------
void main(void);
void PORT_INIT(void);
void SYSCLK_INIT(void);
void UART0_INIT(void);
void TIMER0_INIT(void);
void TIMER0_ISR(void) __interrupt 1;
void reactPress_ISR (void) __interrupt 0;

//-----------------------------------------------------------------
// Main Function
//-----------------------------------------------------------------

void main(void){
  // Declare local variables
  char choice;
  unsigned int rand_;
  unsigned char tenths = 0;
  float reactions = 0;
  unsigned char trials = 0;
  SFRPAGE = CONFIG_PAGE;

  PORT_INIT();
  TIMER0_INIT();
  SYSCLK_INIT();
  UART0_INIT();

  SFRPAGE = LEGACY_PAGE;
  IT0 = 1;      // /INT0 triggered on negative falling edge

  // Display the set up information
  printf("\033[2J");
  printf("MPS Reaction Game \n\n\r");
  printf("Ground /INT0 on P0.2 to generate an interrupt. \n\n\r");
```

```
  SFRPAGE = CONFIG_PAGE;
  EX0 = 1;

  SFRPAGE = UART0_PAGE;
  // Seed the random number generator
  srand(78);
  while(1){
    //Generate random number
    rand_ = rand()%10;
    TR0 = 1;                  //Start Timer0
    // Wait for the random delay to elapse
    while(timer0_flag/2 != rand_){

    }
    // Tell user to press the button and start keeping track of reaction time
    printf("PRESS NOW\n\n\r");
    timer0_flag = 0;
    // Wait for the user to press the reaction button
    while(!react_flag){

    }
    // Determine how long their response took in tenths of a second (truncates)
    tenths = timer0_flag/2;
    // Increment the number of trials and add the reaction time to the running total
    trials += 1;
    reactions += tenths*.1;
    // If they respond in under .2s the output text is green
    if(tenths < 2){
      printf("\033[1;32m");
    }
    // If they respond in under .5s the output text is yellow
    else if(tenths < 5){
      printf("\033[1;33m");
    }
    // Otherwise, the output text is red
    else{
      printf("\033[1;31m");
    }
    // Display the user's response time and average response time
    printf_fast_f("Your response time was: %.2f seconds\n\r", tenths*0.1);
    printf_fast_f("Your average response time is: %.2f\n\n\r", reactions/trials);
    printf("\033[1;37m");

    // Provide the user with the option to reset the program every 5 trials
    if(trials % 5 == 0){
      printf("Do you want to continue? Press Y or N\n\n\r");
      while(1){
        choice = getchar();
        if (choice == 'n'){
          return;
        } else if(choice == 'y'){
          break;
        }
      }
    }
    // Reset the variables
    TR0 = 0;
    TH0 = 0x35;
    TL0 = 0x80;
    timer0_flag = 0;
    react_flag = 0;
  }
}
//-------------------------------------
// Interrupts
//-------------------------------------

void TIMER0_ISR(void) __interrupt 1{  // reset timer to 0x3580 and increment flag
```

```
  TH0 = 0x35;
  TL0 = 0x80;
  timer0_flag += 1;
}

// Set the flag for the reaction button if it is pressed. Uses software debouncing
void reactPress_ISR (void) __interrupt 0{
  if(timer0_flag > 0){
    react_flag = 1;
  }
}

//-----------------------------------------------------------------
// PORT_Init
//-----------------------------------------------------------------
// Configure the Crossbar and GPIO ports

void PORT_INIT(void){
  char SFRPAGE_SAVE;

  SFRPAGE_SAVE = SFRPAGE;      // Save Current SFR page.

  SFRPAGE = CONFIG_PAGE;
  WDTCN   = 0xDE;              // Disable watchdog timer.
  WDTCN   = 0xAD;
  EA      = 1;                 // Enable interrupts as selected.
  XBR0    = 0x04;              // Enable UART0.
  XBR1    = 0x14;              // /INT0 and /INT1 routed to port pins P0.2 and P0.3
      respectively.
  XBR2    = 0x40;              // Enable Crossbar and weak pull-ups.
  P0MDOUT = 0x01;              // P0.0 (TX0) is configured as Push-Pull for output
  // P0.1 (RX0) is configure as Open-Drain input.
  // P0.2 (recatPress button through jumper wire) is configured as Open_Drain for input.
  // P0.3 (recatPress button through jumper wire) is configured as Open_Drain for input.
  P0      = 0x0E;              // Additionally, set P0.0=0, P0.1=1, P0.2=1, and P0.3=1
  SFRPAGE = SFRPAGE_SAVE;      // Restore SFR page.
}

//-----------------------------------------------------------------
// SYSCLK_Init
//-----------------------------------------------------------------

// Initialize the system clock 22.1184Mhz

void SYSCLK_INIT(void){
  int i;

  char SFRPAGE_SAVE;

  SFRPAGE_SAVE = SFRPAGE;      // Save Current SFR page.

  SFRPAGE = CONFIG_PAGE;
  OSCXCN  = 0x67;              // Start external oscillator
  for(i=0; i < 256; i++);      // Wait for the oscillator to start up.
  while(!(OSCXCN & 0x80));     // Check to see if the Crystal Oscillator Valid Flag is set.
  CLKSEL  = 0x01;              // SYSCLK derived from the External Oscillator circuit.
  OSCICN  = 0x00;              // Disable the internal oscillator.

  SFRPAGE = CONFIG_PAGE;       // Configure PLL
  PLL0CN  = 0x04;
  SFRPAGE = LEGACY_PAGE;
  FLSCL   = 0x10;
  SFRPAGE = CONFIG_PAGE;
  PLL0CN |= 0x01;
  PLL0DIV = 0x04;
  PLL0FLT = 0x01;
  PLL0MUL = 0x09;
  for(i=0; i < 256; i++);
```

```
  PLL0CN |= 0x02;
  while(!(PLL0CN & 0x10));
  CLKSEL  = 0x02;              // SYSCLK derived from the PLL.

  SFRPAGE = SFRPAGE_SAVE;      // Restore SFR page.
}

//-------------------------------------------------------------
// UART0_Init
//-------------------------------------------------------------

// Configure the UART0 using Timer1, for <baudrate> and 8-N-1.

void UART0_INIT(void){
  char SFRPAGE_SAVE;

  SFRPAGE_SAVE = SFRPAGE;      // Save Current SFR page.

  SFRPAGE = TIMER01_PAGE;
  TMOD   &= ~0xF0;
  TMOD   |=  0x20;             // Timer1, Mode 2: 8-bit counter/timer with auto-reload.
  TH1     = (unsigned char)-(SYSCLK/BAUDRATE/16); // Set Timer1 reload value for baudrate
  CKCON  |= 0x10;        // Timer1 uses SYSCLK as time base.
  TL1     = TH1;
  TR1     = 1;                 // Start Timer1.

  SFRPAGE = UART0_PAGE;
  SCON0   = 0x50;              // Set Mode 1: 8-Bit UART
  SSTA0   = 0x10;              // UART0 baud rate divide-by-two disabled (SMOD0 = 1).
  TI0     = 1;                 // Indicate TX0 ready.

  SFRPAGE = SFRPAGE_SAVE;      // Restore SFR page
}

// Timer init
void TIMER0_INIT(void){
  char SFRPAGE_SAVE;
  SFRPAGE_SAVE = SFRPAGE;

  SFRPAGE = TIMER01_PAGE;

  TMOD &= 0xF0;         // Timer0, Mode 1: 16-bit counter/timer.
  TMOD |= 0x01;
  TH0 = 0x35;           // Set high byte such that timer0 starts at 0x3580
  CKCON &= ~0x09;
  CKCON |= 0x02;          // Timer0 uses SYSCLK/48 as base
  TL0 = 0x80;           // Set high byte such that timer0 starts at 0x3580

  SFRPAGE = CONFIG_PAGE;
  ET0 = 1;            // Enable timer0 interrupt

  SFRPAGE = SFRPAGE_SAVE;
}
```

6 References

"MPS Lab 2," in RPI ECSE Department, 2016. [Online]. Available: `http://www.rpi.edu/dept/ecse/mps/MPS_Lab_Ex2-Intrpt.pdf`. Accessed: Sep. 22, 2016.

"C8051 Manual," in RPI ECSE Department, 1.4 ed., 2005. [Online]. Available: `https://www.ecse.rpi.edu/courses/CStudio/Silabs/C8051F12x-13x.pdf`. Accessed: Sep. 22, 2016.