Microprocessor Systems
Lab 1: IDE & ANSI Display

Nick Choi Samuel Deslandes

8/26/16

1 Introduction

The overall goal of this lab is to become familiar with utilizing the registers on the 8051 as well as performing fundamental I/O operations with it. This lab also served as an introduction to the VT100 Terminal and the utilization of its ANSI escape sequences.

This lab was divided into 3 parts. The first was an exercise in basic terminal I/O in which a C program was written to await a user keystroke and, if printable, output it onto the terminal display. The second part was an enhancement of the first part with heavy usage of ANSI escape codes to format text on the display. The ANSI escape codes are special sequences beginning with <ESC>(\033 in octal or $1B in hex) which modify terminal behavior. This includes features such as underlined text, background/foreground color changes, changing the cursor position, selectable scrolling, and more. The third part of the lab involved configuring ports on the 8051 to be either inputs or outputs. In this part of the lab, hardware connected to the input port could be used to change the state of hardware connected to the output ports of the 8051.

2 Methods

2.1 Software

The code for parts 1, 2 and 3 can be found in Appendix A, B and C respectively. 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 then cause text to be displayed on the terminal. External interrupt 0 (/INT0) was used as a the interrupt source for this lab; In order to configure the 8051 for this, interrupts must first be globally enabled by setting bit 7 of the "Interrupt Enable" SRF (IE) as well as bit 0 of the same SRF 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 no desirable to have multiple interrupts be 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" SRF (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 interrupt service routine (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 a 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 exactly accurate method. 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. In 2 overflows a tenth of a second has elapsed.

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 happed, 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 were 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 per overflow 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 2 overflows in a tenth of a second. 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

The code for section 3 had a different application than the previous two sections and was more hardware focused. In terms of software, all of port 1 had to be configured to be in open-drain mode (input), and all of port 2 to push-pull mode (output). This was done by setting all of the bits of the P1MDOUT special function register (SFR) low, and all of those of P2MDOUT high. This should be done in the 'PORT_INIT()' function. In the main function, all that had to be done was read the value of port 1 into port 2. This could be done using the P1 and P2 port SFRs.

2.2 Hardware

Parts 1 and 2 of this lab did not require any hardware other than a serial-to-USB adapter in order to interface with the terminal.

In the third section of the lab, the pins on port 1 were connected to the input device, the potentiometer module, and pins on port 2 were connected to the output device, the LED module. Since there were only 4 potentiometers on the potentiometer module only the first 4 pins on each port were used (P1.0 - P1.3 and P2.0 - P2.3). These corresponded to pins 12, 13, 10, and 11 on the EVB for port 1, and 29, 30, 27, and 28 for port 2. A circuit diagram can be found in the appendices, section 5.4.1.

The input pins of the LED module have inverters incorporated into them so that whenever a logic high is applied to an input pin, the signal is changed to be a logic low. This inversion allows the LED to illuminate because there will be a voltage difference across the anode and cathode of the LED. If a logic low is applied to the input pin, the signal is changed to be a logic high. This removes the voltage difference between the anode and cathode of the LED and thus prevents the LED from lighting up.

3 Results

By completing section one of the lab, a functioning C program was produced which read keyboard input and displayed it in an ANSI terminal. After completing section two, an improved version of the program from section one was produced. This version reacted to user input and manipulated the output of the ANSI terminal in several ways. The final deliverable was an LED module which was controlled by a potentiometer module.

4 Conclusion

The end results of this lab ultimately matched with the initial goals however there were numerous instances where our systems expected behavior did not correspond to its actual behavior. In order to properly produce the results that we needed for this lab, various hardware and software debugging techniques were utilized in order to verify the performance of each section of the lab. With some of these testing techniques, it was determined that the inputs to the 8051 utilize Schmitt triggers in order to prevent oscillating logic highs and logic lows. The use of Schmitt triggers makes the inputs of the 8051 more resilient to noise because they create separate thresholds for high and low values rather thancontent... having one shared boundary value.

If more time was given to complete this lab assignment, additional conditional statements could be added to further enhance the responses of the ANSI terminal to user input. These statements could incorporate more escape sequences to display new error messages, to integrate hardware components to the error messages or to further manipulate the text formatting in the ANSI terminal.

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 Part 1

5.2.1 Circuit Schematic

Figure 1: Circuit schematic for part 1

5.2.2 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 through jumper wire) 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;
  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
}
```

5.3 Part 2

5.3.1 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
      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
  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 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
//#define BAUDRATE  19200       // 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();

  SFRPAGE = LEGACY_PAGE;
  IT0 = 1;

  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){
      tenths+=1;
      printf("Elapsed Time: %u\n\r", tenths);
      timer0_flag = 0;
    }
  }
}
//------------------------------------------------------------
// Interrupts
//------------------------------------------------------------

void TIMER0_ISR(void) __interrupt 1{
  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 Circuit Schematic

Figure 2: Circuit schematic for part 3

5.5.2 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
//#define BAUDRATE  19200       // UART baud rate in bps
char timer0_flag = 0;
__bit reactPress = 0;
__bit resetPress = 0;
char react_flag;
//char reset_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;
//void resetPress_ISR (void) __interrupt 2;

//-----------------------------------------------------------------------------
// 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;

// 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 = 0;
    TL0 = 0;
    timer0_flag = 0;
    react_flag = 0;
  }
}
//----------------------------------------
// Interrupts
//----------------------------------------

void TIMER0_ISR(void) __interrupt 1{
  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;
  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;
  TMOD |= 0x01;
  TH0 = 0x35;
  CKCON &= ~0x09;
  CKCON |= 0x02;
  TL0 = 0x80;

  SFRPAGE = CONFIG_PAGE;
  ET0 = 1;

  SFRPAGE = SFRPAGE_SAVE;
}
```

6 References

"MPS Lab 1," in RPI ECSE Department, 2016. [Online]. Available: http://www.rpi.edu/dept/ecse/mps/MPS_Lab_Ex1-IDE_ANSI.pdf. Accessed: Sep. 17, 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. 17, 2016.