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.

In the first section of the lab, a C program was written so that user input from a keyboard could be displayed in the ANSI terminal. The 8051 determined which keyboard character was pressed using the "getchar()" and "putchar()" functions defined in the "putget.h" header file. It should be noted that the "getchar()" function was modified from the original header file to not echo captured keystrokes. The modified version can be found in section 5.1 below. Whenever a printable character was input, it was printed to the terminal display with the following message: "The keyboard character is *." Unprintable characters were not displayed and the program would await the user's next input. Since the program runs in an infinite loop, pressing the <ESC>key both restarted the C program and cleared the terminal. In order to determine whether the input was printable or not, the captured character's ASCII was compared with the printable characters on the ASCII table, which range from 0x20 (space) through 0x7E (~).

The second section enhances the C program written for the first by adding various text effects and formatting within the VT100 terminal. All of these features were produced by utilizing ANSI escape sequences; Table **??** in section 5.3.1 contains all of the ANSI escape sequences used for this lab.

The first modification was to change the background color to blue and the foreground color to yellow. This was done using the codes "\033[1;43m" and "\033[1;33m" respectively. The first line of text contains program termination information similar to that of part 1; It is printed on line 2 of the terminal and is horizontally centered. This can be accomplished using the escape code for changing the cursor position "\033[{row};{col}H". In this case 2 and 30 were used for the row and column respectively. As was done in part 1, the keyboard

response must also be displayed, this time on line 6 with the keyed in character printed in white. The same code above was used to move the cursor to line 6, and "\033[1;37m" was used to change the text's color to white.

If the keyed in character was not printable, the message "The keyboard character $XX is 'not printable'." would appear starting halfway down the terminal (line 12 in our case) and work its way down the screen. This text should be blinking and make an audible 'BEL' noise when a non-printable character is keyed in.

The terminal should be set such that only the bottom half of the screen scrolls. Although the escape code to change the position of the cursor could be used accomplish this, the codes to save and restore the cursors position were used instead so that we would not have to directly keep track of what line to jump to. In order to do this, even before a non-printable character was keyed in, the position of the next error message was already logged and ready to be restored and written to whenever an error message needed to be output. The error messages were terminated with the ASCII newline escape sequence ('\n') to move the cursor down, and this new position was saved for printing the next error. When the error messages reach the bottom of the terminal, only the lines included in the selective scrolling area would scroll. The ANSI codes for saving and restoring the cursor's position are "\033[s" and "\033[u" respectively. In order to set a selective scrolling area the code "\033[{start},{stop}r" must be used. Only the lines between {start} and {stop} will scroll. To get the audible feedback portion the ASCII escape sequence '\a' was used. When writing the code block pertaining to printing the error message, it is important to remember to turn off the underscore and blinking once the desired text with these effects has been printed.

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 than 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

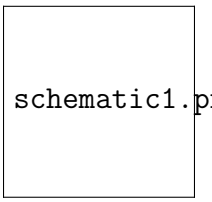

Figure 1: Circuit schematic for part 1

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
//#define BAUDRATE  19200       // UART baud rate in bps
//__bit SW2press = 0;
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){
  __bit restart = 0;

  SFRPAGE = CONFIG_PAGE;
```

```
  PORT_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;

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

void SW2_ISR (void) __interrupt 0{
  butpress = 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
  // P0.1 (RX0) is configure as Open-Drain input.
  // P0.2 (SW2 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
//#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 == 3){
      tenths+=1;
      printf("Elapsed Time: %u\n\r", tenths);
      timer0_flag = 0;
    }
  }
}
//-------------------------------------
// Interrupts
//-------------------------------------

void TIMER0_ISR(void) __interrupt 1{
  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 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)-(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;
  TMOD |= 0x01;
  TH0 = 0x00;
  CKCON &= ~0x0B;
  TL0 = 0x00;
  TR0 = 1;

  SFRPAGE = CONFIG_PAGE;
  ET0 = 1;

  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
//-----------------------------------------------------------------

// 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;
  TR0 = 1;

  SFRPAGE = CONFIG_PAGE;
  ET0 = 1;

  SFRPAGE = SFRPAGE_SAVE;
```

}

5.5 Part 3

5.5.1 Circuit Schematic

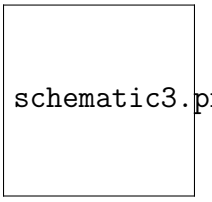

Figure 2: Circuit schematic for part 3

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){
  unsigned int rand_;
  unsigned char tenths = 0;
  unsigned int reactions = 0;
  unsigned char trials = 0;
  SFRPAGE = CONFIG_PAGE;

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

  SFRPAGE = LEGACY_PAGE;
```

```
  IT0 = 1;
  IT1 = 1;

  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;

  reactPress = 0;
  srand(78);
  while(1){
    //generate random number
    rand_ = rand()%10;
    printf("%i\n\r", rand_);
    TR0 = 1; //Start Timer0
    while(timer0_flag/2 != rand_){

    }
    printf("PRESS NOW");
    timer0_flag = 0;
    while(!react_flag){

    }
    tenths = timer0_flag/2;
    trials += 1;
    reactions += 1;
    printf("Your response time was: %u tenths\n\n\r", tenths);
    printf("Your average response time is: %f\n\n\r", (float)reactions/trials)
        ;



    TR0 = 0;
    TH0 = 0;
    TL0 = 0;
    timer0_flag = 0;
    react_flag = 0;
    if(reset_flag){
      reset_flag = 0;
      return;
    }
  }
}
//-----------------------------------
// Interrupts
//-----------------------------------

void TIMER0_ISR(void) __interrupt 1{
  TH0 = 0x35;
  TL0 = 0x80;
```

```
  timer0_flag += 1;
}

void reactPress_ISR (void) __interrupt 0{
  react_flag = 1;
}

void resetPress_ISR (void) __interrupt 0{
  reset_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.