**Microprocessor Systems Lab 2**

Interrupt and Timer ISRs

Shane Conaboy

Justin Yost

ECSE-4790 Microprocessor Systems

September 28, 2015

**Introduction**

This lab investigated how to generate both external and internal interrupts on the 8051 microcontroller. For the external interrupt, a push button was wired to an input port and an interrupt would be thrown when it was pushed. Timer overflows were used to generate the internal interrupts. By changing the system clock settings to obtain different speed in conjunction with modifying the timer settings, a stopwatch-like program that was accurate to one-tenth of a second was created using internal interrupts. At the end, both parts of the lab were integrated to make a game based off of human reaction time.

**Procedure**

The first part of the lab involved wiring a pushbutton to trigger an external interrupt on the microcontroller. The pushbutton was wired to P0.2 since the UART was enabled, taking up pins P0.0 and P0.1. When the button was pressed, the program would set a global variable, *interrupt\_detected*, to 1. The I/O was handled in the *main* function, which would output to the terminal that the interrupt was detected. To handle debouncing, a capacitor was added in parallel to the output.

The next part of the lab was to modify the speed of the system clock and use it to generate timer overflows that would track time accurately to one-tenth of a second. The full calculations for this section are shown in Appendix A. The first attempt to generate the stopwatch was using Timer0 in 13-bit mode with a system clock of 4.1472 Mhz. To generate 0.1 seconds, Timer0 would need to overflow 50.625 times. As an approximation, 51 overflows was chosen, which gave an error of 0.375 overflows per overflow, or 0.375 seconds per second.

The second, more accurate stopwatch was created with Timer1 in 16-bit mode, a clock rate of 4.1472 MHz, and preloading Timer1 each overflow so it would not count its entire range of values. The preload value was 64512, which was set every interrupt, and 405 overflows at this preload would give exactly 0.1 seconds. The calculations can be found in Appendix A.

The last part of the lab was to create a game using the previous two parts. The game would tell the user to press the pushbutton when “GO!” is displayed after a random delay, and it would measure the reaction time of the player using the stopwatch program in part 2 to keep track of time and the external interrupt in program in part 1. The last five of the player’s attempts were displayed on the screen, along with the average.

**Results**

Part 1 of the lab was implemented with little problems. After the debounce circuit was added, the screen would display “Interrupt Detected” only once upon each press of the button. Part 2 of the lab was much trickier because of the clock settings.

Obtaining the inaccurate timer in Part 2 was straightforward since everything was already set up in the given code. To generate the accurate timer, first selecting the external crystal as the system clock was tried, however this caused problems with the baud rate. Next, changing the multiply and divide values of the PLL was attempted. Even with changing the expected frequency filter to the correct range, the PLL would never initialize properly and the program would hang. At this point, time was running short, so the method of preloading the timer was chosen. After performing the calculations, the values were set and the timer was able to achieve a 0.1 second accuracy.

Developing the game was a lot like lab 1 and mostly required text positioning. After computing the random delay, the timers would count up to that point before resetting, after which they would begin counting the reaction time of the player. The times were shifted through an array, and the average of the five stored values was computed every trial. As an add-on the lab, a penalty was implemented that would automatically give a 5 seconds to the current trial if the player pressed the button before the screen displayed “GO!” No major problems were encountered during the implementation of the game.

**Conclusion**

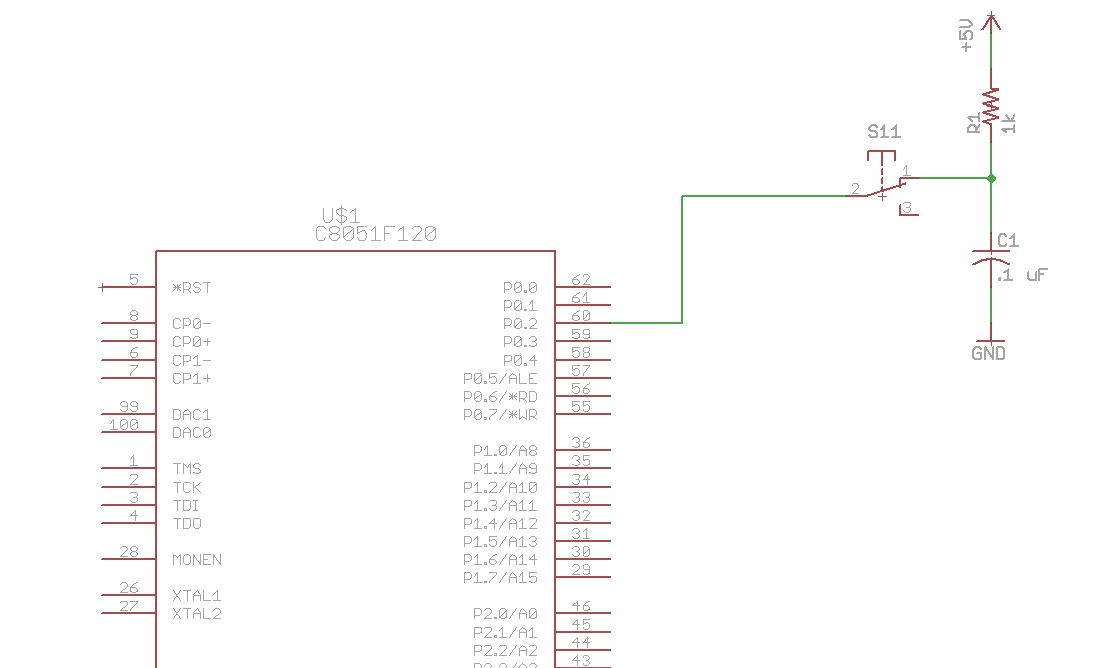
By creating a timing-based game using both external and internal interrupts, different interrupt, timer, and clock settings of the 8051 microcontroller were explored. The most time was spent on trying to get the clock settings correct so the program would run properly, and this part was never implemented in the way it was originally intended. Future suggestions would be to explore this area of the 8051 more to gain a better understanding. Otherwise, the lab was successfully completed.

**Appendix A: Timing Calculations**

Inaccurate clock:

Accurate Clock:

Timer1 in 16-bit mode 🡪 GCF of 4.1472E5 and 65536 is 1024

**Appendix B: Circuit Schematic**

**Appendix C: Code**

Note: Only the code for Part 3 is shown since it also demonstrates the use of the external interrupt from Part 1 and the timer interrupt from Part 2.

// IntrptEx.c

//

// 8051 Interrupt Example Program

//

// This program uses an interrupt to call the ISR handler

// function, SW2\_ISR(), when the /INT0 line is grounded.

// Each time the signal makes a low transition, an interrupt will be

// generated. If the line is held down, the SWR\_ISR()

// function will only be executed once, and not be called

// again unless the line is released, and grounded again.

// Additionally timer0 will be used to count each .1 seconds

// to be used as a game to test reaction time.

//

// /INT0 is configured to be on P0.2

// UART0 is used to communicate to the user through ProCOMM

//

// This code was written and tested using the SiLabs IDE

// and SDCC.

//

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

// Includes

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

#include <c8051f120.h> // SFR declarations.

#include <stdio.h> // Necessary for printf.

#include <putget.h> // Necessary for printf

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

// 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;

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

// Function PROTOTYPES

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

void main(void);

void PORT\_INIT(void);

void SYSCLK\_INIT(void);

void UART0\_INIT(void);

void timer0\_ISR (void) \_\_interrupt 1;

unsigned int interrupt\_count = 0;

char press = 0;

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

// MAIN Routine

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

void main (void)

{

\_\_bit restart = 0;

char character;

unsigned int delay1, delay2 = 0;

unsigned int randnum = 0;

unsigned int ones, tenths = 0;

unsigned char tenths\_seconds = 0;

unsigned int seconds = 0;

unsigned int delay;

unsigned int times[5];

unsigned int time;

unsigned char started = 0;

unsigned int avg;

char i;

char grace;

SFRPAGE = CONFIG\_PAGE;

PORT\_INIT(); // Configure the Crossbar and GPIO.

SYSCLK\_INIT(); // Initialize the oscillator.

UART0\_INIT(); // Initialize UART0.

SFRPAGE = LEGACY\_PAGE;

IT0 = 1; // /INT0 is edge triggered, falling-edge.

SFRPAGE = CONFIG\_PAGE;

EX0 = 1; // Enable Ext Int 0 only after everything is settled

SFRPAGE = UART0\_PAGE; // Direct output to UART0

printf("\033[2J"); // Erase screen and move cursor to the home posiiton.

printf("MPS Timer Game\n\n\r");

printf("Press button 8 ASAP on GO.\n\n\r");

SFRPAGE = UART0\_PAGE;

delay = rand() % 50 + 10;

while (1)

{

if (interrupt\_count == 405) { // Count each .1 seconds

tenths\_seconds = tenths\_seconds + 1;

interrupt\_count = 0;

if (tenths\_seconds % 10 == 0 && tenths\_seconds != 0) {

tenths\_seconds = 0;

seconds = seconds + 1;

if(seconds < 1 && !started) // Don't count quick double presses before the start of the next trial

grace = 1;

else

grace = 0;

}

}

if(seconds >= delay/10 && tenths\_seconds >= delay-delay/10\*10 && !started) // Flash GO and restart the timer

{

seconds = 0;

tenths\_seconds = 0;

started = 1;

printf("\033[16;26HGO");

printf("\033[5;0H");

}

if(press > 0 && !grace) // The user has pressed the button

{

printf("\033[16;26H ");

delay = rand() % 50 + 10; // Gererate a new random delay between 1 and 5 seconds.

press = 0; // Reset interrupt variable

if(started == 1)

{

time = seconds\*10 + tenths\_seconds; // Reaction time for this trial

}

else

{

time = 50; // False button press (GO not flashed on screen). Penalize 5 seconds

}

seconds = 0;

tenths\_seconds = 0;

started = 0;

// Shift 5 most recent times

times[0] = times[1];

times[1] = times[2];

times[2] = times[3];

times[3] = times[4];

times[4] = time;

avg = (times[0] + times[1] + times[2] + times[3] + times[4])/5; // Calculate running average of 5

printf("\033[6;0H"); //Move cursor position

for (i = 0; i <= 4; i++) {

printf("Trial %d: %d.%d seconds\r\n", (5-i), times[i]/10, times[i]-times[i]/10\*10);

}

printf("Avg: %d.%d seconds", avg/10, avg-avg/10\*10);

}

}

}

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

// Interrupt Service Routines

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

// NOTE: this is an example of what NOT to do in an interrupt handler. No I/O should be done

// in ISRs since I/O is very slow and the handler must execute very quickly.

//

// This routine stops Timer0 when the user presses SW2.

//

void timer0\_ISR (void) \_\_interrupt 1 // Interrupt 0 corresponds to vector address 0003h.

// the keyword "interrupt" defines this as an ISR and the number is determined by the

// Priority Order number in Table 11.4 in the 8051 reference manual.

{

TL0 = 64512;

TH0 = 64512 >> 8; // Restart counter to only count for 1024 clock ticks

interrupt\_count = interrupt\_count + 1;

}

void SW2\_ISR (void) \_\_interrupt 0 // Interrupt 0 corresponds to vector address 0003h.

// the keyword "interrupt" defines this as an ISR and the number is determined by the

// Priority Order number in Table 11.4 in the 8051 reference manual.

{

press = 1;

}

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

// PORT\_Init

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

//

// Configure the Crossbar and GPIO ports

//

void PORT\_INIT(void)

{

char 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.

// EX0 = 1; // Enable External Interrupt 0 when everything is settled.

SFRPAGE = SFRPAGE\_SAVE; // Restore SFR page.

}

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

// SYSCLK\_Init

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

//

// Initialize the system clock

//

void SYSCLK\_INIT(void)

{

int i;

char 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 Current SFR page.

SFRPAGE = UART0\_PAGE;

SCON0 = 0x50; // Set Mode 1: 8-Bit UART

SSTA0 = 0x10; // UART0 baud rate divide-by-two disabled (SMOD0 = 1).

SFRPAGE = TIMER01\_PAGE;

TMOD &= ~0xF0;

TMOD |= 0x20; // Timer1, Mode 2: 8-bit counter/timer with auto-reload.

TH1 = -(SYSCLK/BAUDRATE/16); // Set Timer1 reload value for baudrate

CKCON |= 0x10; // Timer1 uses SYSCLK as time base.

TL1 = TH1;

TR1 = 1; // Start Timer1.

TMOD &= 0xF0; // Setup timer0 for 16 bit mode

TMOD |= 0x08;

TMOD |= 0x01;

ET0 = 1;

TR0 = 1;

SFRPAGE = UART0\_PAGE;

TI0 = 1; // Indicate TX0 ready.

SFRPAGE = SFRPAGE\_SAVE; // Restore SFR page.

}