PHYS 319
Labs 3 and 4 Notes

Jonathan Chan (15354146)

February 3, 2018

To compile the C programs to .asm and .elf, I've modified my Makefile as below.

```
SOURCES = $(wildcard *.c)
EXEC    = $(patsubst %.c, %.elf, $(SOURCES))
DEVICE  = msp430g2553
INSTALL_DIR=$(HOME)/ti/msp430_gcc

GCC_DIR =  $(INSTALL_DIR)/bin
SUPPORT_FILE_DIRECTORY = $(INSTALL_DIR)/include

CC      = $(GCC_DIR)/msp430-elf-gcc
GDB     = $(GCC_DIR)/msp430-elf-gdb

#O0 works, O1 works, O2 doesn't -Os works
CFLAGS = -I $(SUPPORT_FILE_DIRECTORY) -mmcu=$(DEVICE) -Os -g
LFLAGS = -L $(SUPPORT_FILE_DIRECTORY) -T $(DEVICE).ld

all: prog1 prog2 adc pwm dimmer

prog1: prog1.c
    $(CC) $(CFLAGS) $(LFLAGS) $? -o prog1.elf
    $(CC) $(CFLAGS) $(LFLAGS) $? -S -o prog1.asm
prog2: prog2.c
    $(CC) $(CFLAGS) $(LFLAGS) $? -o prog2.elf
    $(CC) $(CFLAGS) $(LFLAGS) $? -S -o prog2.asm
adc:   adc.c
    $(CC) $(CFLAGS) $(LFLAGS) $? -o adc.elf
    $(CC) $(CFLAGS) $(LFLAGS) $? -S -o adc.asm
pwm:   pwm.c
    $(CC) $(CFLAGS) $(LFLAGS) $? -o pwm.elf
    $(CC) $(CFLAGS) $(LFLAGS) $? -S -o pwm.asm
dimmer:   dimmer.c
    $(CC) $(CFLAGS) $(LFLAGS) $? -o dimmer.elf
    $(CC) $(CFLAGS) $(LFLAGS) $? -S -o dimmer.asm

clean:
        rm prog1.elf prog1.asm prog2.elf prog2.asm adc.elf adc.asm \
                pwm.elf pwm.asm dimmer.elf dimmer.asm
```

1 Program 1

Compiling from C, the produced Assembly file has a *lot* of extra code, and appears to have been optimized differently. Below is the relevant section of the compiled .asm with some comments comparing lines to last lab's prog1.asm code.

```
.LCFI0:
        .loc 1 21 0
        MOV.W   #23168, &WDTCTL         ; turn off watchdog
        .loc 1 22 0
        MOV.B   #65, &P1DIR             ; set output direction (P1.6 and P1.0 for LEDs)
        .loc 1 23 0
        MOV.B   #1, &P1OUT              ; set initial state (LED1 on)
.LBB2:
        .loc 1 27 0
        MOV.W   #-5536, R12             ; amount to decrement by (60000 shown as signed word)
.L6:
.LVL0:
        MOV.W   R12, @R1                ; use R1 as working register to decrement (first loop)
.L3:
        .loc 1 28 0
        MOV.W   @R1, R13                ; use R13 as working register for this loop
        CMP.W   #0, R13 { JEQ    .L2    ; go to next countdown if this one has reached zero
        .loc 1 29 0
        ADD.W   #-1, @R1                ; decrement counter
        BR  #.L3                        ; loop
.L2:
.LVL1:
        .loc 1 27 0
        MOV.W   R12, @R1                ; reset R1 as working register to decrement (second loop)
.L5:
        .loc 1 28 0
        MOV.W   @R1, R13                ; use R13 as working register for this loop
        CMP.W   #0, R13 { JEQ    .L4    ; break if countdown has reached zero
        .loc 1 29 0
        ADD.W   #-1, @R1                ; decrement counter
        BR  #.L5                        ; loop
.L4:
.LVL2:
.LBE2:
        .loc 1 32 0 discriminator 1
        XOR.B   #65, &P1OUT             ; switch LEDs
        .loc 1 26 0 discriminator 1
        BR  #.L6                        ; loop from top of loops
```

This was generated using the following C code that doubles the count and thus halves the blinking rate.

```c
#include <msp430.h>

void main(void) {
    volatile unsigned int count;
    WDTCTL = WDTPW + WDTHOLD;       // Stop WDT
    P1DIR = 0x41;                   // Set P1 output direction
    P1OUT = 0x01;                   // Set the output

    while (1) {                     // Loop forever
        for (volatile unsigned char i = 0; i < 2; i++) {    // decrement by 60000 twice
            count = 60000;
            while (count != 0) {
                count--;            // decrement
            }
        }
        P1OUT = P1OUT ^ 0x41;       // bitwise xor the output with 0x41
    }
}
```

2 Program 2

Below is the C code used to make the LEDs blink in the red–green–both–none pattern, using the same XOR trick described in the last lab notes.

```c
#include <msp430.h>

volatile unsigned char stateChanger;

void main(void) {
    WDTCTL = WDTPW + WDTHOLD;   // Stop watchdog timer
    P1DIR  = 0xF7;              // C does not have a convenient way of
                                // representing numbers in binary; use hex instead
    P1OUT  = 0x08;              // LEDs off
    P1REN  = 0x08;              // enable resistor
    P1IE   = 0x08;              // Enable input at P1.3 as an interrupt
    stateChanger = 0x1;         // 0x01 to toggle LED0; 0x40 to toggle LED1

    _BIS_SR (LPM4_bits + GIE);  // Turn on interrupts and go into the lowest
                                // power mode (the program stops here)
                                // Notice the strange format of the function, it is an "intrinsic"
                                // ie. not part of C; it is specific to this chipset
}

// Port 1 interrupt service routine
void __attribute__ ((interrupt(PORT1_VECTOR))) PORT1_ISR(void) {
    P1OUT ^= stateChanger;      // toggle the LEDS
    stateChanger ^= 0x40;       // 0x01 -> 0x41 -> 0x01
    P1IFG &= ~0x08;             // Clear P1.3 IFG. If you don't, it just happens again.
}
```

3 Analogue to Digital Conversion

For a 3.3 V CMOS input, $V_{\text{IH}} \geq 2\,\text{V}$ and $V_{\text{IL}} \leq 0.8\,\text{V}$. The ADC scale goes up to 0x3FF, but according to the provided C program, $V_{\text{IH}} \geq$ `0x2FF` $= 767$. Assuming the ADC scale is linear with voltage and that it begins at 0 V, we have $V_{\text{LH}} \leq 0.8\,\text{V} * \frac{767}{2\,\text{V}} = 307 =$ `0x133`. Below is the code for lighting up the red LED on HI, the green LED on LO, and a yellow LED connected to P1.2.

```
#include "msp430.h"

void main(void) {
    WDTCTL    = WDTPW + WDTHOLD;         // Stop WDT
    ADC10CTL0 = ADC10SHT_2 + ADC10ON;    // ADC10ON
    ADC10CTL1 = INCH_1;                  // input A1
    ADC10AE0 |= 0x02;                    // PA.1 ADC option select
    P1DIR    |= 0x45;                    // Set P1.0 to output direction

    while (1) {
        ADC10CTL0 |= ENC + ADC10SC;      // Sampling and conversion start
        while (ADC10CTL1 & ADC10BUSY);   // ADC10BUSY?
        P1OUT &= ~0x45;                  // clear all LEDs first
        if (ADC10MEM > 0x2FF) {          // HI > 2 V   (767 on scale)
            P1OUT |= 0x01;               // set red LED on HI
        } else if (ADC10MEM < 0x133) {   // LO < 0.8 V (307 on scale)
            P1OUT |= 0x40;               // set green LED on LO
        } else {                         // in-between otherwise
            P1OUT |= 0x04;               // set yellow LED connected to P1.2
        }
        unsigned i;
        for (i = 0xFFFF; i > 0; i--);    // Delay
    }
}
```

4 LED Dimmer

Below is the code combining the ADC and the PWM to create a dimmer when the output of the PWM is connected to an LED. We take the sampled voltage as usual and set the duty cycle to a fraction of the period according to the sample, out of `0x3FF`.

```
#include "msp430.h"

void main(void) {
    WDTCTL    = WDTPW + WDTHOLD;         // Stop watchdog
    ADC10CTL0 = ADC10SHT_2 + ADC10ON;    // ADC10ON
    ADC10CTL1 = INCH_1;                  // input A1
    ADC10AE0 |= 0x02;                    // PA.1 ADC option select
    P1DIR |= BIT2;                       // P1.2 to output
    P1SEL |= BIT2;                       // P1.2 to TA0.1
    CCR0  = 1000-1;                      // PWM period
    CCTL1 = OUTMOD_7;                    // CCR1 reset/set
    TACTL = TASSEL_2 + MC_1;             // SMCLK, up mode
```

```
    while (1) {
        ADC10CTL0 |= ENC + ADC10SC;          // Sampling and conversion start
        while (ADC10CTL1 & ADC10BUSY);       // ADC10BUSY?
        CCR1 = (1000 * ADC10MEM) / 0x3FF;    // CCR1 PWM duty cycle set to percentage
                                             // of sampled voltage out of maximum
        for (volatile unsigned int i = 0xFFFF; i > 0; i--);
    }
}
```

5 Sing!

By changing the period of the PWM, we can change the pitch of the tone produced by the piezoelectric buzzer, which means we can make it sing by changing pitches in appropriate intervals. The below makes the buzzer sing Happy Birthday. When connected to the four-digit seven-segment display, it will also display the note being played.

```
#include "msp430.h"

// these aren't actually tuned to those notes,
// they're just named chromatically for convenience
#define c     1000
#define dflat 950
#define d     900
#define eflat 850
#define e     800
#define f     750
#define gflat 720
#define g     670
#define aflat 625
#define a     600
#define bflat 570
#define b     535
#define cc    500
#define n     0

// notes of happy birthday
int hbd[30] = {
    c,  n, c,
    d,  c, f,
    e,  c, n, c,
    d,  c, g,
    f,  c, n, c,
    cc, a, f,
    e,  d, bflat, n, bflat,
    a,  f, g,
    f,  n
};
// relative length of each note
int hbd_lengths[30] = {
    1, 1, 1,
    3, 3, 3,
```

```
    6, 1, 1, 1,
    3, 3, 3,
    6, 1, 1, 1,
    3, 3, 3,
    3, 6, 1, 1, 1,
    3, 3, 3,
    6, 3
};

int scale[13] = {
    c, dflat, d, eflat, e, f, gflat, g, aflat, a, bflat, b, cc
};
int scale_lengths[13] = {
    2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2
};

// P1 pin assignment
// 7 6 4 5   3   2   1  0
// ..note.. str out A1 A0

// show the name of the note in the first (rightmost) digit
// N.B. G displayed as 6
unsigned char show(int note) {
    switch (note) {
        case c:
        case cc:
            return 0xc0;
        case d:
        case dflat:
            return 0xd0;
        case e:
        case eflat:
            return 0xe0;
        case f:
            return 0xf0;
        case g:
        case gflat:
            return 0x60;
        case a:
        case aflat:
            return 0xa0;
        case bflat:
        case b:
            return 0xb0;
        default:
            return 0x00;
    }
}

// if the note is a flat, show F in the second digit
// if the note is a high C, show as CC
```

```
unsigned char acc(int note) {
    switch (note) {
        case bflat:
        case dflat:
        case eflat:
        case gflat:
        case aflat:
            return 0xf1;
        case cc:
            return 0xc1;
        default:
            return 0x01;
    }
}

// argument: one of the notes defined at top
// used as period for PWM
// if 0, set duty cycle to 0 to silence
void play(int note) {
    if (note != 0) {
        CCR0 = note;
        CCR1 = 100;
    } else {
        CCR1 = 0;
    }
}

// strobe output to display
void display(unsigned char out) {
    P1OUT  = out;
    P1OUT |= 0x08;
}

void sing(int* song, int* song_lengths, int length) {
    while (1) {
        for (int i = 0; i < length; i++) {
            int note = song[i];
            play(note);             // set pitch
            display(show(note));    // show note
            display(acc(note));     // show if flat
            // set delay by length of note
            for (volatile unsigned int length = song_lengths[i]; length > 0; length--) {
                for (volatile unsigned int i = 0x3000; i > 0; i--);
            }
        }
    }
}

// set all digits of display to 0
void clear() {
    for (unsigned char i = 0; i < 4; i++) {
```

```
        P1OUT = i;
        P1OUT = i | 0x08;
    }
}

void main(void) {
    WDTCTL = WDTPW + WDTHOLD;     // Stop WDT

    P1DIR  = 0xFF;                // all set to output
    P1SEL |= BIT2;                // P1.2 to TA0.1

    CCTL1 = OUTMOD_7;             // CCR1 reset/set
    TACTL = TASSEL_2 + MC_1;      // SMCLK, up mode

    clear();
    sing(hbd, hbd_lengths, 30);
    //sing(scale, scale_lengths, 13);
}
```