ECE2049 Lab 2 MSP430 Hero Report
Cooper Ducharme
5/2/2023

Stage 1a: Write helper functions to use the 4 buttons and 2 User LEDs

By using the schematics for the ECE2049 Expansion/Lab Board, its four multicolored LEDs were configured as outputs with pins 6.1, 6.2, 6.3, and 6.4. The four SW.PB buttons on the extension board were configured as pullup resistor with pins 7.4, 7.1, 2.2, and 3.6

.

With the function “setLeds”, the LEDs are activated and deactivated. The input takes a number surrounded by asterisks, which binary conversion corresponds to values of turning off one of the four multicpolored LEDs on the expansion board. I have been trying to change the LED initialization so that they only turn on one at a time and have tried to use binary inputs such as 1110, 1101, 1011, and 0111 (aka 14, 13, 11, 7) as opposed to what I would initially think to use, which is 0001, 0010, 0100, and 1000 (or (1, 2, 4, and 8). I have also tried different combinations of setting DIR, REN, and OUT to invert the LED acivation.

Stage 1b: Play different frequencies on the buzzer

Using the function “buzzerTicks,” the buzzer connected to pin 3.5 and configured as an output was pulsed on and off at a certain frequency using the B0 timer. With this method, noise of a certain pitch was emitted. The frequency was controlled by the following equation:
TB0CCR0 = 32768/ticks;
Where ticks is the value input to the buzzer tone function “buzzerTicks”

Stage 2: Configure TimerA2 to generate periodic interrupts and play some notes using the timer to control the duration

The function “swDelay” was included in the template and never used per the lab’s instruction. Instead, the function “pause” was used to space out the time between notes. This function utilized the MSP’s A2 timer by summing the input time delay value “duration” and the “globalTime” value that continuously increments to calculate the value “timeFinish” and stopping when “globalTimer” exceeded “timeFinish”

The function “Timer_A2_ISR” increments “globalTime.

The A2 timer was configured in the function “configureClock” to generate periodic interrupts at a rate of roughly 0.005 seconds with the following equation and due to the fact that periodic interrupts are calculated by the following value divided by 32768:
TA2CCR0= 327/2;

Each note marked as letters A through G was designated as an integer corresponding to the respective frequency.

By stepping through an array of notes called “songNotes” Which was composed of a note designated by a letter and calling the function “buzzerTicks” with each Note as an input at a rate controlled by calling the function “pause” with inputs from a second array of integer values called “songDelays,” a song is automatically played without any inputs from the user.

In addition to the automatic song playing mode, which can be deactivated by setting the integer value “playSongIfEqual1” to a value of zero, I created a test setup which I refer to as keyboard mode. In addition to setting the LEDs to acorosponding configuration, pressing buttons activated a corresponding tone to be emmited by the buzzer.

Conclusion

The skills in digital I/O, internal embedded system timer configuration, and audio manipulation are relivant in control systems and can be applied to many compact uses of sensor reading, output actuation, and computational functionality. Examples of these systems range from a thermostat adjusted heating system, onboard flight control system or even a handheld calculator.

```
/************** ECE2049 DEMO CODE ******************/
/************** 14 May 2018   **********************/
/***************************************************/
// Guitar Hero Template
#include <grlib.h>
#include <LcdDriver/sharp96/Sharp96x96.h>
#include <msp430f5529.h>
#include <peripherals.h>

//#include "lecture.h"//maybe unnessesary
//#include "utils/ustdlib.h"
unsigned long int timer = 0;
// Function Prototypes
void swDelay(char numLoops);
void drawSomeThings(void);
void configureClock();
void pause(unsigned long int duration);
void buzzerTicks(int ticks);
//unsigned char readSButtons(void);
void Timer_A2_ISR(void);


// Declare globals here

unsigned long int globalTime = 0;//timer for entire main

// Main
void main(void)
{
   WDTCTL = WDTPW | WDTHOLD;    // Stop watchdog timer. Always need to stop this!!
```

```
                        // You can then configure it properly, if desired
_BIS_SR(GIE); //global max interrupts
// *** System initialization ***
initLaunchpadButtons();
initLeds();
configDisplay();
KeypadInit();

Graphics_clearDisplay(&g_sContext); // Clear the display

// Look at this function and the comments for some examples of using the LCD
// You can also refer back to lab 0 and the state machine example
// for how to use the keypad and LEDs
drawSomeThings();


//Main Additions

//might go after the rest
P6SEL &= (BIT3|BIT2|BIT1|BIT4); // Set LED pins for digital I/O          edit: used to have ~
//P6DIR |= (BIT3|BIT2|BIT1|BIT0); // Set LED pins as outputs
//P6OUT &= ~(BIT3|BIT2|BIT1|BIT0); // Turn LEDs off (set output register to 0)

//LED Configuration


P6DIR |= BIT2;//red
P6DIR |= BIT1;//green
P6DIR |= BIT3;//blue
P6DIR |= BIT4;//yellow

//P6OUT &= ~(BIT2|BIT1|BIT3|BIT4); // Turn LEDs off (set output register to 0)
P6OUT |= (BIT2|BIT1|BIT3|BIT4);
//Button Configuration


P7SEL &= ~(BIT0);// Set button pins for digital I/O
P2SEL &= ~(BIT2); // Set button pins for digital I/O
P3SEL &= ~(BIT6); // Set button pins for digital I/O
P7SEL &= ~(BIT4); // Set button pins for digital I/O


P7DIR &= ~(BIT0); // Set button pins for digital I/O
P2DIR &= ~(BIT2); // Set button pins for digital I/O
```

```
  P3DIR &= ~(BIT6); // Set button pins for digital I/O
  P7DIR &= ~(BIT4); // Set button pins for digital I/O

  P7REN |= BIT0;//S1
  P3REN |= BIT6;//S2
  P2REN |= BIT2;//S3
  P7REN |= BIT4;//S4

  P7OUT |= BIT0;//S1
  P3OUT |= BIT6;//S2
  P2OUT |= BIT2;//S3
  P7OUT |= BIT4;//S4

  //Buzzer Configuration
     //at this stage, maybe treat like an LED
  P4REN = BIT1;

     //initLeds();// most recent edit: 4/16

  unsigned char ret_val=0;

  int loop = 0;
  int A = 440;
  int B = 494;
  int c = 523;
  int D = 587;
  int E = 659;
  int F = 698;
  int G = 784;


  int state = 0;

  //char endTrigger = 1
  while (1)    // Forever loop
  {
  //  char ret_val = readLaunchpadButtons();
//    setLeds(~ret_val);


      int songNotes[32] = {B, A, G, B, A, G, c, A, G, c, D, G, B, B, E, F, A, c, B, F, F, E, B, A, D, F,
E, B, A, D, c, A};
      int songDelay[32] = {10, 10, 50, 10, 10, 50, 50, 50, 250, 50, 50, 300, 50, 50,50, 50, 25000,
5000, 50, 300, 100, 50000, 5000000, 250000000, 50, 50, 250, 50, 50, 100, 100, 100};
```

```
int wrongNotes = 0;

configureClock();
volatile unsigned int j;
for (j=0; j<31; j++)
{
   buzzerTicks(songNotes[j]);
   pause(songDelay[j]);
}

switch (state)
{

   case 0: //Welcome
   configClock(); //configure time
   //setLeds(0);
   BuzzerOff();
   char ret_val_begin = readSButtons();
   //setLeds(~ret_val_begin);

   while(ret_val_begin)
   {
      //BuzzerOn();

      //pause(2);
      //BuzzerOff();
      //pause(10);


      char ret_val = readSButtons();
      setLeds(~ret_val);
      if (~P7IN & BIT0)
         buzzerTicks(A);
         setLeds('8');
         //pause(20);
      if (~P3IN & BIT6)
         buzzerTicks(c);
         setLeds('4');
         //pause(20);
      if (~P2IN & BIT2)
         buzzerTicks(E);
         setLeds('2');
         //pause(20);
      if (~P7IN & BIT4)
```

```
                buzzerTicks(G);
                setLeds('1');
                //pause(20);
                //ret_val_begin = 0;
                //endTrigger = 0;
            }
        }
    }  // end while (1)
}




void swDelay(char numLoops)
{
    // This function is a software delay. It performs
    // useless loops to waste a bit of time
    //
    // Input: numLoops = number of delay loops to execute
    // Output: none
    //
    // smj, ECE2049, 25 Aug 2013

    volatile unsigned int i,j;  // volatile to prevent removal in optimization
                                // by compiler. Functionally this is useless code

    for (j=0; j<numLoops; j++)
    {
        i = 50000 ;              // SW Delay
        while (i > 0)            // could also have used while (i)
            i--;
    }
}

void drawSomeThings(void)
{
    // Write some text to the display
    // Note:  the constants LCD_HORIZONTAL_MAX and LCD_VERTICAL_MAX
    //        hold the max coordinates of the screen

    int h_center = LCD_HORIZONTAL_MAX / 2; // Variable to keep track of center position
    Graphics_drawStringCentered(&g_sContext, "Welcome",  AUTO_STRING_LENGTH,
h_center, 15, TRANSPARENT_TEXT);
```

```
    Graphics_drawStringCentered(&g_sContext, "to",       AUTO_STRING_LENGTH, h_center,
25, TRANSPARENT_TEXT);
    Graphics_drawStringCentered(&g_sContext, "ECE2049!", AUTO_STRING_LENGTH,
h_center, 35, TRANSPARENT_TEXT);

    // Draw some circles
    Graphics_drawCircle(&g_sContext, 25, 60, 10);
    Graphics_drawCircle(&g_sContext, 25, 60, 7);

    // Horizontal and vertical lines
    Graphics_drawLineH(&g_sContext, 20, 80, 42);
    Graphics_drawLineV(&g_sContext, 10, 20, 80);

    // Arbitrary lines
    Graphics_drawLine(&g_sContext, 60, 60, 80, 80);
    Graphics_drawLine(&g_sContext, 60, 80, 80, 60);

    // To draw a rectangle, we need four points
    // To do this, the graphics library uses a struct to specify each point,
    // which we specify here:
    tRectangle rect; // Declare a struct
    rect.sXMin = 5;  // Fill in each of the four parameters of the struct
    rect.sXMax = 91;
    rect.sYMin = 5;
    rect.sYMax = 91;

    // Draw a rectangle by passing the struct to GrRectDraw
    // (Note that we need to pass the address of the struct!)
    Graphics_drawRectangle(&g_sContext, &rect);

    // This is another way to declare and initialize a struct (if you know all of the values)
    // Here, we also use our constants LCD_HORIZONTAL_MAX and LCD_VERTICAL_MAX
    Graphics_Rectangle box = {.xMin = 5, .xMax = LCD_HORIZONTAL_MAX - 5,
                              .yMin = 5, .yMax = LCD_VERTICAL_MAX - 5 };
    Graphics_drawRectangle(&g_sContext, &box);


    // We are now done writing to the display.  However, if we stopped here, we would not
    // see any changes on the actual LCD.  This is because we need to send our changes
    // to the LCD, which then refreshes the display.
    // Since this is a slow operation, it is best to refresh (or "flush") only after
    // we are done drawing everything we need.
    Graphics_flushBuffer(&g_sContext);
}
```

```
void configureClock() //enables interupts with ACLK + divider + mode
{

   TA2CTL = TASSEL_1 + MC_1 + ID_0;
   TA2CCR0= 327/2;//origionaly 327
   TA2CCTL0 = CCIE;

}

void pause(unsigned long int duration)
{
   volatile long int timeFinish = globalTime+duration;
   while(globalTime <=timeFinish){

   }
}

void buzzerTicks(int ticks)
{
   // Initialize PWM output on P3.5, which corresponds to TB0.5
   P3SEL |= BIT5; // Select peripheral output mode for P3.5
   P3DIR |= BIT5;

   TB0CTL  = (TBSSEL__ACLK|ID__1|MC__UP);  // Configure Timer B0 to use ACLK, divide by
1, up mode
   TB0CTL  &= ~TBIE;                    // Explicitly Disable timer interrupts for safety

   // Now configure the timer period, which controls the PWM period
   // Doing this with a hard coded values is NOT the best method
   // We do it here only as an example. You will fix this in Lab 2.
   TB0CCR0 = 32768/ticks; // Set the PWM period in ACLK ticks
   TB0CCTL0 &= ~CCIE;                // Disable timer interrupts

   // Configure CC register 5, which is connected to our PWM pin TB0.5
   TB0CCTL5  = OUTMOD_7;                  // Set/reset mode for PWM
   TB0CCTL5 &= ~CCIE;                   // Disable capture/compare interrupts
   TB0CCR5   = TB0CCR0/2;                // Configure a 50% duty cycle
}



#pragma vector=TIMER2_A0_VECTOR
__interrupt void Timer_A2_ISR(void)
```

```
{
    globalTime++;
}
```