EE445L – Lab3: Alarm Clock

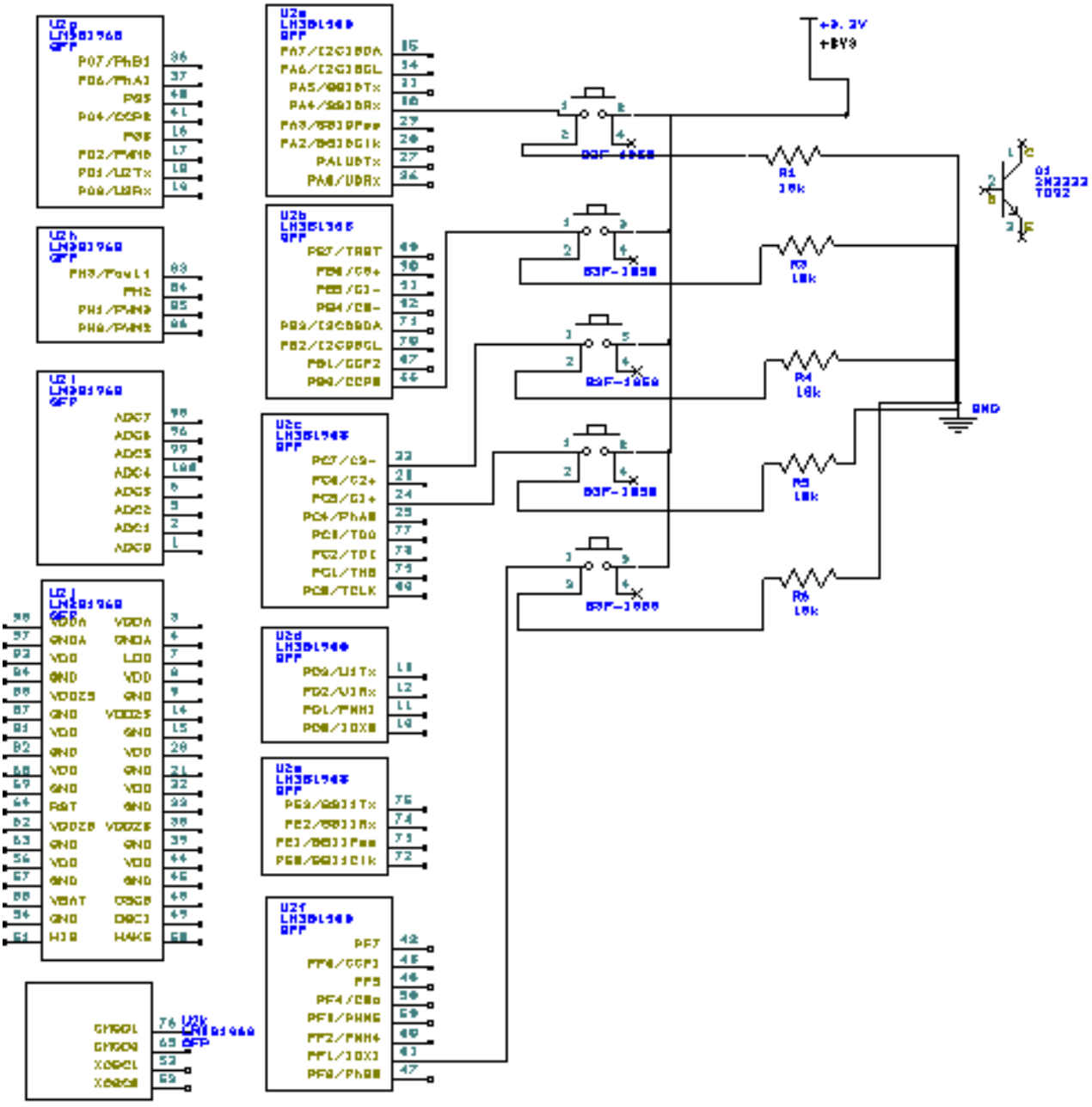
Harley Ross and Dalton Altstaetter

2/4/14

GOALS

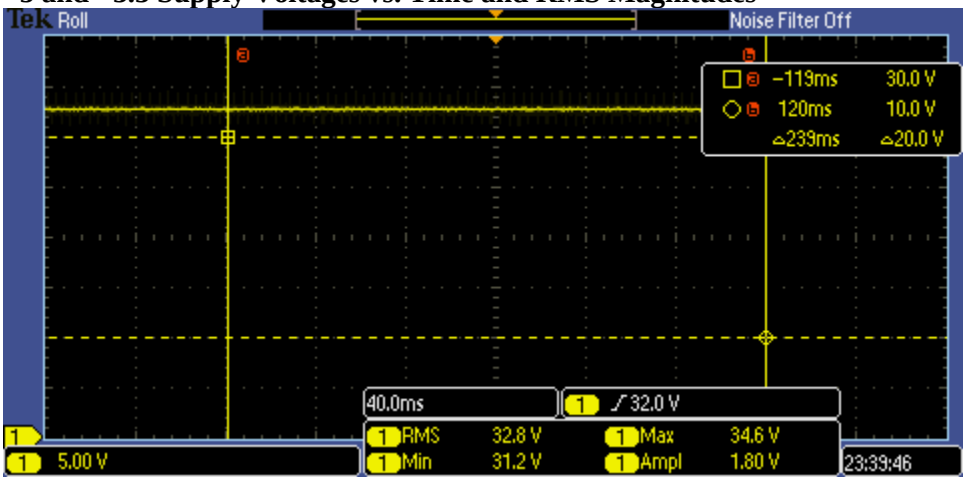
The objectives on this project are to design, build and test an alarm clock. Educationally, we are learning how to design and test modular software and how to perform switch/keypad input in the background.

HARWARE DESIGN

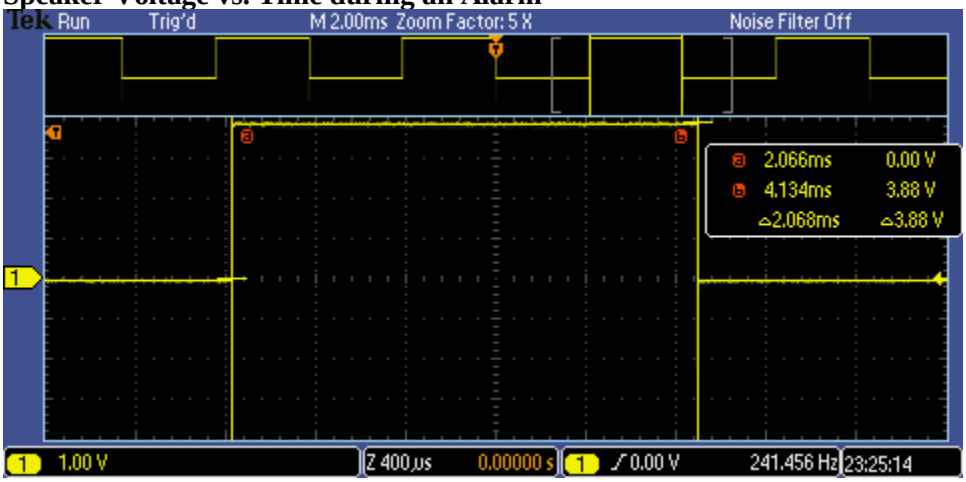


MEASUREMENT DATA

+5 and +3.3 Supply Voltages vs. Time and RMS Magnitudes



Speaker Voltage vs. Time during an Alarm



Measurements of Current Required to Run the Alarm

With the Alarm

Without Alarm

ANALYSIS AND DISCUSION

1. To remove critical sections, you can disable interrupts so that the data being used is not changed during that instruction, or you can have the interrupts not change any data that is used by other instructions.
2. We update the screen every minute and then the OLED takes 10 microseconds to write to the display.
3. You want the interrupt to be a short as possible and the functions called to update the display call other functions.
4. We only used the OLED clear function when switching between displays. Writing over the image on the OLED also clears the image, which is how we updated the screen during the time changes.
5. To save power, you could change the display settings to a lower brightness, have a switch to wake up the screen, or give a time limit to how long the alarm can play so that it does not keep drawing power.

SOURCE CODE

OLED Display

```
#include "OLED_1968/rit128x96x4.h"
#include <stdlib.h>
#include "OledDisplay.h"
#include "lm3s1968.h"
// only adding <stdio.h> for debugging, get rid of it after testing
#include <stdio.h>
#include <string.h>

static const int SIN[POSITIONS] = {  0, 105, 208, 309, 407, 500, 588, 669, 743, 809, 866,
914, 951, 978, 995,
                                  1000, 995, 978, 951, 914, 866, 809, 743, 669, 588, 500,
407, 309, 208, 105,
                                0,-105,-208,-309,-407,-500,-588,-669,-743,-809,-866,-914,-
951,-978,-995,
                                                        -1000,-995,-978,-951,-914,-866,-809,-
743,-669,-588,-500,-407,-309,-208,-105};

static const int COS[POSITIONS] = {1000, 995, 978, 951, 914, 866, 809, 743, 669, 588,
500, 407, 309, 208, 105,
                                                                  0,-105,-208,-309,-407,-
500,-588,-669,-743,-809,-866,-914,-951,-978,-995,
                                  -1000,-995,-978,-951,-914,-866,-809,-743,-669,-588,-500,-
407,-309,-208,-105,
                                 0, 105, 208, 309, 407, 500, 588, 669, 743, 809, 866, 914,
951, 978, 995};

void CreateClockFace(Time* timePtr)
{      // draws the analog clock face
       RIT128x96x4StringDraw("1",51,0,8);RIT128x96x4StringDraw("2",57,0,8);RIT128x96x4Str
ingDraw("!",17,65,8);
       RIT128x96x4StringDraw("1",79,9,8);
       RIT128x96x4StringDraw("2",93,25,8);
       RIT128x96x4StringDraw("3",99,45,8);
       RIT128x96x4StringDraw("4",93,65,8);
       RIT128x96x4StringDraw("5",79,81,8);
       RIT128x96x4StringDraw("6",55,88,8);
       RIT128x96x4StringDraw("7",31,81,8);
       RIT128x96x4StringDraw("8",17,65,8);
       RIT128x96x4StringDraw("9",11,45,8);
       RIT128x96x4StringDraw("1",11,22,8);RIT128x96x4StringDraw("0",17,22,8);
       RIT128x96x4StringDraw("1",28,7,8);RIT128x96x4StringDraw("1",34,7,8);
}


void PrintOneHand(Time* timePtr, int x_pos[],int y_pos[], int* index, int back_forw)
{

       // mod the input for x_Hours[j%60] so it
  // doesn't overruns its buffer size

       // remove the previous minute/hour hand before and
       // after current position for possible setTime
  // function and then print the updated hand
```

```
	RIT128x96x4_LineOff(timePtr->xPivot, timePtr->yPivot,
x_pos[((*index)+2)%POSITIONS], y_pos[((*index)+2)%POSITIONS], 6);
	RIT128x96x4_LineOff(timePtr->xPivot, timePtr->yPivot,
x_pos[((*index)+1)%POSITIONS], y_pos[((*index)+1)%POSITIONS], 6);
	RIT128x96x4_LineOff(timePtr->xPivot, timePtr->yPivot, x_pos[(*index)%POSITIONS],
y_pos[(*index)%POSITIONS], 6);
	RIT128x96x4_LineOff(timePtr->xPivot, timePtr->yPivot, x_pos[((*index)-
1)%POSITIONS], y_pos[((*index)-1)%POSITIONS], 6);
	RIT128x96x4_LineOff(timePtr->xPivot, timePtr->yPivot, x_pos[((*index)-
2)%POSITIONS], y_pos[((*index)-2)%POSITIONS], 6);

	//timePtr->minute_index = (timePtr->minute_index + POSITIONS + back_forw);
	(*index) = (*index) + POSITIONS + back_forw;
	timePtr->direction = back_forw;
}
void DigitalTime(Time* timePtr,int setting)
{
	static char hourCharTime[4];
	static char minCharTime[4];
	static char minCharAlarm[4];
	static char hourCharAlarm[4];

	// this isn't necessary from what I've seen but it may be less buggy
	// by splitting it into separate cases since im using static variables
	if(setting == TIME)
	{
		getValue(hourCharTime,minCharTime,timePtr,setting);
		RIT128x96x4StringDraw(hourCharTime,98,88,15);
		RIT128x96x4StringDraw(hourCharTime+1,104,88,15);
		RIT128x96x4StringDraw(hourCharTime+2,110,88,15);
		RIT128x96x4StringDraw(minCharTime,116,88,15);
		RIT128x96x4StringDraw(minCharTime+1,122,88,15);
  }
	else
	{
		getValue(hourCharAlarm,minCharAlarm,timePtr,setting);
		RIT128x96x4StringDraw(hourCharAlarm,98,88,15);
		RIT128x96x4StringDraw(hourCharAlarm+1,104,88,15);
		RIT128x96x4StringDraw(hourCharAlarm+2,110,88,15);
		RIT128x96x4StringDraw(minCharAlarm,116,88,15);
		RIT128x96x4StringDraw(minCharAlarm+1,122,88,15);
	}
}


void PrintBothHands(Time* timePtr)
{
		// prints both updated clock hands
	RIT128x96x4_Line(timePtr->xPivot, timePtr->yPivot, timePtr->x_minute[timePtr-
>minute_index%POSITIONS], timePtr->y_minute[timePtr->minute_index%POSITIONS], 6);
	RIT128x96x4_Line(timePtr->xPivot, timePtr->yPivot, timePtr->x_hour[timePtr-
>hour_index%POSITIONS], timePtr->y_hour[timePtr->hour_index%POSITIONS], 15);
}

void getValue(char* a_hours, char* a_min, Time* timePtr,int setting)
{ // none of this code should be re-arranged. It will produce incorrect output it done
	// it will change the hour at the XX:48 minute rather than waiting for the 60th
min
```

```
	static int prevMinTime,prevHourTime,prevMinAlarm,prevHourAlarm;
	static int flagTime = 0;
	static int flagAlarm = 0;
  static char hoursTime,hoursAlarm;
  static char minutesTime,minutesAlarm;

	prevMinTime = minutesTime;
	prevHourTime = hoursTime;

	prevMinAlarm = minutesAlarm;
	prevHourAlarm = hoursAlarm;


	if(setting == TIME)
	{
		minutesTime = (timePtr->minute_index%60); // 0-59 minutes
		sprintf(a_min,"%02d",minutesTime);

		if(flagTime)
		{
			if(minutesTime/12 == 4)
			{
				return;
			}
			else
			{
				hoursTime = ((timePtr->hour_index%60))/5; // 0-11 minutes
				sprintf(a_hours,"%2d",hoursTime);
				flagTime = 0;
			}
		}

		// this code cannot be moved around
		if(minutesTime/12 == 4 && timePtr->direction == FORWARDS)
		{
			return;
		}

		// this code cannot be moved around this function moving it below the next
		// if(.) statement produces incorrect  output
		hoursTime = ((timePtr->hour_index%60))/5; // 0-11 minutes
		// I need this so that when advancing forward it stays on 12 if the hour =
0 from
		// the previous statement. this is the main code that runs when everything
is moving
		// only forward when pressing the increment button
		if(hoursTime == 0) // zero corresponds to 12 on the clockface
		{
			hoursTime = 12;
		}

		if(timePtr->direction == BACKWARDS)
		{
			if(prevMinTime == 0 && minutesTime == 59)
			{
				hoursTime = prevHourTime-1;
				if(hoursTime == 0) // zero corresponds to 12 on the clockface
				{
```

```
                                hoursTime = 12;
                        }
                        flagTime = 1;
                }
        }

        sprintf(a_hours,"%2d",hoursTime);
        a_hours[2] = ':';
}
// This basically repeats everything that was written above except
// that is uses its own exclusive variables.
else if(setting == ALARM)
{
        minutesAlarm = (timePtr->minute_index%60);      // 0-59 minutes
        sprintf(a_min,"%02d",minutesAlarm);

        if(flagAlarm)
        {
                if(minutesAlarm/12 == 4)
                {
                        return;
                }
                else
                {
                        hoursAlarm = ((timePtr->hour_index%60))/5; // 0-11 minutes
                        sprintf(a_hours,"%2d",hoursAlarm);
                        flagAlarm = 0;
                }
        }

        // this code cannot be moved around
        if(minutesAlarm/12 == 4 && timePtr->direction == FORWARDS)
        {
                return;
        }

        // this code cannot be moved around this function moving it below the next
        // if(.) statement produces incorrect  output
        hoursAlarm = ((timePtr->hour_index%60))/5; // 0-11 minutes
        // I need this so that when advancing forward it stays on 12 if the hour =
0 from
        // the previous statement. this is the main code that runs when everything
is moving
        // only forward when pressing the increment button
        if(hoursAlarm == 0) // zero corresponds to 12 on the clockface
        {
                hoursAlarm = 12;
        }

        if(timePtr->direction == BACKWARDS)
        {
                if(prevMinAlarm == 0 && minutesAlarm == 59)
                {
                        hoursAlarm = prevHourAlarm-1;
                        if(hoursAlarm == 0) // zero corresponds to 12 on the clockface
                        {
                                hoursAlarm = 12;
                        }
```

```
                        flagAlarm = 1;
                }
            }
            sprintf(a_hours,"%2d",hoursAlarm);
            a_hours[2] = ':';
        }
}

// input: unfilled struct
// purpose: to fill the struct (x,y) positions
// for the hour and minute hands
static void CalculateClockHandPositions(Time* timePtr)
{
        int i; // indices

        for(i = 0; i < POSITIONS; i++)
        {
            // This sets the (x,y) coordinates for all possible hour hand coordinates
            // there are 60 coordinates, one for every 5 minute increment
            timePtr->x_hour[i] = (timePtr->initHour_x*1000 + SIN[i]*HOURHANDLENGTH +
500)/1000;
            // the cosine makes it negative so adding HOURHANDLENGTH gets it back to
the initial position
            timePtr->y_hour[i] = (timePtr->initHour_y*1000 - COS[i]*HOURHANDLENGTH +
500)/1000 + HOURHANDLENGTH;


            // This sets the (x,y) coordinates for all possible minute hand coordinates
            // there are 60 coordinates, one position for every minute
            timePtr->x_minute[i] = (timePtr->initMinute_x*1000 +
SIN[i]*MINUTEHANDLENGTH + 500)/1000;
            // the cosine makes it negative so adding MINUTEHANDLENGTH gets it back to
the initial position
            timePtr->y_minute[i] = (timePtr->initMinute_y*1000 -
COS[i]*MINUTEHANDLENGTH + 500)/1000 + MINUTEHANDLENGTH;
        }
}

Time* Time_Init(void)
{
        // create a timePtr to contain the struct
        // of type Time* which is a pointer to a Time struct
        // casted to (Time*) bc thats the data type
        // of size(Time) bc thats how large a struct is

        Time* timePtr = (Time*)malloc(sizeof(Time));

        timePtr->hour_index = 0;
        timePtr->minute_index = 0;

        timePtr->xPivot = XPIVOT;
        timePtr->yPivot = YPIVOT;

        timePtr->initHour_x = 0;
        timePtr->initHour_y = 0;

        timePtr->initMinute_x = 0;
        timePtr->initMinute_y = 0;
```

```
	// set minute hand to 12 O'clock position
	timePtr->initMinute_x = XPIVOT;
	timePtr->initMinute_y = YPIVOT - MINUTEHANDLENGTH;

	// set minute hand to 12 O'clock position
	timePtr->initHour_x = XPIVOT;
	timePtr->initHour_y = YPIVOT - HOURHANDLENGTH;

	// set pointers to position Array
	timePtr->x_hour = &x_hour1[0];
	timePtr->y_hour = &y_hour1[0];

	timePtr->x_minute = &x_minute1[0];
	timePtr->y_minute = &y_minute1[0];;

	return timePtr;
}


static void CreateClockDisplay(Time* timePtr, int back_forw)
{
	PrintOneHand(timePtr,timePtr->x_minute,timePtr->y_minute,&timePtr-
>minute_index,back_forw);
	PrintBothHands(timePtr);
	// move the hour hand if the minute hand has move 12 spots,
	// I added 6 for the same effect as rounding so it looks more fluid
	if(((timePtr->minute_index + 11) % 12) == 0)
	{
		// remove the previous hour hands before and after
	  // current position for possible setTime() function
		PrintOneHand(timePtr,timePtr->x_hour,timePtr->y_hour,&timePtr-
>hour_index,back_forw);
		RIT128x96x4_Line(timePtr->xPivot, timePtr->yPivot, timePtr->x_hour[timePtr-
>hour_index%POSITIONS], timePtr->y_hour[timePtr->hour_index%POSITIONS], 15);
	}
}

static void ChangeTimeManually(Time* timePtr,long* seconds,int update)
{
	// now I am not so sure why I included this if() to begin with
//	if(*seconds == -1)
//	{
//	  // this is here bc we first want to put the clock hands on the screen on the
//		// screen at startup/initialization without them changing automatically to
12:01
//		CreateClockDisplay(timePtr,DEFAULT);
//		(*seconds)++;
//	}
//	else
	{
		if(update == FORWARDS) // move forward
		{ // this is for moving forwards in setTime
			CreateClockDisplay(timePtr,FORWARDS);
			*seconds = 0; // bc we just set a newTime and want to begin a new
Sec count
			GPIO_PORTH_DATA_R ^= 0x0D; // flashes & tics the OLED screen
		}
```

```
		else if(update == BACKWARDS) // move backward
		{
			CreateClockDisplay(timePtr,BACKWARDS);
			*seconds = 0; // bc we just set a newTime and want to begin a new
Sec count
			GPIO_PORTH_DATA_R ^= 0x0D; // flashes & tics the OLED screen
		}
		else if(update == DEFAULT) // move backward
		{// at the beginning of the program this sets seconds cnt to 0, where it
should be
			CreateClockDisplay(timePtr,DEFAULT);
			(*seconds)++; // bc we just set AlarmTime and want to Revert to
current time
		}
	}
}

static void PeriodicTimeChange(unsigned long* count0,long* seconds,Time* timePtr)
{
	static unsigned long temp = 0;

	// 5 interrupts->1
	// if the count is a multiple of 5, update display
	// Int Freq is 5Hz => LED updates every second
	if(!((*count0)%5) )
	{
		// this only updates the display when its absolutely necessary
		// and so it doesn't do this for all time until the next interrupt
		if(temp != (*count0))
		{
			GPIO_PORTG_DATA_R = GPIO_PORTG_DATA_R^0x04; // toggle PG2
			temp = (*count0);
			GPIO_PORTH_DATA_R ^= 0x01; // implements a Tic-Toc sound
			// These flash the screen for some reason
			// not sure why
			if(!((*seconds)%60) && ((*seconds) != 0))
			{
				// update the display if its been 60 seconds
				// seconds != 0 is included as a corner case for
				// when the seconds is first initialized so that it
				// doesn't start at 12:01
				CreateClockDisplay(timePtr, FORWARDS);
			}
			(*seconds)++;
		}
	}
}

void DisplayFunction(Time* timePtr, unsigned long count0, signed int update)
{ // temp is a local private variable that only has scope within this function

	static long seconds = -1;

	// this is for changing any of the alarm times or printing the
	// previous time after a new alarm/real time was changed
	if((seconds == SETUP) || (update == FORWARDS) || (update == BACKWARDS) || (update
== DEFAULT)) // move forward
	{ // this is for moving forwards in setTime
```

```
                // Changes minute/
                ChangeTimeManually(timePtr, &seconds,update);
        }
        // this is for moving the hands only when it is
        // supposed to every minute/hour that occurs
        else
        {
                PeriodicTimeChange(&count0,&seconds,timePtr);
        }
}

void Clock_Init(Time* timePtr)//,int forw_back)
{
  CalculateClockHandPositions(timePtr);
        CreateClockFace(timePtr);

        DisplayFunction(timePtr,global_count0,0);
}

void SetTime(Time* timePtr, volatile int* global_flag)
{
        static unsigned long prevTime0;
        // clear any pending interrupts that were triggered
        GPIO_PORTB_RIS_R = 0;
        GPIO_PORTF_RIS_R = 0;
        // disable all interrupts except the SysTick_Handler()
        // writing to the port to disable it is a friendly operation for that IRQ
        NVIC_DIS0_R = NVIC_DIS0_INT0;
        NVIC_DIS0_R = NVIC_DIS0_INT1;
        NVIC_DIS0_R = NVIC_DIS0_INT30;

        prevTime0 = global_count0;
        // this
        while((global_count0-prevTime0) < 50)
        {
                // ^^^ this waits 10 sec for a switch to be pressed
                // reads PC5(incr) & PC7(decr)
                if(GPIO_PORTC_DATA_R & 0x20)
                {
                        SysTick_Wait10ms(1);
                        if(GPIO_PORTC_DATA_R & 0x20)
                        {
                                // increment timePtr index of minute hand and then draw it
                                DisplayFunction(timePtr, global_count0,FORWARDS);
                                return;
                        }
                }
                else   if(GPIO_PORTC_DATA_R & 0x80)
                {
                        SysTick_Wait10ms(1);
                        if(GPIO_PORTC_DATA_R & 0x80)
                        {
                                DisplayFunction(timePtr, global_count0,BACKWARDS);
                                return;
                        }
                }
        }
```

```
        // stop
        *global_flag = 0; // reset flag
  global_count0 = 0; // this resets the seconds count

        // re-enable lower priority interrupts
        NVIC_EN0_R = NVIC_EN0_INT0;
        NVIC_EN0_R = NVIC_EN0_INT1;
        NVIC_EN0_R = NVIC_EN0_INT30;
}

int TimerCompare(Time* timePtr, Time* alarmPtr)
{
        int tMin;
        int tHour;
        int aMin;
        int aHour;

        tMin = (timePtr->minute_index)%POSITIONS;
        tHour = (timePtr->hour_index)%POSITIONS;

        aMin = (alarmPtr->minute_index)%POSITIONS;
        aHour = (alarmPtr->hour_index)%POSITIONS;

        if(tMin == aMin)
        {
                if(tHour == aHour)
                {
                        return 1;
                }
        }

        return 0;
}
```

Speaker

```
extern volatile int flagB0; // global variable that turns on alarm
#include "lm3s1968.h"

void play_Alarm(void)
{
        static int cnt;
        unsigned long delayCnt;

        NVIC_DIS0_R = NVIC_DIS0_INT0;  // disables set time interrupt,
        NVIC_DIS0_R = NVIC_DIS0_INT30; // mode, & setAlarmTime interrupts

        cnt = 0;
        while(flagB0)// button is not pressed
        {
                delayCnt = 1000000;
                cnt++;
                for(delayCnt = 50000; delayCnt != 0; delayCnt--)
                {
                        if((delayCnt %25000) == 0)
                        {
                                GPIO_PORTH_DATA_R ^= 0x01;
                        }
```

```
		}

		GPIO_PORTG_DATA_R &= ~0x04;
		for(delayCnt = 5000000; delayCnt != 0; delayCnt--)
		{
			if((delayCnt %15000) == 0)
			{
				GPIO_PORTH_DATA_R ^= 0x01;
			}
			delayCnt--;
		}

		for(delayCnt = 5000000; delayCnt != 0; delayCnt--)
		{
			if((delayCnt %12000) == 0)
			{
				GPIO_PORTH_DATA_R ^= 0x01;
			}
			delayCnt--;
		}
	}
	NVIC_EN0_R = NVIC_EN0_INT0;  // disables set time interrupt,
	NVIC_EN0_R = NVIC_EN0_INT30; // mode, & setAlarmTime interrupts
}
```

Switches

```
#include "lm3s1968.h"
#include <stdio.h>

extern void SysTick_Wait10ms(unsigned long delay);
extern void DisableInterrupts(void);

extern volatile int flagA0;
extern volatile int flagB0;
extern volatile int flagF1;
extern volatile int flagF2;
extern volatile int clear_flag;

static void Delay(unsigned long count)
{
		while(count)
	{
		count--;
	}
}

static void PortA_Init(void) // PL = 0 (Highest) SetTime()
{
//----------- PA4 -> setNewTime(), PL = 0 (Highest)--------------------------
		SYSCTL_RCGC2_R |= SYSCTL_RCGC2_GPIOA; // enable port A
		Delay(100000); // give it time to enable the port
		GPIO_PORTA_DIR_R &= ~0x10;   // make PA4 input
	  GPIO_PORTA_AFSEL_R &= ~0x10; // disable alt funct on PF0-1
		GPIO_PORTA_DEN_R |= 0x10;    // enable digital I/O on PF0-3
		GPIO_PORTA_IS_R &= ~0x10;    // makes PA0 level-triggered interrupts
		GPIO_PORTA_IBE_R &= ~0x10;   //sets it so it looks at GPIO_IEV
		GPIO_PORTA_ICR_R = 0x10;     // clear flag0, do this every ISR call
		GPIO_PORTA_IEV_R |= 0x10;    // interrupt triggers on HIGH level
```

```
              GPIO_PORTA_IM_R |= 0x10;     // arm interrupt
              NVIC_PRI0_R = (NVIC_PRI0_R&0xFFFFFF1F)|0x0000000; // sets bits 5-7 to 0.
pri = 0;
              NVIC_EN0_R = NVIC_EN0_INT0; // enables intr in PA, its a friendly operation
              // NVIC_DIS?_R disables interrupts for that particular port letter ?
//---------------------------------------------------------------------
}

static void PortB_Init(void) // PL = 7 (Lowest) EnAlarm()
{
//----------- PortB PB0 -> EnableAlarm(),Priority Level(PL)=4  ---------
              SYSCTL_RCGC2_R |= SYSCTL_RCGC2_GPIOB; // enable port B
              Delay(100000); // give it time to enable the port
              GPIO_PORTB_DIR_R &= ~0x01;   // make PB0 input
              GPIO_PORTB_AFSEL_R &= ~0x01; // disable alt funct on PF0-1
              GPIO_PORTB_DEN_R |= 0x01;    // enable digital I/O on PF0-3
              GPIO_PORTB_IS_R &= ~0x01;    // makes PA0 level-triggered interrupts
              GPIO_PORTB_IBE_R &= ~0x01;   //sets it so it looks at GPIO_IEV
              GPIO_PORTB_ICR_R = 0x01;     // clear flag0, do this every ISR call
              GPIO_PORTB_IEV_R |= 0x01;    // interrupt triggers on HIGH level
              GPIO_PORTB_IM_R |= 0x01;     // arm interrupt
              NVIC_PRI0_R = (NVIC_PRI0_R&0xFFFF8FFF)|0x0000E000; // sets bits 13-15 to
001 respect. pri = 4;
              NVIC_EN0_R = NVIC_EN0_INT1; // enables intr in PA, its a friendly operation
//            // NVIC_DIS0_R disables interrupts for that particular port letter
//---------------------------------------------------------------------
}

static void PortC_Init(void) // increment and decrement buttons
{
//----------- PortB PC0 -> Inc&Dec Minute Hand  ---------
// PC5 -> increment, PC7 -> decrement
              SYSCTL_RCGC2_R |= SYSCTL_RCGC2_GPIOC; // enable port B
              Delay(100000); // give it time to enable the port
              GPIO_PORTC_DIR_R &= ~0xA0;   // make PB5,PC7 input
              GPIO_PORTC_AFSEL_R &= ~0xA0; // disable alt funct on PB5,PC7
              GPIO_PORTC_DEN_R |= 0xA0;    // enable digital I/O on PB5,PC7
//---------------------------------------------------------------------
}

static void PortF_Init(void) // PL = 5 (Medium) chmod() setAlarm()
{
// PortF PF2 -> setAlarmTime() & PF1->DisplayMode(), they have same Priority lvl
//--------------------- Priority Level(PL) = 5 ----------------------------
              SYSCTL_RCGC2_R |= SYSCTL_RCGC2_GPIOF;
              Delay(100000); // give it time to enable the port
       GPIO_PORTF_DIR_R &= ~0x06; // make PF2-PF1 inputs
              GPIO_PORTF_AFSEL_R &= ~0x06;// disable alt funct on PF2-1
              GPIO_PORTF_DEN_R |= 0x06; // enable digital I/O on PF2-1
              GPIO_PORTF_IS_R &= ~0x06; // makes PF2-1 level-triggered interrupts
              GPIO_PORTF_IBE_R &= ~0x06; //sets it so it looks at GPIO_IEV
              GPIO_PORTF_ICR_R = 0x06;     // clear flag0, do this every ISR call
              GPIO_PORTF_IEV_R |= 0x06; // interrupt triggers on HIGH level
              GPIO_PORTF_IM_R |= 0x06; // interrupt triggers on HIGH level
              NVIC_PRI7_R = (NVIC_PRI7_R&0xFF1FFFFF)|0x00A00000;//|0x00600000;
//
//            // sets bits 21-23 to prior_lvl=3;
//            // bit21 = 1, bit22 = 1, bit23 = 0
```

```
    NVIC_EN0_R = NVIC_EN0_INT30; // enables intr in PF, its a friendly operation

    // NVIC_DIS0_R disables interrupts for that particular port letter
//-------------------------------------------------------------------
}

static void PortG_Init(void)
{
//----------- PortG -------------------------
              SYSCTL_RCGC2_R |= SYSCTL_RCGC2_GPIOG; // enable port B
              Delay(100000); // give it time to enable the port
              GPIO_PORTG_DIR_R |= 0x04; // enables the on-board LED
              GPIO_PORTG_DEN_R |= 0x04;
         GPIO_PORTG_AFSEL_R &= ~0x04;
//-------------------------------------------------------------------------
}

static void PortH_Init(void)
{
       SYSCTL_RCGC2_R |= SYSCTL_RCGC2_GPIOH;
       Delay(100000);
       GPIO_PORTH_DIR_R |= 0x0F;
       GPIO_PORTH_DEN_R |= 0x0F;
       GPIO_PORTH_AFSEL_R &= ~0x0F;
       GPIO_PORTH_DATA_R = 0x00;
}

void GPIO_Ports_Init(void)
{
       PortA_Init();
       PortB_Init();
       PortC_Init();
       PortF_Init();
       PortG_Init();
       PortH_Init();
}

void GPIOPortA_Handler(void) // PL = 0 (Highest) SetTime()
{
       // PL = 0 (Highest) SetTime()

       // acknowledge the interrupt
       // we only have to check bit 0
       GPIO_PORTA_ICR_R = 0x10;
       SysTick_Wait10ms(1); // switch debouncing

       if(GPIO_PORTA_DATA_R & 0x10)
       {
              //printf("PA0->setTime");
              flagA0 = 1; // used to enter SetTime()
       }
}

void GPIOPortB_Handler(void) // PL = 7 (Lowest) EnAlarm()
{
       // PL = 7 (Lowest) EnAlarm()

       // acknowledge the interrupt
```

```
	// we only have to check bit 0
	GPIO_PORTB_ICR_R = 0x01;
	SysTick_Wait10ms(1); // switch debouncing

	if(GPIO_PORTB_DATA_R & 0x01)
	{
		if(flagB0 == 1)
		{
			flagB0 = 0; // disables alarm once this occurs
								// while the alarm is already
going off
		}
		else
		{
			flagB0 = 1; // enables alarm
		}
	}

	// this leave with flagB0 = 1, enabling the alarm or
	// or leaves with flagB0 = disabling the alarm
  // you press the same button to enable it as you do to disable it
}

void GPIOPortF_Handler(void) // PL = 5 (Medium) PF1->setAlarm() PF2->chmod()
{
	// PL = 5 (Medium) chmod() setAlarm()

	// acknowledge the interrupt
	// we only have to check
	// both bit 0 and bit 1
	GPIO_PORTF_ICR_R = 0x01;
	SysTick_Wait10ms(1); // switch debouncing

	// PF1 interrupt -> SetAlarm()
	if(GPIO_PORTF_RIS_R & 0x00000002)
	{
		// ACK interrupt
		GPIO_PORTF_ICR_R = 0x02;  // clear flag, ack intrp
		if(GPIO_PORTF_DATA_R & 0x02)
		{
			flagF1 = 1;
		}
	}
	// PF2 interrupt -> change Display mode
	else if(GPIO_PORTF_RIS_R & 0x00000004)
	{
		// ACK interrupt
		GPIO_PORTF_ICR_R = 0x04;
		if(GPIO_PORTF_DATA_R & 0x04)
		{
			if(flagF2) // if digital is on flagF2=0 turns it off
			{
				flagF2 = 0;
				clear_flag = 1;
			}
			else
			{
				flagF2 = 1;
```

```
                        clear_flag = 1;
                    }
                }
            }
}
```

SysTick

```
// SysTick.c
// Runs on LM3S1968
// Provide functions that initialize the SysTick module, wait at least a
// designated number of clock cycles, and wait approximately a multiple
// of 10 milliseconds using busy wait.  After a power-on-reset, the
// LM3S1968 gets its clock from the 12 MHz internal oscillator, which
// can vary by +/- 30%.  If you are using this module, you probably need
// more precise timing, so it is assumed that you are using the PLL to
// set the system clock to 50 MHz.  This matters for the function
// SysTick_Wait10ms(), which will wait longer than 10 ms if the clock is
// slower.
// Daniel Valvano
// February 22, 2012

/* This example accompanies the book
   "Embedded Systems: Real Time Interfacing to the Arm Cortex M3",
   ISBN: 978-1463590154, Jonathan Valvano, copyright (c) 2011
   Program 2.11, Section 2.6

 Copyright 2011 by Jonathan W. Valvano, valvano@mail.utexas.edu
    You may use, edit, run or distribute this file
    as long as the above copyright notice remains
 THIS SOFTWARE IS PROVIDED "AS IS".  NO WARRANTIES, WHETHER EXPRESS, IMPLIED
 OR STATUTORY, INCLUDING, BUT NOT LIMITED TO, IMPLIED WARRANTIES OF
 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE APPLY TO THIS SOFTWARE.
 VALVANO SHALL NOT, IN ANY CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL,
 OR CONSEQUENTIAL DAMAGES, FOR ANY REASON WHATSOEVER.
 For more information about my classes, my research, and my books, see
 http://users.ece.utexas.edu/~valvano/
 */
#define GPIO_PORTH_DATA_R       (*((volatile unsigned long *)0x400273FC))
#define NVIC_SYS_PRI3_R         (*((volatile unsigned long *)0xE000ED20))
#define NVIC_ST_CTRL_R          (*((volatile unsigned long *)0xE000E010))
#define NVIC_ST_RELOAD_R        (*((volatile unsigned long *)0xE000E014))
#define NVIC_ST_CURRENT_R       (*((volatile unsigned long *)0xE000E018))
#define NVIC_ST_CTRL_COUNT      0x00010000  // Count flag
#define NVIC_ST_CTRL_CLK_SRC    0x00000004  // Clock Source
#define NVIC_ST_CTRL_INTEN      0x00000002  // Interrupt enable
#define NVIC_ST_CTRL_ENABLE     0x00000001  // Counter mode
#define NVIC_ST_RELOAD_M        0x00FFFFFF  // Counter load value


extern volatile unsigned long global_count0;

// Initialize SysTick with busy wait running at bus clock.
void SysTick_Init(unsigned long reloadValue)
{
  NVIC_ST_CTRL_R = 0;                   // disable SysTick during setup
  NVIC_ST_RELOAD_R = reloadValue;       // reloads every 200ms
  NVIC_ST_CURRENT_R = 0;                // any write to current clears it
                                        // enable SysTick with core clock
```

```
        // I added  NVIC_ST_CTRL_INTEN to NVIC_ST_CTR_R
        NVIC_SYS_PRI3_R = (NVIC_SYS_PRI3_R & 0x1FFFFFFF); // PL = 0
        NVIC_ST_CTRL_R = NVIC_ST_CTRL_ENABLE+NVIC_ST_CTRL_CLK_SRC+NVIC_ST_CTRL_INTEN;
}
// The delay parameter is in units of the core clock. (units of 20 nsec for 50 MHz clock)
void SysTick_Wait(unsigned long delay)
{
  volatile unsigned long elapsedTime;
  unsigned long startTime = NVIC_ST_CURRENT_R;
  do{
    elapsedTime = (startTime-NVIC_ST_CURRENT_R)&0x00FFFFFF;
  }
  while(elapsedTime <= delay);
}
// This assumes 50 MHz system clock.
void SysTick_Wait10ms(unsigned long delay)
{
  unsigned long i;
  for(i=0; i<delay; i++)
        {
    SysTick_Wait(500000);  // wait 10ms (assumes 50 MHz clock)
  }
}

void SysTick_Handler(void)
{
        global_count0++;
}
```