EE445L – Lab4: Alarm Clock

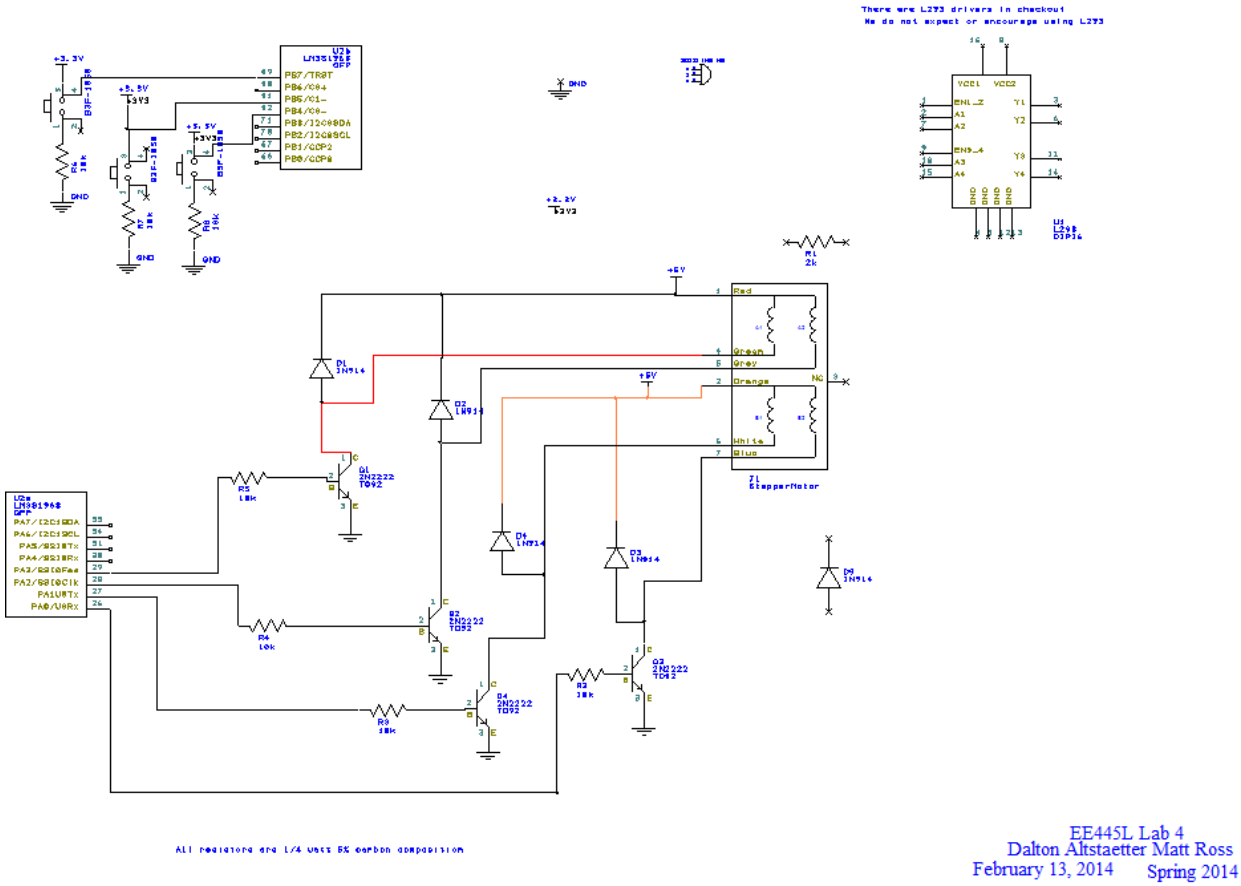
Harley Ross and Dalton Altstaetter

2/18/14

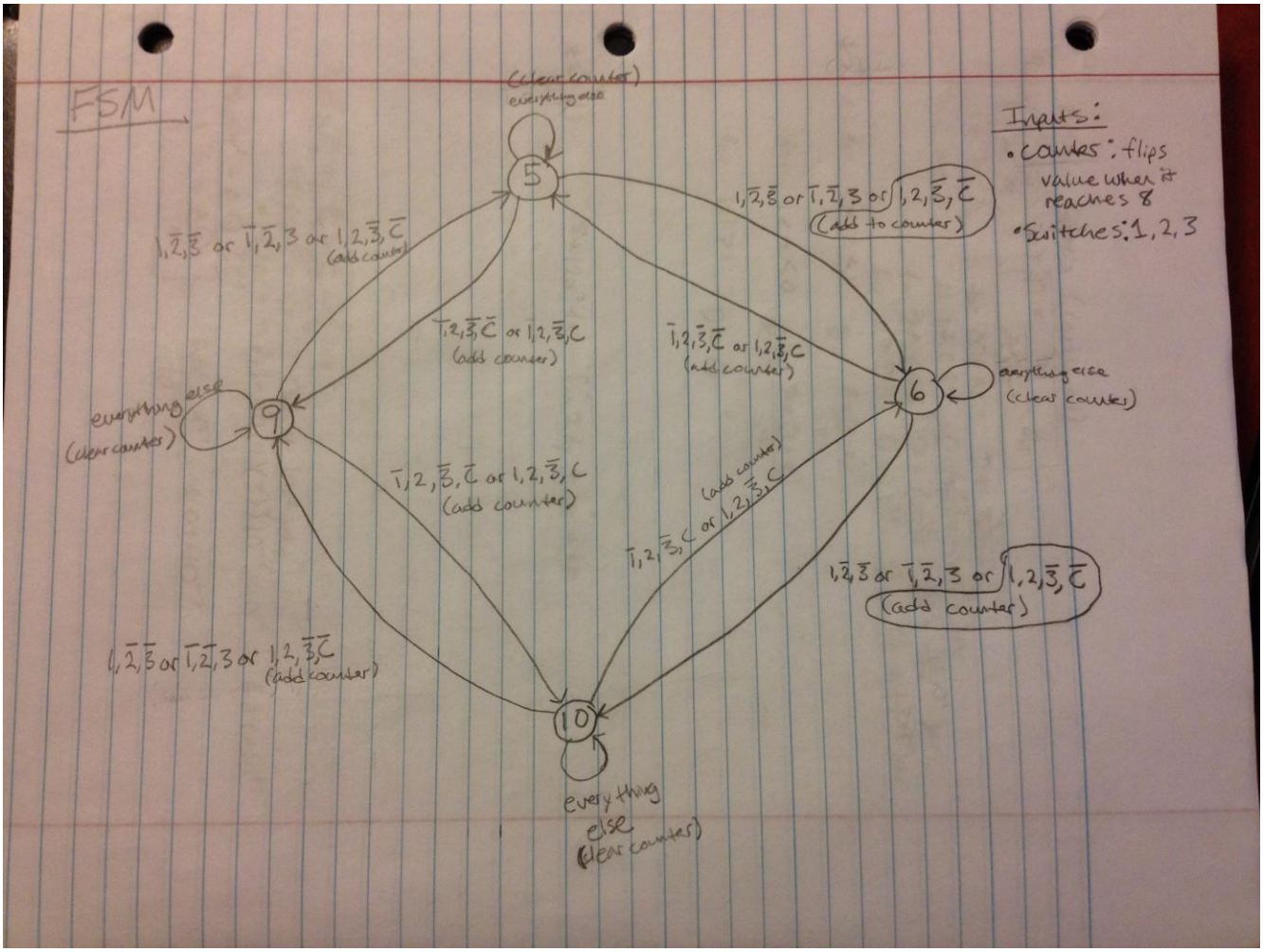
GOALS

The objectives on this project are to interface a stepper motor, to implement background processing with periodic interrupts, and to develop a linked command structure.

HARWARE DESIGN



SOFTWARE DESIGN



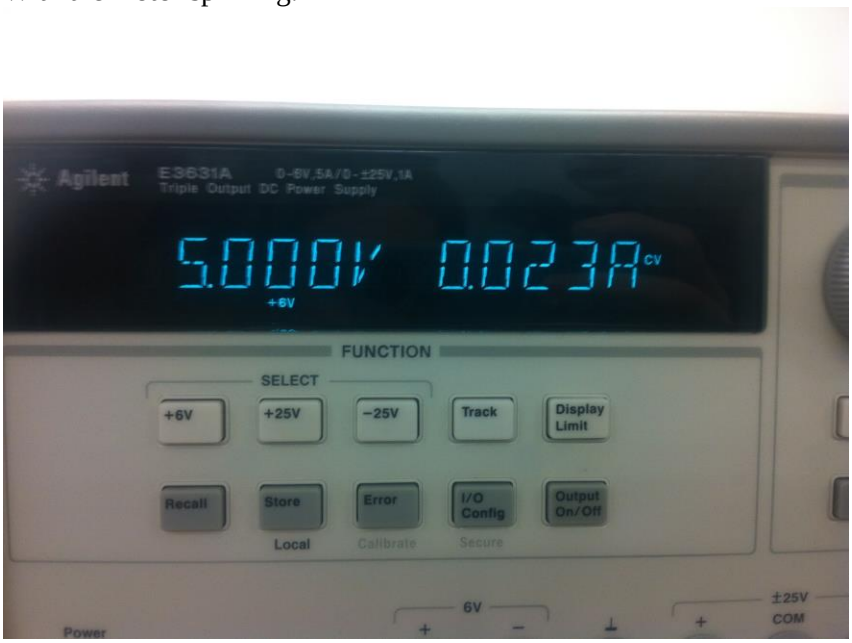
MEASUREMENT DATA
Voltage: 5V
Current: 100mA
Resistance: 50 ohms

Motor Rate: 107 rpm clockwise, 106 rpm counterclockwise, ~188Hz oscillation

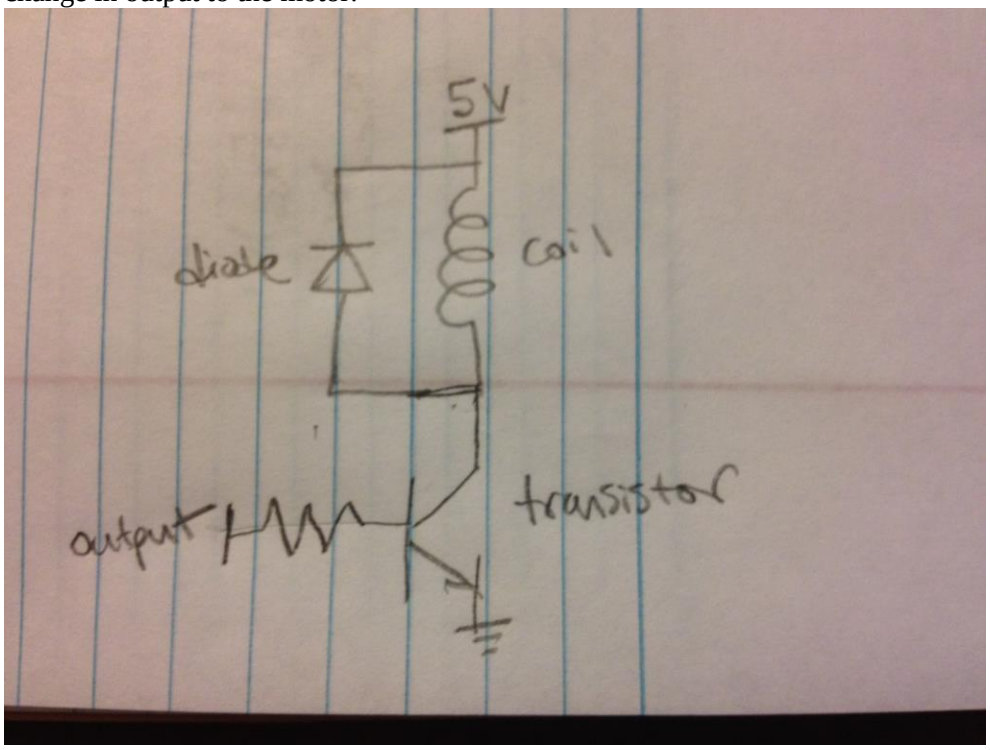
ISR maximum time:

Current required to run the system
Without the motor spinning:

With the motor spinning:



ANALYSIS AND DISCUSION

1. Jerk is the high acceleration of the stepper motor. Jerk is reduced by putting a delay between each change in output to the motor.
2. 

The transistor allows current to flow through when the microcontroller changes the output to 1. This allows the 5V to flow through the coil and creates an electric field. When the output changes to a zero, the transistor cuts the connection and the coil reverses the current when the electric field dissipates. The diode is put in place to not allow the current to reverse towards the 5V source voltage.
3. We had to select the right voltage and resistance to supply the correct current to the coils. We satisfied these parameters by choosing the 5V source and 50ohm resistance.
4. The current will increase when a mechanical load is applied because the electric field caused by the coil will change.
5. P = IV, so the power produced by our motor is .5 watts. Mechanical power is the amount of work or energy over time. Electrical power is related to mechanical power because both are the rate of energy being used over time.

SOURCE CODE

PLL

```
// PLL.c
// Runs on LM3S1968
// A software function to change the bus speed using the PLL.
// Commented lines in the function PLL_Init() initialize the PWM
// to either 25 MHz or 50 MHz.  When using an oscilloscope to
// look at LED0, it should be clear to see that the LED flashes
// about 2 (50/25) times faster with a 50 MHz clock than with a
// 25 MHz clock.
// Daniel Valvano
// February 21, 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 2.10, Figure 2.31

 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 SYSCTL_RIS_R            (*((volatile unsigned long *)0x400FE050))
#define SYSCTL_RIS_PLLLRIS      0x00000040  // PLL Lock Raw Interrupt Status
#define SYSCTL_RCC_R            (*((volatile unsigned long *)0x400FE060))
#define SYSCTL_RCC_SYSDIV_M     0x07800000  // System Clock Divisor
#define SYSCTL_RCC_SYSDIV_4     0x01800000  // System clock /4
#define SYSCTL_RCC_SYSDIV_5     0x02000000  // System clock /5
#define SYSCTL_RCC_SYSDIV_6     0x02800000  // System clock /6
#define SYSCTL_RCC_SYSDIV_7     0x03000000  // System clock /7
#define SYSCTL_RCC_SYSDIV_8     0x03800000  // System clock /8
#define SYSCTL_RCC_SYSDIV_9     0x04000000  // System clock /9
#define SYSCTL_RCC_SYSDIV_10    0x04800000  // System clock /10
#define SYSCTL_RCC_SYSDIV_11    0x05000000  // System clock /11
#define SYSCTL_RCC_SYSDIV_12    0x05800000  // System clock /12
#define SYSCTL_RCC_SYSDIV_13    0x06000000  // System clock /13
#define SYSCTL_RCC_SYSDIV_14    0x06800000  // System clock /14
#define SYSCTL_RCC_SYSDIV_15    0x07000000  // System clock /15
#define SYSCTL_RCC_SYSDIV_16    0x07800000  // System clock /16
#define SYSCTL_RCC_USESYSDIV    0x00400000  // Enable System Clock Divider
#define SYSCTL_RCC_PWRDN        0x00002000  // PLL Power Down
#define SYSCTL_RCC_OEN          0x00001000  // PLL Output Enable
#define SYSCTL_RCC_BYPASS       0x00000800  // PLL Bypass
#define SYSCTL_RCC_XTAL_M       0x000003C0  // Crystal Value
#define SYSCTL_RCC_XTAL_6MHZ    0x000002C0  // 6 MHz Crystal
#define SYSCTL_RCC_XTAL_8MHZ    0x00000380  // 8 MHz Crystal
#define SYSCTL_RCC_OSCSRC_M     0x00000030  // Oscillator Source
#define SYSCTL_RCC_OSCSRC_MAIN  0x00000000  // MOSC

// configure the system to get its clock from the PLL
void PLL_Init(void){
  // 1) bypass PLL and system clock divider while initializing
  SYSCTL_RCC_R |= SYSCTL_RCC_BYPASS;
  SYSCTL_RCC_R &= ~SYSCTL_RCC_USESYSDIV;
  // 2) select the crystal value and oscillator source
  SYSCTL_RCC_R &= ~SYSCTL_RCC_XTAL_M;   // clear XTAL field
  SYSCTL_RCC_R += SYSCTL_RCC_XTAL_8MHZ; // configure for 8 MHz crystal
  SYSCTL_RCC_R &= ~SYSCTL_RCC_OSCSRC_M; // clear oscillator source field
  SYSCTL_RCC_R += SYSCTL_RCC_OSCSRC_MAIN;// configure for main oscillator source
  // 3) activate PLL by clearing PWRDN and OEN
  SYSCTL_RCC_R &= ~(SYSCTL_RCC_PWRDN|SYSCTL_RCC_OEN);
  // 4) set the desired system divider and the USESYSDIV bit
  SYSCTL_RCC_R &= ~SYSCTL_RCC_SYSDIV_M; // system clock divider field
  SYSCTL_RCC_R += SYSCTL_RCC_SYSDIV_4;  // configure for 50 MHz clock
```

```
//  SYSCTL_RCC_R += SYSCTL_RCC_SYSDIV_5;  // configure for 40 MHz clock
//  SYSCTL_RCC_R += SYSCTL_RCC_SYSDIV_6;  // configure for 33.33 MHz clock
//  SYSCTL_RCC_R += SYSCTL_RCC_SYSDIV_7;  // configure for 28.57 MHz clock
//  SYSCTL_RCC_R += SYSCTL_RCC_SYSDIV_8;  // configure for 25 MHz clock
//  SYSCTL_RCC_R += SYSCTL_RCC_SYSDIV_9;  // configure for 22.22 MHz clock
//  SYSCTL_RCC_R += SYSCTL_RCC_SYSDIV_10; // configure for 20 MHz clock
//  SYSCTL_RCC_R += SYSCTL_RCC_SYSDIV_11; // configure for 18.18 MHz clock
//  SYSCTL_RCC_R += SYSCTL_RCC_SYSDIV_12; // configure for 16.67 MHz clock
//  SYSCTL_RCC_R += SYSCTL_RCC_SYSDIV_13; // configure for 15.38 MHz clock
//  SYSCTL_RCC_R += SYSCTL_RCC_SYSDIV_14; // configure for 14.29 MHz clock
//  SYSCTL_RCC_R += SYSCTL_RCC_SYSDIV_15; // configure for 13.33 MHz clock
//  SYSCTL_RCC_R += SYSCTL_RCC_SYSDIV_16; // configure for 12.5 MHz clock (default
setting)
  SYSCTL_RCC_R |= SYSCTL_RCC_USESYSDIV;
  // 5) wait for the PLL to lock by polling PLLLRIS
  while((SYSCTL_RIS_R&SYSCTL_RIS_PLLLRIS)==0){};
  // 6) enable use of PLL by clearing BYPASS
  SYSCTL_RCC_R &= ~SYSCTL_RCC_BYPASS;
}
```

Switches

```
// Dalton Altstaetter
// 2/11/14
// Switches.c
// Switches module for the stepper motor
// Meant for the LM3S1968

// Use PB4,PB5,PB7 for the input switches bc of their location on the board
// Use PA0,PA2,PA4,PA6 for the outputs bc of their location on the board


#include "lm3s1968.h"
#include "Switches.h"


/*static unsigned long DeterminePortX(volatile unsigned long* baseAddress)
{
	volatile unsigned long SYSCTL_RCGC2_GPIOX = 0;

	if(baseAddress == GPIO_PORTA_DATA_BITS_R)
	{
		SYSCTL_RCGC2_GPIOX = SYSCTL_RCGC2_GPIOA;
	}
	else if(baseAddress == GPIO_PORTB_DATA_BITS_R)
	{
		SYSCTL_RCGC2_GPIOX = SYSCTL_RCGC2_GPIOB;
	}
	else if(baseAddress == GPIO_PORTC_DATA_BITS_R)
	{
		SYSCTL_RCGC2_GPIOX = SYSCTL_RCGC2_GPIOC;
	}
	else if(baseAddress == GPIO_PORTD_DATA_BITS_R)
	{
		SYSCTL_RCGC2_GPIOX = SYSCTL_RCGC2_GPIOD;
	}
	else if(baseAddress == GPIO_PORTE_DATA_BITS_R)
	{
```

```
		SYSCTL_RCGC2_GPIOX = SYSCTL_RCGC2_GPIOE;
	}
	else if(baseAddress == GPIO_PORTF_DATA_BITS_R)
	{
		SYSCTL_RCGC2_GPIOX = SYSCTL_RCGC2_GPIOF;
	}
	else if(baseAddress == GPIO_PORTG_DATA_BITS_R)
	{
		SYSCTL_RCGC2_GPIOX = SYSCTL_RCGC2_GPIOG;
	}
	else if(baseAddress == GPIO_PORTH_DATA_BITS_R)
	{
		SYSCTL_RCGC2_GPIOX = SYSCTL_RCGC2_GPIOH;
	}

	return SYSCTL_RCGC2_GPIOX;
}
*/

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



void GPIO_PortX_Init(volatile unsigned long* baseAddress, unsigned long bits, unsigned
int input_output)
{
	// Let X be the letter of the port you wish to initialize
	// Enables PortX
	SYSCTL_RCGC2_R |= (SYSCTL_RCGC2_GPIOA + SYSCTL_RCGC2_GPIOB);
//DeterminePortX(baseAddress);
	Delay(100); // give it time to enable the port

	if(input_output == OUTPUT)
	{
		*(baseAddress+0x0400) |= bits; // makes them output pins
	}
	else
	{
		*(baseAddress+0x0400) &= ~bits; // makes them input pins
	}

	*(baseAddress+0x051C) |= bits;    // digital enabled
	*(baseAddress+0x0420) &= ~bits; // disable alternate function
}



void PortB_Init(void)
{
	SYSCTL_RCGC2_R |= SYSCTL_RCGC2_GPIOB;
```

```
        // dummy instructions to allow the port time to initialize
        Delay(100);

        GPIO_PORTB_DIR_R &= ~0x38;   // make PB3,4,5,7 inputs
        GPIO_PORTB_DEN_R |= 0x38;    // digital enable
        GPIO_PORTB_AFSEL_R &= ~0x38; // disable alternate function

        GPIO_PORTB_DIR_R |= 0x47;   // make PB0-3,6 inputs
        GPIO_PORTB_DEN_R |= 0x47;    // digital enable
        GPIO_PORTB_AFSEL_R &= ~0x47; // disable alternate function
}

void PortD_Init(void)
{
        SYSCTL_RCGC2_R |= SYSCTL_RCGC2_GPIOD;

        // dummy instructions to allow the port time to initialize
        Delay(100);

        GPIO_PORTD_DIR_R |= 0x0F;   // make outputs
        GPIO_PORTD_DEN_R |= 0x0F;    // digital enable
        GPIO_PORTD_AFSEL_R &= ~0x0F; // disable alternate function
}

void PortA_Init(void)
{
        SYSCTL_RCGC2_R |= SYSCTL_RCGC2_GPIOA;

        // dummy instructions to allow the port time to initialize
        Delay(100);

        GPIO_PORTA_DIR_R |= 0xAA;    // make PA1,3,5,7 outputs
        GPIO_PORTA_DIR_R &= ~0x55;    // make PA1,3,5,7 outputs
        GPIO_PORTA_DEN_R |= 0xFF;    // digital enable
        GPIO_PORTA_AFSEL_R &= ~0xFF; // disable alternate function
}

void GPIO_Init(void)
{
        //PortA_Init();
        PortB_Init();
        PortD_Init();
}
```

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/
 */

#include "FSM.h"
#include "lm3s1968.h"
#include "SysTick.h"

extern SM* StatePt; // Ptr to our FSM object which we will use to move between states
extern SM FSM[SIZE];   // our FSM object
extern unsigned int currentState;
extern unsigned int Switch3_Flag;

void SysTick_Wait(unsigned long delay);
void SysTick_Wait10ms(unsigned long delay);
void SysTick_Init(unsigned long reloadValue);
void Turn_Motor(const int Direction);
void PressSwitch1(void);
void PressSwitch2(void);
void PressSwitch3(void);
void PressSwitch12(unsigned long* count);
unsigned int GetButtonPress(void);


int cw_Switch1_Only;
int cw_Switch3_Only;
int ccw_Switch2_Only;
int cw_ccw_Switch_12;

void SysTick_Handler(void)
{
	static unsigned long count = 0;
	int data;
	int logic;
	DisableInterrupts();

	logic = GetButtonPress();

	// or switch statement
	switch(logic)
	{
		// need to add logic for when multiple ones are pressed.
		case 1:
			GPIO_PORTB_DATA_R |= 0x01;data = GPIO_PORTB_DATA_R;
```

```
                        PressSwitch1();
                        GPIO_PORTB_DATA_R &= ~0x01;data = GPIO_PORTB_DATA_R;
                        break;
                case 2:
                        GPIO_PORTB_DATA_R |= 0x02;data = GPIO_PORTB_DATA_R;
                        PressSwitch2();
                  GPIO_PORTB_DATA_R &= ~0x02;data = GPIO_PORTB_DATA_R;
                        break;
                case 3:
                        GPIO_PORTB_DATA_R |= 0x40;data = GPIO_PORTB_DATA_R;
                        PressSwitch3();
                  GPIO_PORTB_DATA_R &= ~0x40;data = GPIO_PORTB_DATA_R;
                        break;
                case 12:
                        GPIO_PORTB_DATA_R |= 0x03;data = GPIO_PORTB_DATA_R;
                        PressSwitch12(&count);
                  GPIO_PORTB_DATA_R &= ~0x03;data = GPIO_PORTB_DATA_R;
                        break;
                default:
                        break;
        }
        EnableInterrupts();
}

// 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;       // reload value ->___seconds
  NVIC_ST_CURRENT_R = 0;                // any write to current clears it
                                        // enable SysTick with core clock
  NVIC_ST_CTRL_R = NVIC_ST_CTRL_ENABLE+NVIC_ST_CTRL_CLK_SRC+NVIC_ST_CTRL_INTEN;
}
// Time delay using busy wait.
// 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);
}
// Time delay using busy wait.
// 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 Turn_Motor(const int Direction)
{
```

```
	GPIO_PORTD_DATA_R = FSM[currentState].output;
	SysTick_Wait10ms(FSM[currentState].delay);
	currentState = (currentState+SIZE+Direction)%SIZE; // keeps it in the range 0-3
}

void PressSwitch1(void)
{
	SysTick_Wait10ms(1); // for switch debouncing

		if(GetButtonPress() == 1)
		{
			Turn_Motor(CW);
  /*
			// send the current output through the ports
//			GPIO_PORTA_DATA_R = FSM[currentState].output;
//			SysTick_Wait10ms(FSM[currentState].delay);
//			currentState = (currentState+1)%SIZE; // keeps it in the range 0-3
//			// update to the next state for the next interrupt
	*/
		}
}


void PressSwitch2(void)
{
	SysTick_Wait10ms(1); // for switch debouncing
	if(GetButtonPress() == 2)
	{

/*
		// send the current output through the ports
		GPIO_PORTA_DATA_R = FSM[currentState].output;
		SysTick_Wait10ms(FSM[currentState].delay);
		// keeps it in the range 0-3, adding SIZE to the
		// currentState keeps  from modding negative numbers.
		currentState = ((currentState+SIZE)-1)%SIZE;
		// update to the next state for the next interrupt
*/
		Turn_Motor(CCW);
	}
}

void PressSwitch3(void)
{
	SysTick_Wait10ms(1); // for switch debouncing
		if(GetButtonPress() == 3)
		{
	/*		if(Switch3_Flag)
			{
				// do logic so that it waits to see that u release the switch
				// maybe use an interrupt that looks for the low-triggered
interrupt
				// and changes the flag back to zero
				// what I want to do is enable an interrupt on switch3 pin
when
				// I jump into this elif that then looks for the falling edge
of
				// the switch3 pin and then sets Switch3_Flag=0
```

```
                        // what I will do instead is polling
                        return;

                }

                // send the current output through the ports
                GPIO_PORTA_DATA_R = FSM[currentState].output;
                SysTick_Wait10ms(FSM[currentState].delay);
                currentState = (currentState+1)%SIZE; // keeps it in the range 0-3
        */

                Turn_Motor(CW);
                // poll until the switch is released
                while(GetButtonPress() == 3)
                {}
        }
}


void PressSwitch12(unsigned long* countPtr)
{
        SysTick_Wait10ms(1); // for switch debouncing
        if(GetButtonPress() == 12)
        {
                if((*countPtr) < 8)
                {
                        // rotate CW
                        Turn_Motor(CW);
                }
                else if((*countPtr) < 16)
                {
                        // rotate CCW
                        Turn_Motor(CCW);
                }
                (*countPtr)++;
                *(countPtr) %= 16; // keep count between 0-15
        }
}

unsigned int GetButtonPress(void)
{
        int s1,s2,s3;

        s1 = GPIO_PORTB_DATA_R & 0x10;
        s2 = GPIO_PORTB_DATA_R & 0x20;
        s3 = GPIO_PORTB_DATA_R & 0x08;

        if(s1 && s2 && !s3)
        {
                return 12;
        }
        else if(s1 && !s3 && !s2)
        {
                return 1;
        }
        else if(s2 && !s3 && !s1)
        {
                return 2;
```

```
	}
	else if(s3 && !s1 && !s2)
	{
		return 3;
	}
	else
	{
		return 0;
	}
}
```

Main

```
// SysTickTestMain.c
// Runs on LM3S1968
// Test the SysTick functions by activating the PLL, initializing the
// SysTick timer, and flashing an LED at a constant rate.
// 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/
 */

// PG2 is an output for debugging

#include "SysTick.h"
#include "PLL.h"
#include "lm3s1968.h"
#include "Switches.h"
#include "FSM.h"

//-------- All Global Variables are Here-----------------------
//order for next states are: clockwise, counterclockwise
//SM FSM[SIZE] = {
//     { 0x21, 10, {six, nine}},
//     { 0x81, 10, {ten, five}},
//     { 0x84, 10, {nine, six}},
//     { 0x24, 10, {five, ten}}
//};


//     { 0x21, 10, {six, nine}},
//     { 0x24, 10, {ten, five}},
//     { 0x84, 10, {nine, six}},
//     { 0x81, 10, {five, ten}}
```

```
#define DELAY 1
SM FSM[SIZE] = {
       { 0x05, DELAY, {six, nine}},
       { 0x06, DELAY, {ten, five}},
       { 0x0A, DELAY, {nine, six}},
       { 0x09, DELAY, {five, ten}}
};

//SM FSM[SIZE] = {
//  { 0x05, 4, {six, nine}},
//     { 0x06, 4, {ten, five}},
//     { 0x0A, 4, {nine, six}},
//     { 0x09, 4, {five, ten}}
//};



SM *StatePt;
unsigned long currentState;
unsigned int Switch3_Flag;

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

void FSM_Init(void)
{
       // initializes FSM
       return;
}

int main(void)
{
       DisableInterrupts();
       //GPIO_PortX_Init(GPIO_PORTA_DATA_BITS_R,0xAA,OUTPUT);
       //GPIO_PortX_Init(GPIO_PORTB_DATA_BITS_R,0xB0,INPUT);
       GPIO_Init();

  SYSCTL_RCGC2_R |= SYSCTL_RCGC2_GPIOG; // activate port G
  PLL_Init();               // set system clock to 50 MHz
  //SysTick_Init(755000);    // Set interrrupt time at 10Hz. 1e6 => 2 milliseconds =>
500Hz
         SysTick_Init(600000);    // Set interrrupt time at 10Hz. 1e6 => 2 milliseconds
=> 500Hz




       currentState = 0; // begin with output 5.
       //Switch3_Flag = 0;

       GPIO_PORTD_DATA_R = FSM[currentState].output;
       //  GPIO_PORTG_DIR_R |= 0x04; // make PG2 out (built-in LED)
//  GPIO_PORTG_AFSEL_R &= ~0x04;// disable alt funct on PG2
//  GPIO_PORTG_DEN_R |= 0x04; // enable digital I/O on PG2
       EnableInterrupts();
  while(1)
       {
              // wait for periodic SysTick interrupt
```

```
            // All the action of this program will occure in the SysTick Interrupt
    // GPIO_PORTG_DATA_R = GPIO_PORTG_DATA_R^0x04; // toggle PG2

  }
}
```