

Oscar

	Report Section	%	E	S	D	U	
1	Introduction, background, purpose etc.	5		x			3.8
2	Identify and Formulate problem/design goals (1a)	10	x				10.0
3	Develop design/solution (1b)	15	x				15.0
4	Acquire background knowledge (e.g., Data-sheets, papers, etc.) (7a)	10	x				10.0
5	Describe design judgements regarding global, economic, environmental, and societal contexts (4b)	10			x		5.0
6	Conduct Experiments, acquire data* (6a)	15	x				15.0
7	Conduct Simulations, make predictions* (6b)	10			x		5.0
8	Interpret Results, draw inferences* (6c)	10	x				10.0
9	Iterate, based on tests, simulations, results to improve design	10	x				10.0
10	Summary/Conclusion	5			x		2.5
	Total	100					86.3
	20-pt scale						17.0

Anything you could do to bump these up, even a sentence or two, would be helpful.

Report

May 7, 2024

1 Lab 3 Report

This lab is about motors

The first motor implemented is the PWM-controlled microservo. I used the PSoC ADC to implement the joystick from the last lab here. The joystick is read and used to control the compare value in a PWM component to vary the pulse width from 1 to 2 milliseconds.

This is intended to correspond to 0-180 degrees of rotation on the servo, though the servo is cheap and this resulted in a strange range of motion. After adjusting the scaling and offset values of the ADC readings, I was able to get the range closer to 0-180 degrees. The physical deadzone on the joystick also meant that the servo would not start moving until the joystick was moved past a certain point in either direction. After adjustments, the pulse width ranged from 0.75ms to 2.65ms. These values achieved a 180 degree range of motion.





180 degrees - 2.65ms:

The ADC values being used to vary the PWM compare value correspond to approximately 0.1 ADC units per degree.

The video demo is [here](#)

Controlling the stepper motor

Using the EasyDriver from sparkfun, the PSoC only needs to send step and direction signals to control the motor. The direction signal is simpler and is 1 or 0 to indicate the direction for the motor to spin. The actual direction depends on how the motor is connected to the driver board. The step signal is a pulse signal that tells the driver to step the motor once. The delay between pulses controls the speed that the motor turns. I want to control the stepper motor with the joystick's vertical axis, so I need to determine how the changing ADC values should correspond to the direction and speed. The higher absolute values should correspond to a higher speed. Positive and negative from the neutral position will decide the direction the motor turns


To control the speed of the motor, I decided to use a PWM module and use the APIs to change the clock divider value going into the pwm. I use the PWM component to generate the pulse signal the stepper needs, but do not modulate the pulse-width. I change the delay between pulses by changing the divider on the clock input.

The reading of the vertical axis on the joystick is used to divide the clock divider value, which results in a range of values from 0 to 100. The direction is controlled simply by the sign of the ADC reading and used to update the direction pin's logic value.

Stepper speed: From the data sheet ([Datasheets/ST-PM35-15-11C.pdf](#)), the stepper motor has a stride angle of 7.5° , meaning there are $360/7.5 = 48$ steps per revolution. My program runs the pulses at up to 2KHz which should correspond to $2000/48$ or about 42.7 revolutions per second. In reality, I measured approximately 5.5 revolutions per second. On another run, I got 5.25 rev/sec. Based on this, the steps per revolution actually comes out to between 360 and 380. ($2000 \text{ steps/sec} / 5.5 \text{ rev/sec}$). This suggests that the driver is microstepping the motor at about

1/8 speed. This is, after reviewing the hookup guide for the SparkFun EasyDriver, the default configuration for the driver without modifying the MS1 and MS2 pins.


Truth Table defining microstepping configuration.



MS1	MS2	Microstep Resolution
L	L	Full Step (2 Phase)
H	L	Half Step
L	H	Quarter Step
H	H	Eighth Step (Default configuration)


Controlling the DC motor

Using the SparkFun TB6612FNG, a full H-Bridge motor driver, the goal here is to control the speed of a DC motor with PWM and use logic to control the direction. Another goal is to measure the output speed of the motor, which can be accomplished using a Hall Effect sensor and mounted magnet.



To control the speed of the motor, the joystick position is read using the PSoC's ADC. The absolute value of the ADC reading is scaled to -20k to 20k and used to set the compare value of the PWM, which in turn sets the duty cycle of the motor. The direction is obtained from the sign of the scaled reading and used to set the H-Bridge pins on the motor driver. When the value is near 0 (the joystick is neutral), the driver is set to brake.

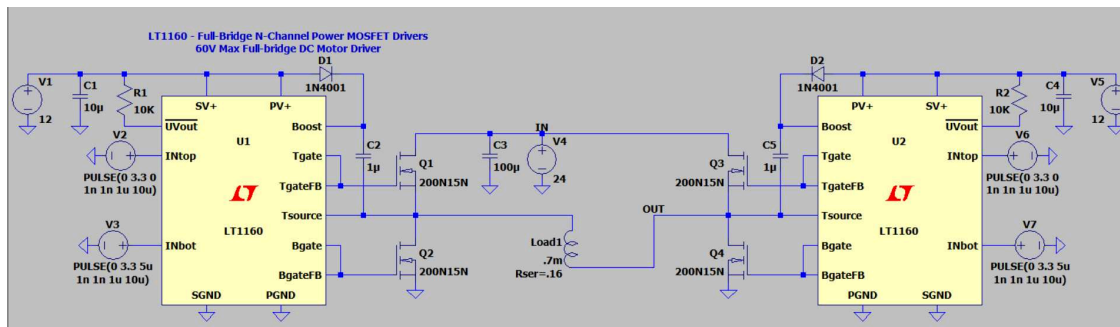
To measure the speed of the motor, a hall effect sensor is attached to the motor so it can detect a magnet attached to the output shaft of the motor. The hall effect sensor uses an open collector, so it needs to be pulled up to logic voltage before connecting it to the oscilloscope. On the oscilloscope, the frequency outputted by the hall effect sensor can be multiplied to convert to revolutions per minute of the motor's output shaft. At 100% duty cycle, the motor turns at approximately 180 rpm.



There are demo videos for the DC Motor PWM, Hall sensor output, and visual demo of the system in the media folder

Controlling the Large DC Motor

The large DC motor runs on 36V power. This portion will use two LT1160 half-bridge MOSFET Drivers to drive the motor in a full H-bridge configuration. Analog Devices provides an example schematic showing an implementation of the drivers in a half-bridge configuration. I opened this in LTSpice and modified the schematic to create the full-bridge configuration below.



In PSoC Creator, my program reads values from a potentiometer joystick and uses that to set the duty cycle on a PWM signal. This signal is shared to two multiplexers that send the signal to any of 4 combinations of the top and bottom inputs on the 1160s.

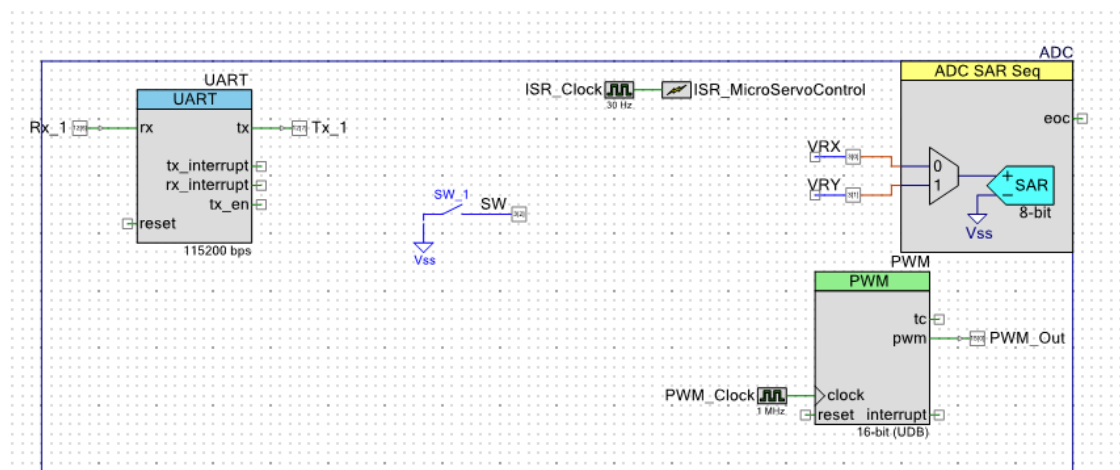
Issues: With the whole system running at 12V, turning the connected motor in one direction would bring it to full speed for a moment before the drivers shut down until the motor stops. Turning the motor in the other direction, the system would be able to sustain its full speed continuously.

Turning the motor in either direction creates significant heat in the LT1160 drivers. When turning the motor in the same direction that cuts out, the driver with the top gate active heats up very rapidly. While testing, one of the 1160 drivers seems to have been damaged by the heat and drew much more current than expected. Replacing the driver appeared to fix the problem.

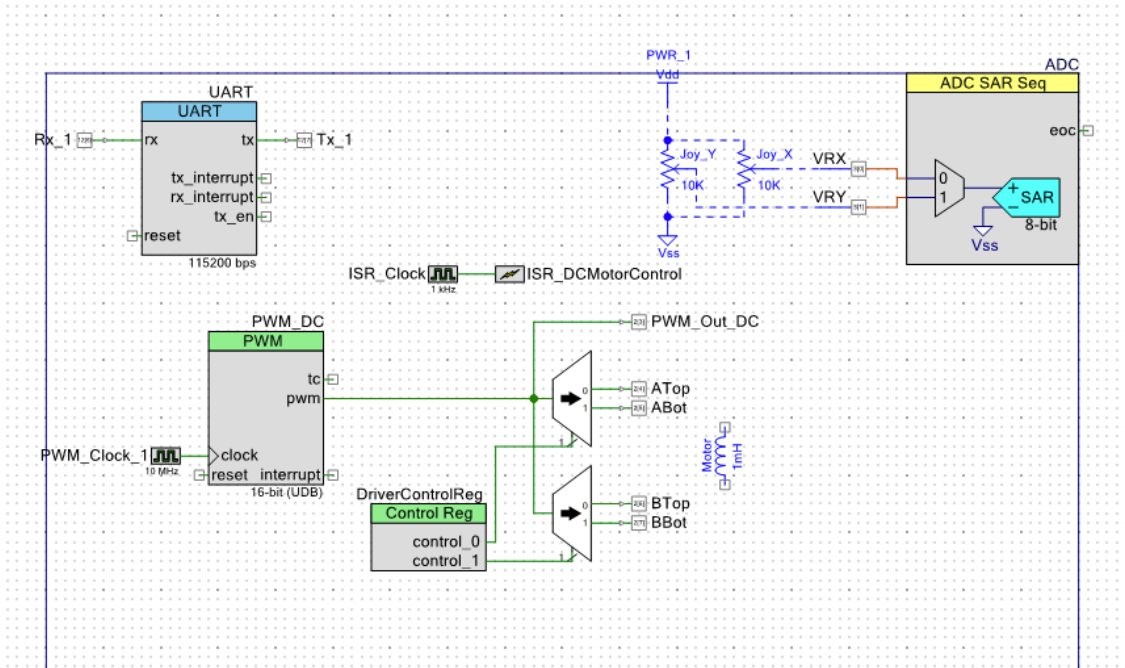
2 Appendix Schematics

Did you check the gates involved in this direction? Were they getting reliable drive?

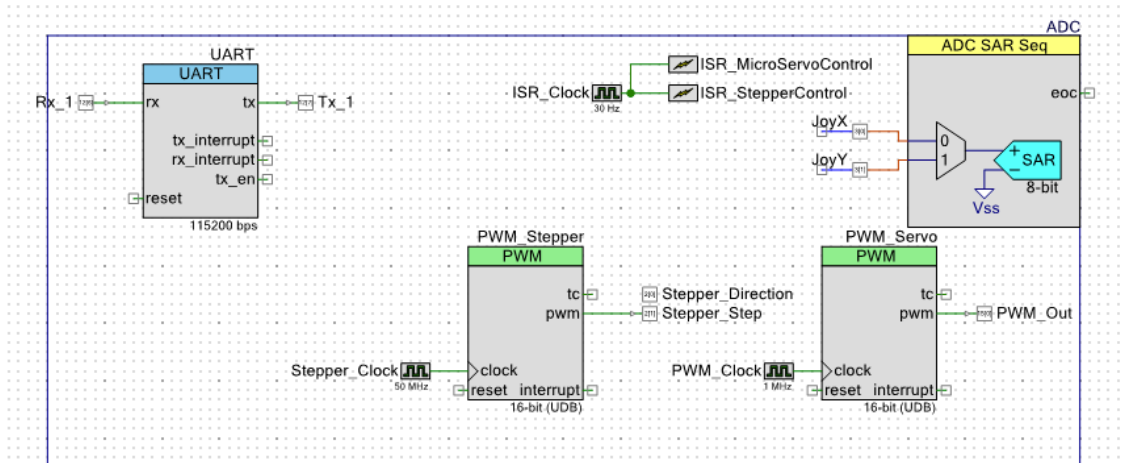
Joystick Servo Motor Interface:



Joystick DC Motor Interface:



Servo-Stepper Combined:



3 Appendix: Code

DC Motor Control

```
#include <project.h>
#include <stdio.h>
#include <stdlib.h>
```

```
double temp = 0;
int range = 450;
```

```

CY_ISR(DCMotorControl)
{
    ADC_StartConvert();
    ADC_IsEndConversion(ADC_WAIT_FOR_RESULT);
    temp = (ADC_GetResult16(0)/128.0)*range-range; //reading channel zero, horizontal
    //temp should range from -450 to +450

    //Set PWM, 475 being ~95% duty cycle since the drivers are limited by the charge pump
    PWM_DC_WriteCompare(abs((int)temp));

    //Set direction based on sign of temp
    if(temp > 15){
        //CCW
        DriverControlReg_Write(1);

    }else if(temp < -15){
        //CW
        DriverControlReg_Write(2);
    }else{
        /*//Brake, both grounded FETs activated... Don't do this, the source goes into over-vol
        DriverControlReg_Write(3);
        PWM_DC_WriteCompare(range);
        */
        DriverControlReg_Write(1);
        PWM_DC_WriteCompare(0);
    }
    //Don't set DriverControlReg to 0, that activates both top FETs connecting 12v to 12v.
}

CY_ISR(DCMotorControl);

int main()
{
    CYGlobalIntEnable;                                     /* Enable Global Interrupts */

    PWM_DC_Start();
    ADC_Start();
    UART_Start();
    UART_PutString("UART Open\n");
    char buffer[100];
    ISR_DCMotorControl_StartEx(DCMotorControl);

    for(;;)

```



```

    {

        sprintf(buffer, "temp:%i\n", (int)temp);
        UART_PutString(buffer);

        CyDelay(1);
    }

}

Micro Servo Control

#include <project.h>
#include <stdio.h>

static uint default_compare = 18500; //should be 0 degrees on the servo
double temp = 0;

CY_ISR(MicroServoControl)
{
    ADC_StartConvert();
    ADC_IsEndConversion(ADC_WAIT_FOR_RESULT);
    temp = (ADC_GetResult16(0)/127.0)*950-1150; //reading channel zero, horizontal
    //temp values rang from -1150 to 757, which correspond pretty closely to 0 to 180 degrees

    PWM_WriteCompare(default_compare+temp);
}

CY_ISR(MicroServoControl);

int main()
{
    CYGlobalIntEnable;                                     /* Enable Global Interrupts */

    PWM_Start();
    ADC_Start();
    UART_Start();
    UART_PutString("UART Open\n");
    char buffer[100];
    ISR_MicroServoControl_StartEx(MicroServoControl);

```



```

        for(;;)
        {

            sprintf(buffer, "temp:%i\n", (int)temp);
            UART_PutString(buffer);

            CyDelay(1);
        }
    }

Servo And Stepper

/*****
* File Name: main.c
*
* Description: Joystick control for microservo and stepper motor
*
*****/

#include <project.h>
#include <stdio.h>
#include <stdlib.h>

static uint default_compare = 18500; //should be 0 degrees on the servo
double temp = 0;
char buffer[100];
uint Stepper_Direction;

CY_ISR(MicroServoControl)
{
    ADC_StartConvert();
    ADC_IsEndConversion(ADC_WAIT_FOR_RESULT);
    temp = (ADC_GetResult16(0)/127.0)*950-1150; //reading channel zero, horizontal
    //temp values rang from -1150 to 757, which correspond pretty closely to 0 to 180 degrees

    PWM_Servo_WriteCompare(default_compare+temp);
}

CY_ISR(StepperControl)
{
    ADC_StartConvert();
    ADC_IsEndConversion(ADC_WAIT_FOR_RESULT);
    temp = (ADC_GetResult16(1)/127.0)*75-75; //Reading channel one, vertical
    //temp values should range from -100 to 100 +/- 1.
    //instead of pwm control it should be delay control.

```

```

//change timer delay depending on temp value
//delay has a minimum of 500 microseconds. to go from 0 to 100% speed,

//Set speed based on magnitude of temp
//Accomplished by dividing the clock going to the PWM for the stepper
//from 50 MHz (0.3ms) to 0.5 MHz (30ms), divide clock by up to 100.
Stepper_Clock_SetDivider(100/(abs((int)temp+1)));

//disable stepping if joystick is neutral
if((int)temp == 0){
    Stepper_Clock_StopBlock();
}
else{
    Stepper_Clock_Start();
}
//Set direction based on sign of temp
if(temp > 0){
    Stepper_Direction_Write(1);
}else{
    Stepper_Direction_Write(0);
}
sprintf(buffer, "Stepper temp:%i\tdivider: %u\n", (int)temp, (100/abs((int)temp)+1));
    UART_PutString(buffer);

}

CY_ISR(StepperControl);

CY_ISR(MicroServoControl);


int main()
{
    CYGlobalIntEnable;                                     /* Enable Global Interrupts */

    PWM_Stepper_Start();
    PWM_Servo_Start();
    ADC_Start();
    UART_Start();
    UART_PutString("UART Open\n");

    ISR_MicroServoControl_StartEx(MicroServoControl);
    ISR_StepperControl_StartEx(StepperControl);
}

```

```
for(;;)
{

    sprintf(buffer, "temp:%i\n", (int)temp);
    //UART_PutString(buffer);

    CyDelay(1);
}

}

/* End of File */
```