[Ricardo Stefano Reyna]
[08/July/2016]
[21010521]

Stepper Motor Module Report

Introduction

In this module we were introduced with the 28byj-48 stepper motor, and ULN2003A transistor array. Where we connected the setup to our microcontroller, read the frequency of a function generator with the timer, and the voltage using the ADC. After gather the values respectively we displayed said values into the left and right stepper motors to simulate a speedometer and a fuel reader.

Design

The way I proceeded to connect the stepper motor to the microcontroller was connecting the 4 inputs of the left and right motor to pins 4.7 down to 4.0 for convenience. Then I set up a timer interrupt at 16Mhz, and made a grey counter array in order to move based on the steps it takes where iterating forward makes it go clockwise, and from 7 down to 0 will go counter clockwise. After getting the steps done, I proceeded to read an input using the laser beam which is using the opto2 and opto1 pins. I connected them to pins 1.1 and 1.0 respectively. I used the laser as a send each motor home as a form of calibration which was simple, but with trial and error I was able to use a for loop from 0 to 370 to set the speedometer to 0 going clockwise and the fuel gauge to 10 going the other way. Then I proceed making a few functions such as move motor left or right, and tick. These two functions are the whole base of the module since I call them for most other scenarios. The way it works is tick takes in the new positon, the current position, and a flag to choose the motor. Based on the distance (new position – current position) I can get how many ticks, or how long my delay should be, and the direction I want the gauge to go. Based on the flag I'll just call the respective motor function passing in the direction and let it iterate depending on the delay. After giving some test values and checking if it worked I proceed to read the function generator and ADC which I connected to A1 and A0. The timer interrupt and adc interrupt allows me to get each value as I read the frequency and the voltage based on a few flags. For the frequency since the highest value I'll read is 100Hz, I'd figure to sample at 200Hz. I read all the values putting it into a size 200 array where after I'm done filling the array I'll iterate checking for high and lows to get the edges, thus the amount of edges gives me the frequency (or new position). For the adc since the values are going to range from 0 to 1023 I just divided the value I got by 10. Since the adc was giving me trouble I figured to capture around 20 values put them into an array, grab

the minimum since it's the closest to the real value and do the same as before only that this time just consider values that are plus or minus 50 the minimum, then grab the average and output said value. After testing and tweaking the results was close enough to what I wanted, but the fuel sometimes was still fuzzy which I think is because is not getting the enough amount of power, but it worked most of the time. All needed left was adding the LED to read when it was less than 10% which was connected to pin 3.0 and inside my fuel function I just check when the average is less than 13 (it was off by 2 usually).

Conclusion

The way the stepper motor works is by using a half step method which is why the array of 8. This holds a grey code that only one bit changes in each step, thus no two bits can change which I why that particular order. Each bit represents the wires of the motor, thus the sequence is: in4; in4 and in3; in3; in3 and in2; in2; in2 and in1; in1; and in1 and back to in4. That is why the whole project relies on the two methods mentioned above because they are the ones that make the stepper motor work since all you need is the position you want the gauge to be and which of the two motors. No matter what the user wants to do depending on what the task is the base is already done.

Appendix

```
#include <msp430.h>

#include <stdlib.h>


/*

 * main.c

 */


#define LEFT 1

#define RIGHT 0


void timer_init();

void delay();
```

```
void moveLeft(int flag);

void moveRight(int flag);

void home_Left();

void home_Right();

void speed_start();

void fuel_start();

void tick(int amount, int place, int motor);

void adc_setup();

void go();

int get_fuel();

int frequency_value();


unsigned int speed;

int edges;

int index = 0;

int indL = 0;

int indR = 7;

unsigned int positionLeft = 0;

unsigned int positionRight = 100;

volatile unsigned int StoredCount = 0;

static unsigned int adc_value;

int speed_array[200];

int fu_array1[20];

int fu_array2[20];

//flags start
```

```
static unsigned int busy_flag = 0;
int fu_i;
int fu_in;
int fu_flag = 0;
//flags end
int min = 1023;
int new_val = 0;
unsigned int fuel;
unsigned int stepL[8] = {0x01, 0x03, 0x02, 0x06, 0x04, 0x0C, 0x08, 0x09}; // Step sequence
unsigned int stepR[8] = {0x10, 0x30, 0x20, 0x60, 0x40, 0xC0, 0x80, 0x90};

int main(void) {
   WDTCTL = WDTPW | WDTHOLD;         // Stop watchdog timer
   DCOCTL = CALDCO_16MHZ; //Set DCO step + modulation
   BCSCTL1 = CALBC1_16MHZ; //Set range

   timer_init();
   adc_setup();

         P4DIR = 0xFF; // Allow P4 pins 0 to 7 to output
         P3DIR = 0x01;
         P3OUT = 0x00;

         home_Left();
         home_Right();
```

```
	speed_start();
	fuel_start();
	busy_flag = 1;
	//tick(15, positionLeft, LEFT);
	int temp = 0;
	while(1) {
		go();
	}

	return 0;
}


void moveLeft(int flag) {
	P4OUT = stepL[indL];
	//clockwise
	if (flag) {
		indL++;
		if(indL == 8)
			indL = 0;
	}
	//counter clockwise
	else {
		indL--;
		if(indL == -1)
```

```
                indL = 7;
        }
        __bis_SR_register(LPM0_bits + GIE);
}

void moveRight(int flag) {
        P4OUT = stepR[indR];
        if (flag) {
                indR++;
                if(indR == 8)
                        indR = 0;
        }
        else {
                indR--;
                if(indR == -1)
                        indR = 7;
        }
        __bis_SR_register(LPM0_bits + GIE);
}

void home_Left() {
        while(1) {
                int temp = P1IN;
                temp &= 0x1;
                if (temp) {
```

```
                TACCR0 = 0;
                break;
            }
            else {
                TACCR0 = 25000;
                moveLeft(1);
            }
        }
}

void home_Right() {
        while(1) {
            int temp = P1IN;
            temp &= 0x2;
            if (temp) {
                TACCR0 = 0;
                break;
            }
            else {
                TACCR0 = 25000;
                moveRight(0);
            }
        }
}
```

```
void tick(int amount, int place, int motor) {
	int iterate = 0;
	if (amount < 0) {
		amount = 0;
	}

	if (amount > 100) {
		amount = 100;
	}
	//check absolut value between current position and new position
	int distance = amount - place;
	int direction = 1;
	if (amount < place)
		direction = 0;

	distance = abs(distance);

	while (iterate < distance) {
		unsigned int i = 0;
		unsigned int end = 0;
		//Making the motor exact
		if (distance < 70) {
			end = 35;
		}
		else {
```

```
                end = 34;
            }
            for (i = 0; i < end; i++)
            {
                TACCR0 = 25000;
                if (motor)
                    moveLeft(direction);
                else
                    moveRight(direction);
            }
            iterate++;
        }
}

void speed_start() {
        unsigned int i = 0;
        for (i = 0; i < 370; i++)
        {
            TACCR0 = 25000;
            moveLeft(1);
        }
}

void fuel_start() {
        unsigned int i = 0;
```

```
                for (i = 0; i < 370; i++)
                {
                        TACCR0 = 25000;
                        moveRight(0);
                }
}

void timer_init() {
        TACCTL0 = CCIE;
        TACCR0 = 25000;
        TACTL = TASSEL_2 + MC_1 + TACLR;
}

void go() {
        TACCR0 = 40250;
        busy_flag = 1;
        __bis_SR_register(LPM0_bits + GIE);
}

#pragma vector = TIMERA0_VECTOR
__interrupt void TIMERA0_ISR(void){
        if (busy_flag) {
                busy_flag = 0;
                ADC10CTL0 |= ENC + ADC10SC;           // conversion start
                __bis_SR_register(CPUOFF + GIE);      // enter the LPM
```

```
                ADC10CTL0 |= ENC + ADC10SC;           // conversion start
                __bis_SR_register(CPUOFF + GIE);      // enter the LPM



                frequency_value();
                get_fuel();


                busy_flag = 1;
        }
        __bic_SR_register_on_exit(CPUOFF);
}


// ADC10 interrupt service routine
#pragma vector=ADC10_VECTOR
__interrupt void ADC10_ISR(void) {
        adc_value = ADC10MEM;                                                   // save the value
 __bic_SR_register_on_exit(CPUOFF);      // Clear CPUOFF bit from 0(SR)
}


void adc_setup(void) {
        // ADC
        ADC10CTL1 = INCH_2 + CONSEQ_1;
        ADC10CTL0 = ADC10SHT_2 + ADC10ON + ADC10IE + MSC; //0x1018, 16xADC10CLks, ADC10 on,
and ADC10 interupt enable
        ADC10AE0 |= 0x03; //Enable reg 0, Pin2.0
}
```

```
int get_fuel(void) {
    TACCR0 = 0;
    fuel = adc_value;
    fuel /= 10;
    //this is to find the minimum since it's more accurate
    if (index % 5 == 0) {
        if (fu_flag == 0) {
            fu_array1[fu_i++] = fuel;
            if (fu_i == 20) {
                int i;
                for (i = 0; i < 20; i++)
                {
                    if (fu_array1[i] < min)
                    {
                        min = fu_array1[i];
                    }
                }
                fu_flag = 1;
                fu_i = 0;
            }
        }

        //this is to get the average on the values that resembles the min
        if (fu_flag) {
```

```
        if (fuel-min < 50)
                fu_array2[fu_in++] = fuel;
        if(fu_in == 20) {
                int i;
                        int sum = 0;
                for (i = 0; i < 20; ++i)
                {
                        sum += fu_array2[i];
                }
                        int avg = sum / 20;
                        if (avg < 13) {
                                P3OUT = 0X01;
                        }

                        else {
                                P3OUT = 0x00;
                        }
                tick(avg, positionRight, RIGHT);
                positionRight = avg;
                fu_flag = 0;
                min = 1023;
                fu_in = 0;
        }
}
```

```
    }
        TACCR0 = 40250;
}

int frequency_value() {
        TACCR0 = 0;
        speed = adc_value;
        edges = 0;
        speed_array[index++] = speed;
        if (index == 200) {
                index = 0;
                int i;
                for (i = 0; i < 200; i++)
                {
                        if (abs(speed_array[i+1]-speed_array[i]) > 0xFF) {
                                edges++;
                        }
                }
                if (edges < 0)
                        edges = 0;
                if (edges > 100)
                        edges = 100;

                new_val = edges - 3;
```

```
            tick(new_val, positionLeft, LEFT);
            positionLeft = new_val;
        }
        TACCR0 = 40250;
}
```