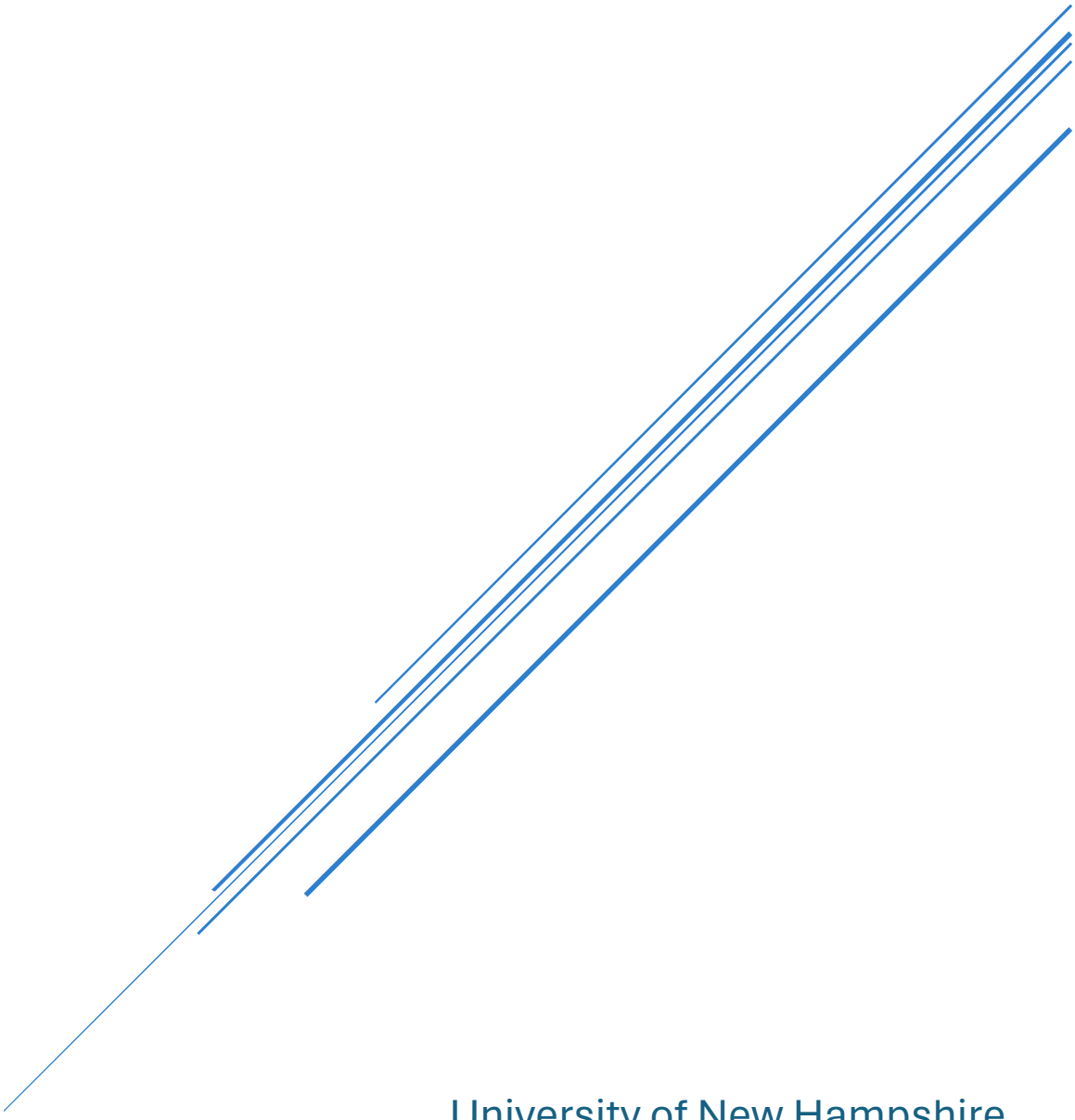
FINAL PROJECT: HALL SENSOR ARRAY

Nicholas Snyder



University of New Hampshire
ECE 649/796

Objectives:

The purpose of my final project was to create a hall-effect sensor array with the goal of measuring the speed and direction of a spinning object. To do this, I designed a 3D printed wheel and base plate for the sensors. I embedded magnets into the wheel and super glued two hell-effect sensors on the base plate close to the rotating magnets. Both the wheel and base plate were designed in SolidWorks and printed in the Makerspace.

Background:

I came up with this idea from studying electric motors. Many consumer brushless motors have some kind of sensor to determine the position of the rotor at all times. The most common method is with a hall sensor array. This method uses the passing magnetic field of the magnets embedded within the rotor to determine the speed of the motor. I thought it to be interesting to create my own sensor array with the knowledge of hall sensors and the MSP430 microcontroller. The outputs of the hall sensors would serve as port interrupts and the time interval between two sensors would determine the speed of the rotor.

Analysis:

I designed a spinning top to resemble the rotor of a motor. The hall sensors were mounted to the equivalent of the stator. In my code, the LCD was able to display the speed and direction based on the order of the hall sensor port interrupts and the time interval between the interrupts.

Challenges:

Challenges were very prevalent in this project. Firstly, the first print failed and the second lifted on one side and curled up which made inserting magnets more difficult. Secondly, I believe I bought the wrong hall effect sensors. This is because while verifying my design with an oscilloscope, I never got more than a few millivolts when passing a magnet by the sensor. This hurdle prevented me from testing my code because the output from the sensor was too low to be read as high from the MSP430.

Appendix:

Main.c:

```
#include <msp430.h>
#include "driverlib.h"
#include "myGpioLab3.h"
#include "myClocks.h"
#include "myLcdLab3.h"

#define HALLA BIT2
#define HALLB BIT3
#define redLED BIT0
```

```
#define greenLED BIT7

#define POLES 6
#define CIRCUMFERENCE 314

char direction;
char newHall = 'A';
char oldHall = 'A';
int speed;

void update_LCD(void);

int main(void)
{
    WDTCTL = WDTPW | WDTHOLD;    // stop watchdog timer
    PM5CTL0 &= ~LOCKLPM5; // Enable the GPIO pins

    initClocks();   // Required for the LCD
    myLCD_init();   // Required for the LCD

    // HALL. P4.2
    P4DIR &= ~BIT2; // Input
    P4REN |= BIT2;  // Resistors enabled
    P4OUT &= ~BIT2; // Pull-up enabled
    P4SEL1 &= ~BIT2; // Primary selected
    P4SEL0 |= BIT2; // Primary selected
    P4IES &= ~BIT2; // Low-to-high transition
    P4IE |= BIT2;   // Port interrupt enabled
    P4IFG &= ~BIT2; // Clear interrupt

    // HALL. P4.3
    P4DIR &= ~BIT3; // Input
    P4REN |= BIT3;  // Resistors enabled
    P4OUT &= ~BIT3; // Pull-up enabled
    P4SEL1 &= ~BIT3; // Primary selected
    P4SEL0 |= BIT3; // Primary selected
    P4IES &= ~BIT3; // Low-to-high transition
    P4IE |= BIT3;   // Port interrupt enabled
    P4IFG &= ~BIT3; // Clear interrupt

    // RED LED. P1.0
    P1DIR |= BIT1;  // Output
    P1REN &= ~BIT1; // Resistors disabled
    P1OUT &= ~BIT1; // Start low

    // GREEN LED. P9.7
    P9DIR |= BIT7;  // Output
    P9REN &= ~BIT7; // Resistors disabled
    P9OUT &= ~BIT7; // Start low

    // Timer_A: ACLK, divide by 1, continuous mode, clear TAR (leaves TAIE=0)
    TA1CTL = TASSEL_1 | ID_0 | MC_1 | TACLR;

    /**
     * define number of poles
     * define circumference
     *
     * wait for first port interrupt
     * start timer
     * wait for second port interrupt
     * calculate time between interrupts
     *
     * show speed based on time
     * show direction based on order of interrupts
```

```
     */

    _enable_interrupts();

    return 0;
}

#pragma vector = PORT4_VECTOR
__interrupt void PORT4_ISR(void)
{
    oldHall = newHall;

    // if Hall A is sensed
    if (P4IN == HALLA)
    {
        P4IFG &= ~HALLA; // Clear interrupt
        newHall = 'A';

        // flip green
        P9OUT ^= greenLED;
    }

    // if Hall B is sensed
    else
    {
        P4IFG &= ~HALLB; // Clear interrupt
        newHall = 'B';

        // flip red
        P1OUT ^= redLED;
    }

    if (newHall != oldHall)
    {
        // somehow determine speed from timer
        speed = (CIRCUMFERENCE / POLES) / (TA1R / 32768);
    }
    else
    {
        speed = 0;
    }

    // reset timer
    TA1CTL |= TACLR;
    update_LCD();
}

// if timer reaches the interrupt, speed too low
#pragma vector = TIMER0_A0_VECTOR
__interrupt void T1A0_ISR()
{
    // clears the flag (CCIFG in TA1CCTL0)
    TA1CCTL0 &= ~CCIFG;

    speed = 0;
    update_LCD();
}

void update_LCD(void)
{
    char speed_c;
    direction = newHall;

    if (direction == 'A')
```

```
        {
            myLCD_showChar('A', 1);
        }
        else
        {
            myLCD_showChar('B', 1);
        }

        // 1 2 m / s
        if (speed >= 10)
        {
            int speed_temp = speed % 10;
            speed_c = speed_temp - '0';
            myLCD_showChar(speed_c, 2);
        }
        else
        {
            myLCD_showChar(' ', 2);
        }
        speed_c = speed - '0';
        myLCD_showChar(speed_c, 3);

        myLCD_showChar('m', 4);
        myLCD_showChar('/', 5);
        myLCD_showChar('s', 6);
    }
```

Media: