ECE 218
EXERCISE 7. DC GEAR MOTOR INTERFACING

This exercise will guide you through the process of interfacing a DC gear motor to the PIC24. You will use an H-bridge driver circuit, powered by a regulated battery pack voltage, to convert the PIC24 output signals to motor control signals. Part A of the exercise will use simple on/off control of the motor, and part B will use PWM to control the speed.

Figure 1 shows the major components of the hardware: the PIC24, H-bridge driver chip, the DC motor, 6V battery pack, and the 5V regulator. Their connections to the PIC24 will be the same in Parts A and B of the exercise, and we will add encoder connections in Part C.

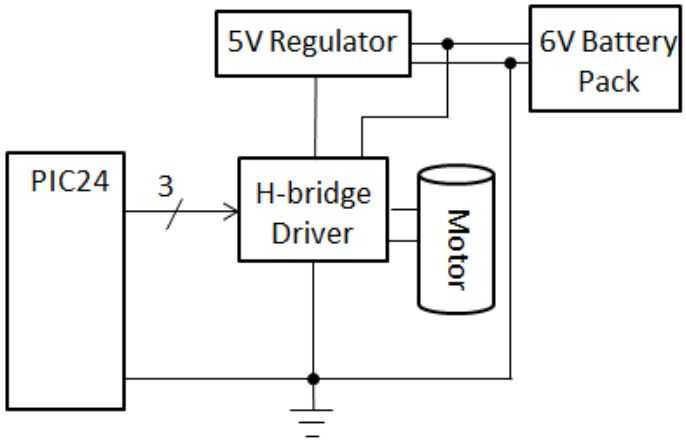

Figure 1. Block Diagram of Part A and B Hardware

PART A – ON-OFF MOTOR CONTROL

The H-bridge driver allows the 3.3V PIC24 outputs to control the high current signals to the DC motor. This chip has two voltage supply inputs, a 5V supply that powers the transistors in the circuit, and a motor voltage (6V) that provides the current for the DC motor. The two inputs to the DC motor can be reversed in polarity to reverse the direction of the motor.

HARDWARE SETUP

Figure 2 shows the details of the connections required, and the additional two capacitors required for the voltage regulator circuit. In this circuit we will control the motor with the digital RB1, RB2, and RB12 port pins. Use the datasheets for the motor driver and voltage regulator on Nexus to verify specific pin locations and understand the purpose of the signals in the schematic. Note that the ground is common to all components in this circuit.

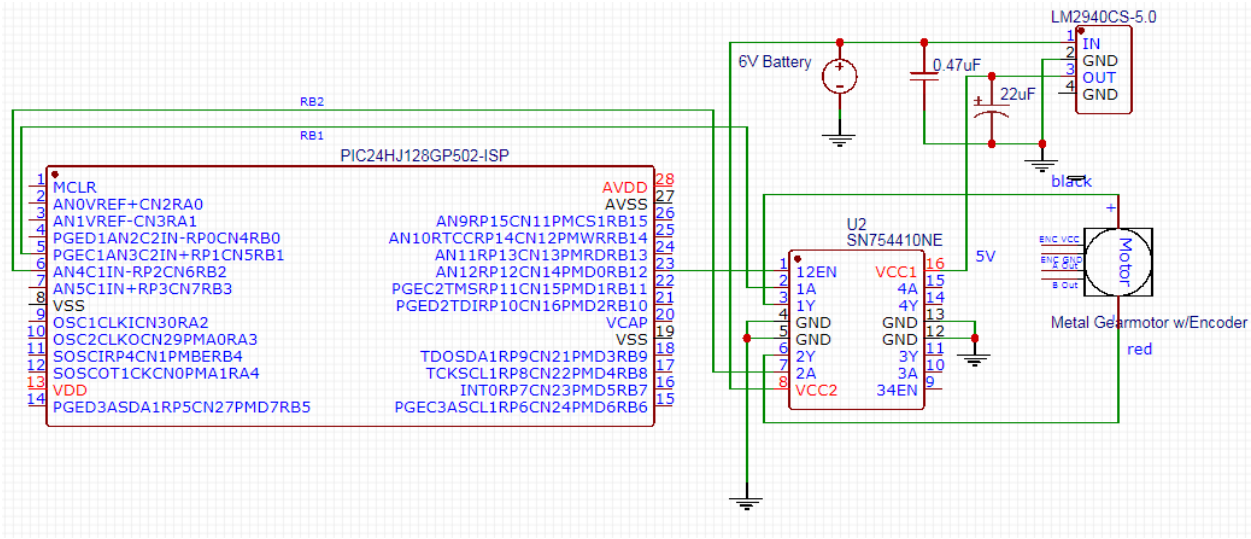

Figure 2. Schematic for Part A and B Hardware

SOFTWARE SETUP

The goal of the first program is to test our hardware connections by repeatedly moving the motor in one direction, full speed, pause, and then run it in the opposite direction.

Definitions and Header Files:

We will start a new project using the textbook library files. Starting with the library template file, define logical names for the RB1, RB2, and RB12 LATB pins that correspond to the 1A, 2A and EN_12 signals of the H-bridge driver chip. Note that #define constants cannot start with a number, so A1 will work, but 1A will not.

Main program initialization:

The only initialization required is to set the direction of the IO pins to output with the TRISB register, or with the library macros such as CONFIG_RB2_AS_DIG_OUTPUT(). Remember that to output a value on the RBx pins, we write to the LATBx bit.

Main program loop:

For the motor to run, the enable signal must be high, and one of the two driver signals, 1A or 2A, must be high, with the other low. Using this information, and delay functions from the library, write a program to repeatedly:

1. run the motor in one direction for 5 seconds
2. turn the motor off for 1 second.
3. run the motor in the opposite direction for 5 seconds
4 turn the motor off for 1 second.

PART B – PWM MOTOR CONTROL

Now we will use PWM to control the speed of the motor. Since two control signals are required to achieve both directions, we will need two output compare modules, OC1 and OC2. For the DC motor we will need a much faster PWM signal (T= 102us) and a much larger duty cycle (about 50% for a slow speed to 100% for full speed). To accomplish this we will set the prescale for Timer 2 to be 1:8 so that Tck = 0.2us. Calculate the number of Tcks needed for the period and minimum and maximum speeds below.

PWM_PERIOD = 102us = 20.4 Tck.

pmin = 51us = 10.2 Tck

pmax = 102us = 20.4 Tck

HARDWARE SETUP

The circuit for this part is the same as Figure 2, but the software will configure RB1 and RB2 to be connected to OC1 and OC2 to produce the PWM outputs needed to vary the speed. As in Project 2, this is done in the OC1 and OC2 configurations. The EN1_2 signal will remain connected to RB12.

SOFTWARE SETUP

The goal of the second program is to slowly increase the speed of the motor to its maximum speed in one direction, then reduce the speed to a stop, and then repeat in the opposite direction.

Definitions and Header Files:

Define constants *p_min*, *p_max*, and *PWM_PERIOD* to be the values you calculated using *#define*. Keep the definition of the EN1_2 name. A1 and A2 are no longer constants, but can be defined as global variables that will hold the pulse widths.

Functions:

Add the *configOC1(), configOC2()*, *configTimer2()* and Timer 2 interrupt service routine that you used in Project 2. Change the prescale settings of the Timer to have a 1:8 prescale for the clock.

Main program initialization:

You will need to configure the OC1 and OC2 modules and Timer 2, turn Timer2 on, and enable the Timer 2 interrupts.

Main program loop:

For the motor to run, the enable signal must be high, and one of the two driver signals, A1 or A2, must be sending a PWM signal with a non-zero pulse width, with the other sending a pulse width of 0. Using this information, write a program that repeatedly:

1. increases the speed of the motor from minimum to full speed in one direction (try an increment of 10 and a delay of 60ms).
2. runs at full speed for 1 second
3. decreases the motor speed back to the minimum
4. runs at minimum speed for 1 second
5. turns the motor off for 1 second.
6. repeats steps 1-5 in the opposite direction.

PART C – USING ENCODER INPUTS

This section will guide you through the process of using the Input Capture peripheral to monitor the encoder signal from the DC gearmotor. The logic analyzer will be used to measure the encoder pulse period for different speeds, and then the Input Capture module will be used to measure the period in software and display it on a monitor through the serial link. We can use the logic analyzer results to check if the speed is correct.

HARDWARE ADDITIONS

Figure 3 shows the new connections to the motor required, powering the Hall sensor circuit with the 5V regulated source and ground, and the connection of the A (yellow) and B (white) encoder outputs to the Logic Analyzer. (don't connect the encoder to RB9 yet.)

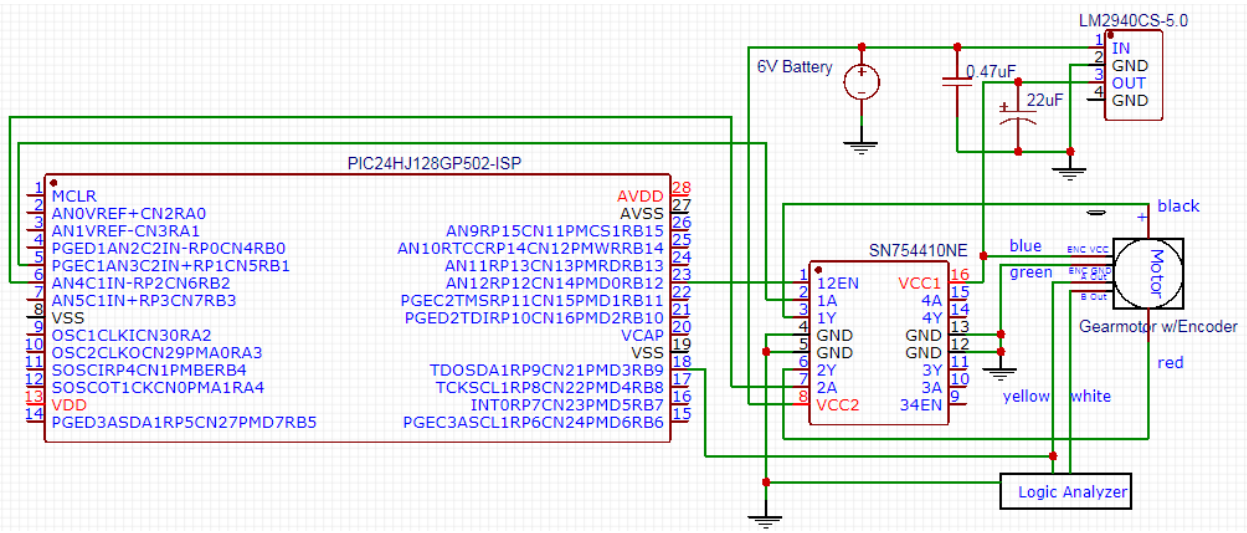

Figure 3. Schematic for Part C

SOFTWARE FOR SPEED MEASUREMENT WITH LOGIC ANALYZER

The next program is a modification of the software from Part B, so save your part B program with a new name, and replace the current main program with this one in the project. Modify the program to set the speed of the motor to its minimum speed (50% duty cycle) and observe the encoder pulse signals using the logic analyzer.

Measure the pulse signal period, T_e, at this minimum speed and record it in Table 1. You can adjust the sample rate or the zoom level in LogicPort to get an accurate measurement. Repeat for the midrange speed (75% duty cycle) and the max speed. The motor encoder produces 48 encoder counts or 12 encoder periods per motor revolution, and since the gearbox reduces the gear shaft by the gear ratio, we must multiply the gear ratio by 12 periods/motor rotation to find the number of periods per revolution, PPR. The angular speed of the output shaft in revolutions per second can then be calculated using the equation below and the PPR in Table 1 for your gearmotor.

$$\omega = \frac{1}{PPR\ \frac{period}{revolution} \times T_e \frac{seconds}{period}}$$

Gear Ratio	PPR	Gear Ratio	PPR
4.4:1	52.8	20.4:1	244.8
9.68:1	116.16	34:1	408.17

Table 1. Periods per Revolution for 4 Gearmotors

Fill in the timer counts used for the three motor speeds, the encoder period, T_e, you measure on the logic analyzer, and then the calculated angular shaft speed, ω, in Table 2.

PWM pulse width(timer counts)	T_e (s)- Encoder Period	ω(rev/s)
(min 50%)	4.33 ms	4.37
(mid 75%)	2 ms	9.47
(max 100%)	1.32 ms	14.34

Table 2. Shaft Speeds vs PWM Duty Cycles

SPEED MEASUREMENT WITH INPUT CAPTURE

Now we will use the PIC24 for both controlling the motor speed, and measuring and displaying it. This will require the serial cable connection, one more hardware modification, and more software changes.

HARDWARE CONNECTION TO IC1

We will use the Input Capture module, IC1, to measure the encoder pulse period. Note that because we are powering the encoder with 5V, the pulses will be 5V in magnitude. It will be important to use one of the 5V tolerant pins of the PIC24 for the pulse input. We will configure the IC1 pin to be RP9 (pin 18). Make the connection from one of the motor encoder signals, A or B, to pin 18. In Figure 3 the A signal is connected.

SOFTWARE FOR SPEED MEASUREMENT WITH IC1

The goal of this last program is to use the Input Capture peripheral to calculate the period of the encoder signal. We will be using the serial interface from Exercise 2 to display the result. We will compare this to what we measured with the logic analyzer.

Global Functions and Variables:

We will need to initialize the Input Capture peripheral. The initialization is similar to Output Compare, and is provided here, with comments explaining the settings. See the datasheet for more details.

```
void configIC1(void) {
  CONFIG_IC1_TO_RP(RB9_RP);  //Map IC1 to RB9 (pin 18)
  T3CONbits.TON = 0;  //Disable Timer3 when writing to IC control registers
  IC1CONbits.ICTMR = 0; //use timer 3
  IC1CONbits.ICM = 0b011; //capture every rising edge.
  IC1CONbits.ICI = 0b00; //interrupt every capture
}
```

As you can see from the ICTMR setting, we will be using Timer 3 with the IC1 module. Therefore Timer 3 must be configured below. We will use a 1:8 prescale, and so the configuration is similar to Timer 2. And we will set the PR3 period register to its maximum value.

```
void configTimer3(void){
   T3CONbits.TON = 0;  //Disable Timer3 when writing to IC control registers
   T3CONbits.TCKPS=0b01;  //prescale 1:8 - Timer 3 clock period = 0.2us
   PR3 = 0xFFFF;          //Maximum Timer3 interval
   T3CONbits.TCS = 0;     //(instruction clock = Tcy)
   TMR3 = 0;
   _T3IF = 0;
}
```

The input capture interrupt service routine has just 4 jobs in this exercise and the code is provided below.

```
void _ISR _IC1Interrupt(void)
{
  EdgeD = IC1BUF;  //Read the IC1 buffer to capture the time of the most recent edge
  IC1_period = EdgeD - EdgeA; //Calculate the counts in one period by subtracting
                                    //most recent edge from the previous edge.
  EdgeA = EdgeD;          //Store the most recent edge as the previous edge
  _IC1IF = 0;  //Clear the IC1 interrupt flag so that interrupts can happen again
}
```

The global variables, EdgeD, EdgeA, and IC1_period are unsigned integers, and must be declared for communicating between the interrupt service routine and the main program. These should be declared before the IC1 interrupt service routine is defined.

Main program configuration:

In addition to the Clock, OC1, OC2, and Timer 2 configurations that you called in the previous program, for this section we must also configure the Input Capture module, Timer 3, and the UART.

```
configClock();  //Sets the clock to 40MHz using FRC and PLL
configOC1();
configOC2();
configIC1();
configTimer2();
configTimer3();
configUART1(230400); //Configure UART1 for 230400 baud
```

Initializations:

In addition to configuring RB12 as an output, you will need statements to turn on Timers 2 and 3, and enable the Timer 2 (_T2IE) and IC1 (_IC1IE) interrupts. Keep in mind that Timer 2 is updating

the PWM signal when it interrupts, and IC1 is capturing the encoder signal edges. Note that we do not use Timer 3 in interrupt mode, it is only used by the Input Capture peripheral.

Main program loop:

In the main loop you will use the IC1_period (in timer counts) that is calculated in the IC1 interrupt service routine and the period of Timer 3 to calculate the encoder pulse period in seconds and print it to the monitor. You should then use the encoder period and PPR of your motor to calculate the angular speed and print it so you can compare it to the speeds predicted in Table 2. You will want to put at least a 2 second delay in your main loop so that you can read the speed readings as they are updated.

Printing hints:

As in Project 1, you will have the problem of converting numbers (an integer (encoder period) and a floating point value (angular speed)) to a string for sending to the terminal. You can use your function from Project 1 and the outString() function, or the *printf()* function that is found in the *stdio.h* library. Documentation for this function can be found online.

Remember that the \n \r characters together move the cursor to the beginning of a new line, and can be used to make the display easier to read.

CHECK FOR UNDERSTANDING

Given the period of Timer 3, what is the precision of the encoder pulse reading? That is, what is the difference, in time, between IC1BUF readings of 0x1113 and 0x1114?

How much does the speed vary from reading to reading? Do you think this difference, if any, is a function of the measurement process or
How much does the average speed reported differ from what you calculated?