Lab #4 Parallel Output/Input Control

1. Description

In this laboratory, we implemented a finite state machine algorithm in the main loop and the routines: low(), high(), speed(), and stop for the states, initializeSM(), and vel(). The purpose of this lab is to understand using PWM (pulse width modulation) signal to control the velocity of a DC motor and to learn about finite state machines. The main loop contains an algorithm for a finite state machine that has 4 states: low, high, speed, and stop. The low state refers to setting the output of the PWM signal to be at 0A ($\overline{run} = 1$), while the high state refers the PWM signal to 0.25A ($\overline{run} = 0$). The PWM signal output switches between low and high in one Basic Time Interval (BTI). At the start of each BTI (or when the Clock = N), the clock is set at 0, and $\overline{run}$ is set at 0 (current flowing). Once the Clock reaches M, $\overline{run}$ is set at 1 (no current flowing). This sequence is repeated to achieve a sequence of PWM whose duty cycle we can input. The duty cycle is inputted by the user by prompting the value of N and M before running the finite state machine, where double_in() is used to prompt for user input. The state machine has two other states, speed and low. The speed state has a purpose of printing the current motor RPM by using the encoder counter value. In short, we can calculate the position difference between one BTI by subtracting the counter from the current and previous BTI and multiplying it with the Basic Displacement Increment (BDI); this process is done by the vel() routine. Once the positional difference is found, we can use the equation below to obtain the motor RPM.

$$RPM = vel(\tfrac{BTI}{BDI}) * 1000(\tfrac{s}{ms}) * 60(\tfrac{s}{min}) / (5.002(\tfrac{ms}{clock}) * N(\tfrac{clock}{BTI}) * 2000(\tfrac{rev}{BTI}))$$

The speed state is achieved by transitioning from the high state when the button press in Channel 7 is detected. The stop state is entered when a button press at Channel 6 is detected. The stop states purpose is to exit the state machine and to turn off the motor. An additional feature to the stop state is that it exports the motor RPM measurement values to a MATLAB file, which is used to analyze the step response of the motor RPM. Before running the state machine, the routine initializeSM() is implemented to initialize the channels and to setup the preconditions. This routine initializes Channel 0, 6, and 7, and the encoder counter. It sets $\overline{run}$ to 1 (no current flow), and it sets the current state to LOW and the clock to 0.

Inside the while loop for the state machine, a wait() routine is implemented to set the BTI to 5.002s calculated from the clock cycles of each commands of its assembly code and from the microprocessors clock frequency of 667 MHz. One limitation that also serves as a safety feature of this program is the inability to set the current output of the PWM signal. The current output is limited to 0.25 A, while if we would like to spin the motor faster, we would need higher current value, which might cause parts to be overloaded. Once the motor speed is fast

enough, it is possible that the RPM value cannot be calculated by the encoder if the counter exceeds the maximum bit value (0xFFFFFFFF) at a BTI interval.

The functional hierarchy of this lab can be found in Figure 1 below.

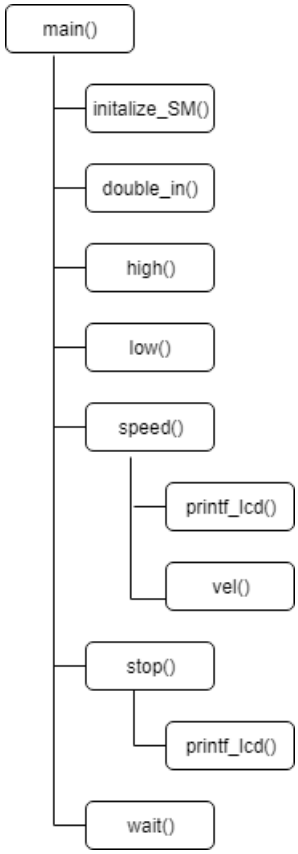

Figure 1: Function Hierarchy

2. Testing

Running the program will invoke the sequences described below:

1. The LCD screen prompts for a value of N, use the keypad to input the N value desired. This value of N is stored in the program to be used in the finite state machine. N refers to the number of counts the clock has to be incremented to achieve 1 BTI. For testing, I inputted N = 5.

2. The LCD screen then prompts for a value of M, use the keypad to input the M value desired. The duty cycle of the PWM signal is N/M. Therefore, the higher the value of M, the higher the velocity output is. M refers to the number of clock counts in 1 BTI for the PWM signal to be in high state. M is stored in the program as well. For testing, I inputted M = 3.

3. Once both N and M are inputted, the motor will run. The function generator will show switching values from Channel 0. In order for the motor to run, we need to implement the routine Dio_WriteBit(&Ch0, run) which sets channel 0 to $\overline{run}$. In addition, we also need to ensure that initialize_SM() is properly implemented.

4. Once the motor is running, validate the speed state by pressing the button at Channel 7. Once the button is pressed, the LCD screen should print the current motor RPM value. Check if the speed is properly printed by holding the button and observing the trend of the value. To validate the RPM value, use the optical tachometer and compare the values. A delay is expected between the tachometer reading and the value printed in the LCD. One way to calibrate the accuracy of the motor RPM printed is by waiting until the motor reaches steady state and comparing the value with that of the optical tachometer (this reading will actually still be inaccurate, and the reason is discussed in the Results section).

   In the script submitted, the print capabilty has been replaced with the export to MATLAB capability in which when the button is pressed, myRIO generates Lab4.m file that contains the motor RPM value across time, with the corresponding N and M values.

5. To stop the motor, press the button at Channel The motor should decelerate until it stops and the LCD screen should display Stopping. Upon stopping, myRIO exports the motor velocity profile and exports it to Lab4.m.

3. Results

The testing procedure runs as expected. The actual length of a BTI is approximately 25 ms (accurate within +-5ms), which is as expected from the program, since N is set to 5, and each state takes 5 ms to run. When Channel 7 is pressed, the length of one BTI increased to 40 ms. The reason for this increase in BTI is because printf_lcd() routine takes more clock cycles to run, causing more tasks to be operated by the microcomputer. Hence, when the motor RPM is low,

the program will print relatively accurate values while at higher motor RPM, the value printed will be around 1000 RPM greater than that of the optical tachometer since the BTI has increased.

Fixing the M = 1 Case

When M = 1, upon pressing the button at Channel 6, the motor will run to its maximum speed and the buttons will not cause state transitions. The reason for is because the state machine is stuck at the low state where $\overline{run}$ is set to 0 (allowing current flow). The state machine is stuck because the only transition condition is if Clock == M. In this case, M = 1, and since the speed state is only achievable by transitioning from the high state (where the Clock is reset to 0), since at each while loop iteration, the clock is incremented such that Clock = 1 is skipped. Therefore, the state of the system will be stuck at the low state while $\overline{run}$ was previously set to 0. This case can be solved by ensuring that when M = 1, the state can transition from the speed state directly to high state so that the system can transition back to the low state. An if statement to catch such condition is implemented in the speed state.

Step Response Analysis

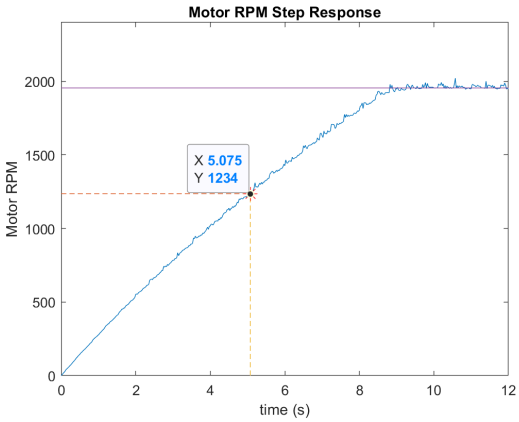

Figure 2: Motor RPM Step Response

Figure 2 represents the step response of the motor RPM with the PWM signal of N = 5 and M = 3. The time constant of this response is around 5s ($\tau = 5.075s$). The steady state motor RPM is $1955rpm$ according to the vel() routine. The steady state RPM is calculated by picking one of the point that corresponds to the flat line on the plot. The time constant is calculated by drawing a horizontal line corresponding to 63% of the steady state value.