Seth Cram

ECE341

Lab10 Prelab

11/30/2021

**Prelab 10: Input Capture to Measure Motor Speed**

Goal:

Use the input capture module to measure frequency, which will allow us to find the speed of the DC motor.

Background Information:

The DC motor has hall effect sensors that generate a signal in response to a magnetic field. This magnetic field is created through the rotation of two magnets in and out of the range of hall effect sensors SA and SB. Because of the sensor's positioning, SB lags behind SA by about 1 digital output. The result is called Quadrature Encoding, and it's useful to determine the shaft speed and determine what direction the DC motor is turning in. For this lab, we’ll only use SB since we’ll only be determining the motor’s speed and SA is not a designated input capture pin.

To determine the speed of the DC motor, we’ll have to capture a timer’s value when an edge is generated by SB. Since the duty cycle generated by SB isn’t perfect due to hall effect tolerance, we should measure the rising edge or the falling edge, not both. The output capture module will capture a specified edge for us and store a timer’s value at the time of the event. We can just read this value out of the FIFO memory that the output compare module stores the timer’s value in.

We could poll until something is stored in the FIFO memory area, or we could configure the input capture module to generate an interrupt every edge we told it to capture. For this lab, we’ll use the falling edge and configure the input capture module ISR every interrupt it generates. Now that we have the time between consecutive falling edges generated by SB, we can invert this to get a frequency. This frequency is the motor’s rotations per second, its speed, since a sensor only pulses once every revolution.

If desired, we could configure the input capture module to only capture every 4th or 16th edge using a prescaler. We'll only use a 16-bit timer, timer 3, since combining two 16-bit timers to get a 32-bit timer would only be appropriate for incredibly widely spaced out events. In order to keep track of higher counts, we could also use a timer’s prescaler, but doing this decreases our resolution. As always, when dealing with timers we want to clock them as fast as possible without overflow.

Similar to lab 9, we will once again be using the H-bridge to communicate between the PIC32 and DC Motor/tachometer. Sensor A and sensor B are outputs from the H-bridge to the PIC32. The DC motor tachometer is also used in this lab to measure the analog speed of the motor. It acts as a voltage to frequency converter for the DC motor.

Some important parameters for verification in the lab is that using a 10V supply, the max motor speed is 525 RPS, and due to friction the min motor speed is around 79.5 RPS. We’ll be using input capture interrupt 5, since SB is connected to input capture pin 5.

Plan:

First, I plan on building lab 10 off of lab 9. So, all function code for lab 10 will be included in a new file that’ll be included by the main() file used in lab 9. In this new file I’ll create an input capture initialization function that’ll set sensors A and B as inputs as well as clear, configure, and open the input capture 5 module. Most of this will be done using macros provided by the lab handout. We’ll use two timers during this lab, since the period registers of the timer’s are set differently to fulfill different design goals.

Then, I’ll create the timer 3 initialization function based off of the timer 2 one present in lab 9. Its priority and sub-group priority is specified in lab, and it’ll be opened with a prescaler of 256 as well as a period register of the maximum value possible. We’ll be setting the period register for timer 3 to its maximum value of 65,536 to avoid incorrect period measurement when the timer rolls-over multiple times and to achieve a high level of resolution. This interrupts corresponding ISR will merely toggle LEDC and clear the timer 3 flag.

Finally, the Input Capture ISR will be structure based off-of Dr. J’s pseudocode provided in-class and the code given in the lab handout. Also, truncating our unsigned integers to unsigned shorts allows for period calculation to be correct when the timer rolls over a single time. I’ll store each calculated t\_diff as an element in an array, and the counting variable will reset to zero every time it reaches 16, which will make the array act like a circular buffer. We will average multiple period measurements to reduce measurement noise. Once 16 or more t\_diffs are recorded, I’ll calculate the speed through averaging these and inverting the period to get the frequency. I’ll need to use floats for this to avoid large rounding errors. The speed will be stored in a global variable. Once rps is calculated, I’ll output the appropriate message to the LCD

To only write the second line of the LCD in the input capture ISR, and only the first line of the LCD in the Change Notice ISR, I’ll need to create a new LCD function to clear any desired line of the LCD and put the cursor at the start of that line when done. Delving into my LCDlib file, I’ll create this by accepting what line to clear as an argument to a function and then writing 16 blank spaces to the specified line, then changing the cursor’s location back to the start of that line. I’ll have to take into account using the busy flag to see if the LCD Controller is ready yet when changing the cursor location. Like in previous labs, everytime I write to the LCD, I’ll also have to disable, and after writing, re-enable the change notice interrupt.

Back to the main() file, I’ll do everything in lab9, then also initialize the input compare module, and Timer3 as an interrupt. Within the while(1) loop, I’ll disable the Change Notice interrupt before clearing LCD line 2, formatting my desired RPS string using sprintf(), and output that formatted string onto the LCD. Finally, I’ll re-enable the Change Notice interrupt and delay for 100 ms before repeating. I’ll be sure to include the global variable in my main() file using extern.



