

ECE 414L Microprocessor Applications Laboratory

Xilinx Motor Controller

John Risch
jdrisch@hotmail.com

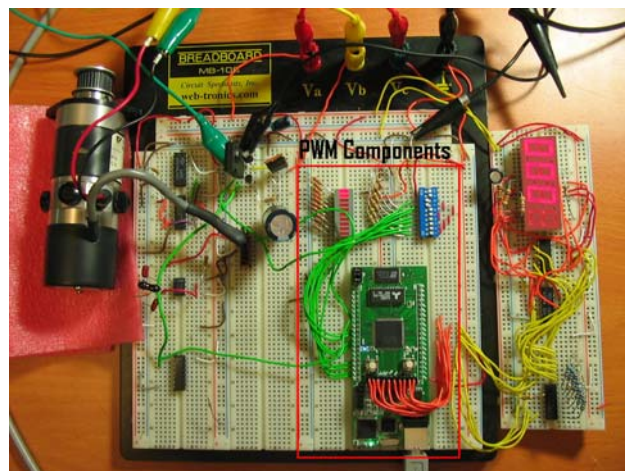
Abstract

The objective of this laboratory is to develop and implement a PWM (Pulse Width Modulated) motor controller with a proportional closed feedback loop using a Xilinx FPGA microprocessor.

In this portion of the lab we implemented a PWM output to control our motor speed. We set the period τ using the mentioned equation given the T was 40 ns for my 25 Mhz Xilinx FPGA board.

Introduction

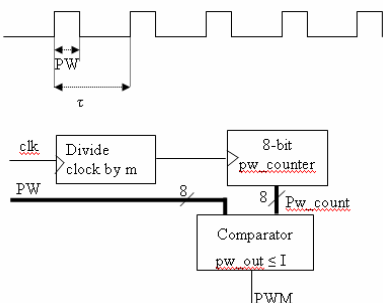
In this laboratory we implemented a PWM motor controller component by component. First we programmed the FPGA to output a PWM with an 8-bit # input from hard switches. Then implemented a speed monitor within the FPGA by counting the encoder speed and outputting an 8-bit # onto a 3 digit 7-segment display. Then built a driver circuit to power the motor to view the open-loop step response. Next we interrupted the switch input with a proportional controller that inputs the current speed and switch input and attempts to alter the PWM to match the current speed measurement with the current switch input. Last we tuned our controller by installing a frequency to voltage IC to output a voltage given the motor encoder speed. This enabled us to observe the step response of the motor speed to a given input value, which we used to tune K_p the proportional constant of the controller.



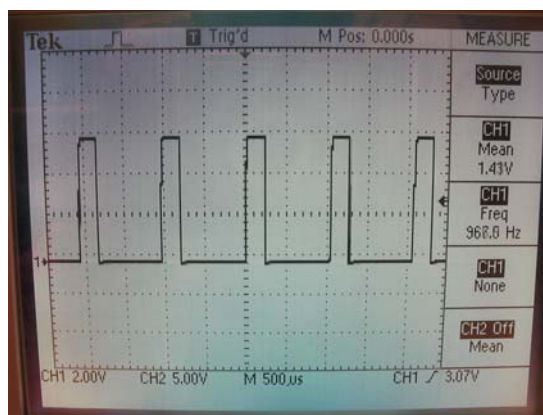
This image highlights the components of the physical hardware that implement the PWM. The PWM was created entirely in verilog with only the 8-bit switches as a control input, the LEDs I simply added to indicate the presence of 5V, 3V, and each bit value MSB from top. The resistors were needed to bias the LEDs and maintain a 3V input to the FPGA.

Laboratory

PWM

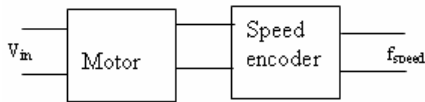


$$m = \frac{\tau}{255 \times T} \quad \text{Where T is the clock period}$$

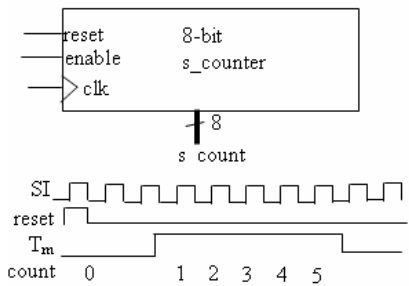
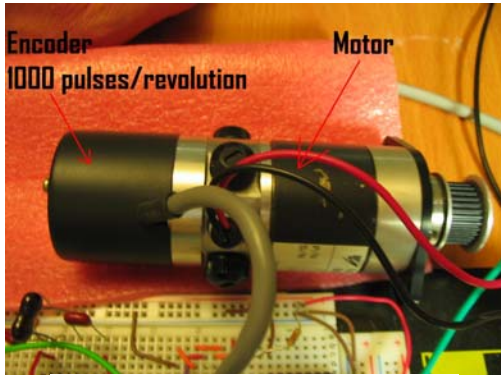


This image shows the Time domain appearance of the PWM, the duty cycle shown is about 25%.

SPEED COUNTER/DISPLAY

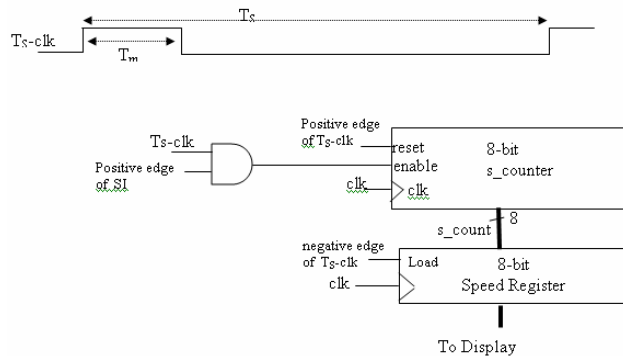


The motor required for the laboratory needed to have an encoder module attached so that the RPMs could be measured. In my case the motor had a 1000 pulse/rotation encoder.

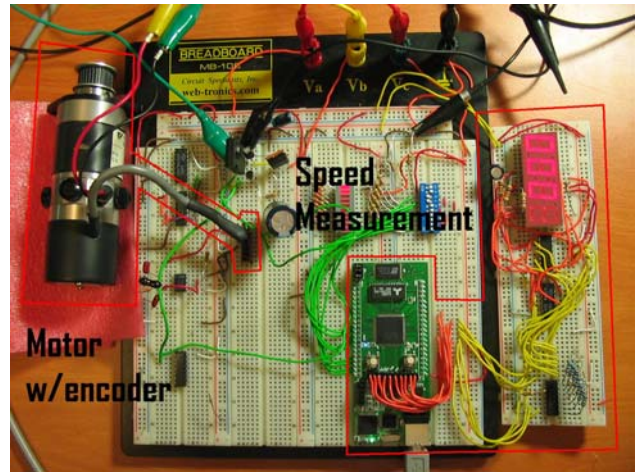


To measure the frequency a counter can be used. After resetting the counter it is enabled for a period T_m . The counter is then incremented at the positive edge of each input signal's pulse (SI). The count at the end of the period T_m is proportional to the speed

$T_m = 255 \times$ shortest period of the speed signal.
Thus the measured speed will range from 0 to 255. The speed should be measured once every sampling period.



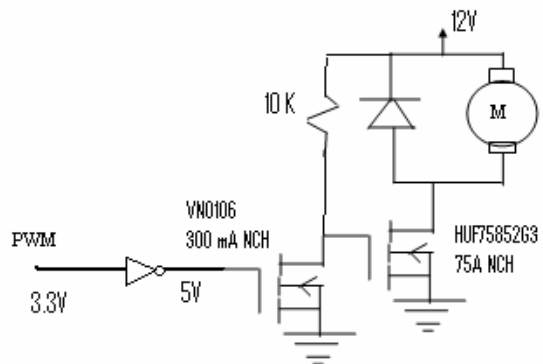
The above image describes the implementation of the verilog code using discrete logic components.



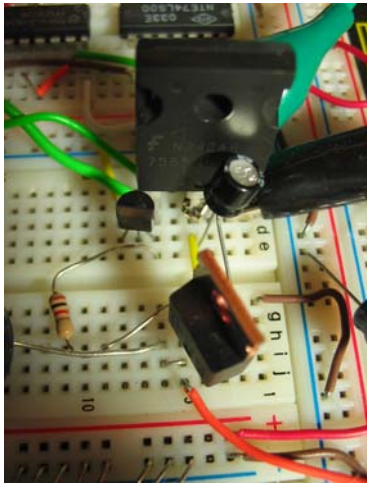
This image highlights the physical components used to implement the Speed measurement and Display. Again most of the implementation is contained within the Xilinx FPGA in verilog code, the Display is driven with an 74LS138 BCD to 7-Segment IC and a 74LS04 inverter IC. Using the Xilinx FPGA to time the output, only one BCD signal exists and a wire to enable each digit, in total only 7 bits was needed to run the display. The encoder connector simply outputs a 5V square wave that pulses 1000 times per revolution. The Speed measurement is accomplished by counting a given period that will reach a maximum of 1024 at maximum speed, then the MSB 8-bits are passed, this allows a higher resolution, and smoothes the display appearance.

Open-loop step response

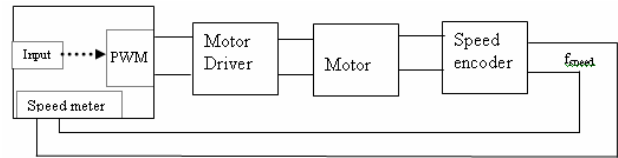
In this step we implemented the motor driver.



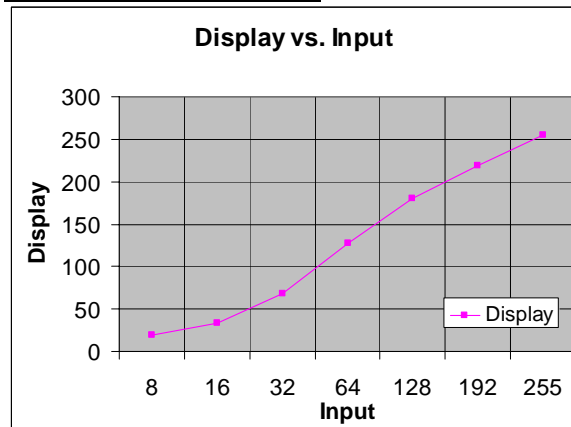
This schematic shows the components used to implement my motor driver. The inverters and common source MOSFET helped isolate the FPGA from the Drive system while lifting the control voltage to a 12V signal, this provided cleaner turn-on for the large Power MOSFET. The Large power MOSFET was what I had on-hand and is far beyond necessary for this no-load application.



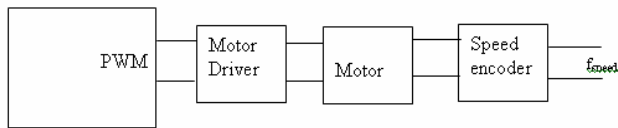
This image shows the physical hardware used to implement the motor driver. The largest device is the 75A POWER MOSFET, the small TO-92 package is the VN0106 MOSFET, the TO-220 package in the bottom is a 20L15T 20A Schottky Diode. The Electrolytic capacitor was just used in parallel with the motor to smooth the motor response, and quite the motor.



| Input | Display |
|-------|---------|
| 8 | 19 |
| 16 | 34 |
| 32 | 68 |
| 64 | 128 |
| 128 | 180 |
| 192 | 219 |
| 255 | 255 |

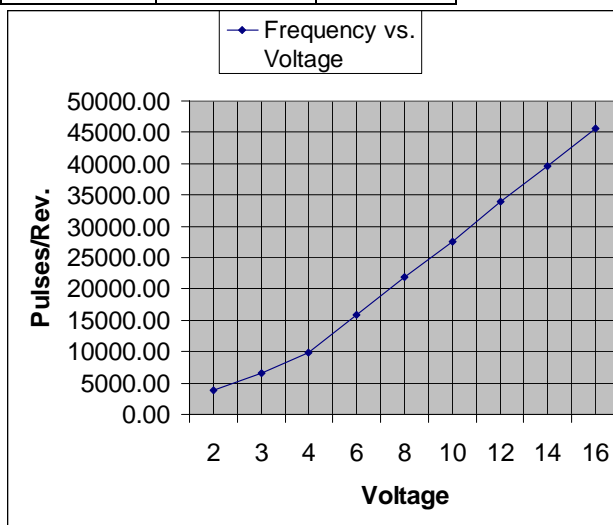


The Input verse Display relationship is near linear but the Display shows that the unadjusted PWM output causes the motor speed to go twice as fast as expected.



The next step was to adjust the Period τ to better suit my motor.

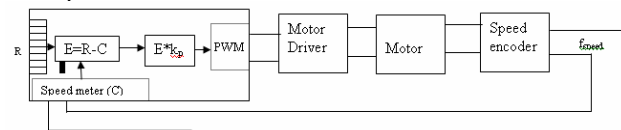
| Voltage (V) | Period (us) | Freq. (Hz) |
|-------------|-------------|------------|
| 2 | 263.2 | 3799.39 |
| 3 | 151.5 | 6600.66 |
| 4 | 101.5 | 9852.22 |
| 6 | 63.2 | 15822.78 |
| 8 | 45.7 | 21881.84 |
| 10 | 36.2 | 27624.31 |
| 12 | 29.5 | 33898.31 |
| 14 | 25.2 | 39682.54 |
| 16 | 21.9 | 45662.10 |



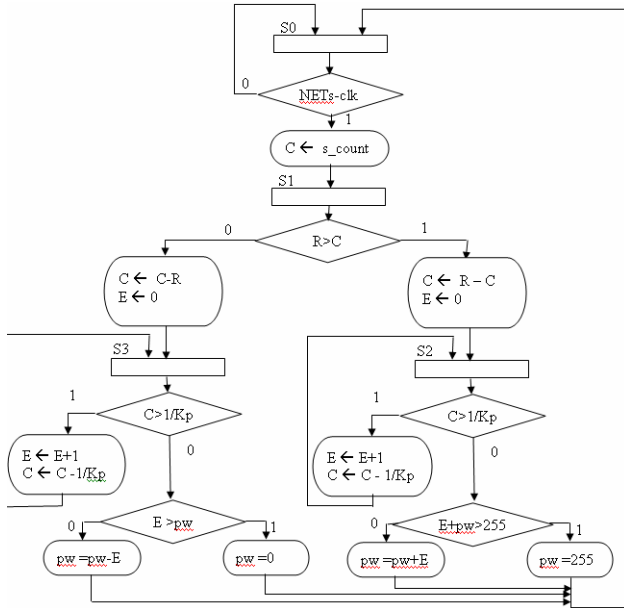
The Frequency to Voltage curve appears very linear, this means the driver setup is sufficient to continue.

Proportional controller

Now that we see how we need to adjust the PWN to match the input and the output speed of the motor, we will implement a Proportional controller to adjust the PWM output so that the output speed that is displayed matches the input.



This block diagram shows the system implementation.

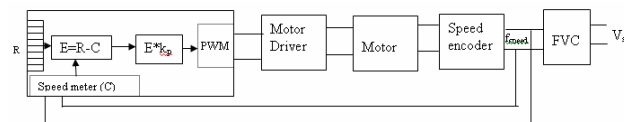


This State diagram shows the implementation of the proportional controller using states. This translates to case statements in verilog, which will be used to program the FPGA.

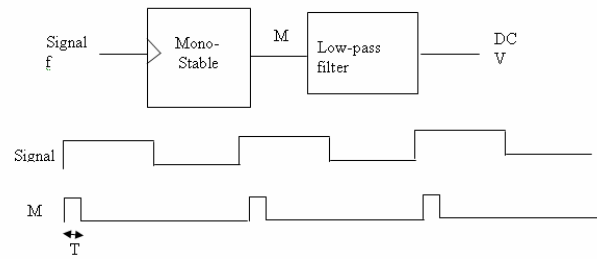
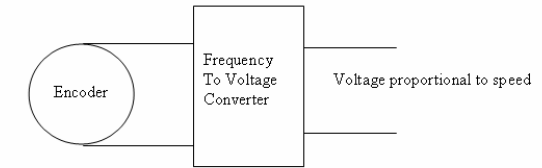
| SW input | Display | T (us) | Encoder Frequency |
|----------|---------|--------|-------------------|
| 23 | | 256 | 3906.25 |
| 31 | 57 | 125.2 | 7987.22 |
| 47 | 95 | 70.1 | 14265.34 |
| 63 | 138 | 49.5 | 20202.02 |
| 80 | 169 | 41.17 | 24289.53 |
| 112 | 198 | 34.5 | 28985.51 |
| 144 | 216 | 31.47 | 31776.29 |
| 176 | 225 | 29.63 | 33749.58 |
| 192 | 238 | 28.96 | 34530.39 |
| 223 | 248 | 27.83 | 35932.45 |
| 255 | 251 | 27.38 | 36523.01 |

This data shows the implementation of the proportional controller has greatly improved the Steady state error and linearity of the system.

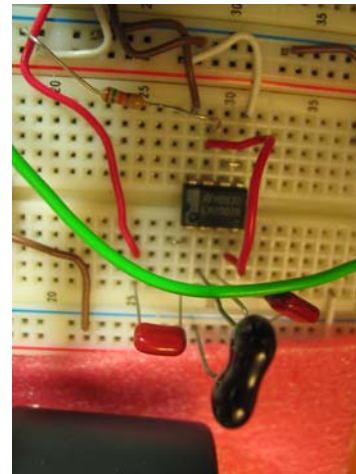
Closed loop step response



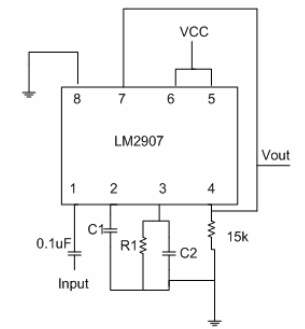
We now will add a Frequency to Voltage IC to output a DC voltage proportional to the motor speed.



This image shows how the Frequency to Voltage IC works, translating a Square wave input into a DV voltage.



This image shows the LM2907 IC, a Frequency to Voltage converter chip.

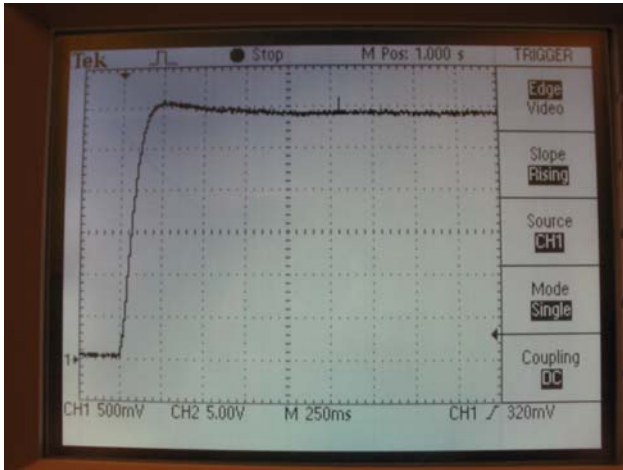


$$V_{out} = V_{CC} f_{in} C_1 R_1 k$$

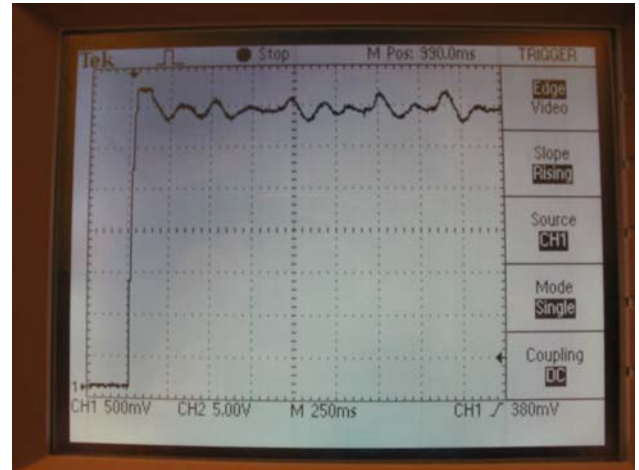
The Circuit diagram to the right shows the pinout and external component configuration needed to use the IC.

| Input | Display | Vout |
|-------|---------|-------|
| 8 | 8 | 0.240 |
| 16 | 18 | 0.320 |
| 32 | 31 | 0.560 |
| 64 | 65 | 1.040 |
| 128 | 128 | 2.000 |
| 162 | 194 | 2.960 |
| 255 | 255 | 3.680 |

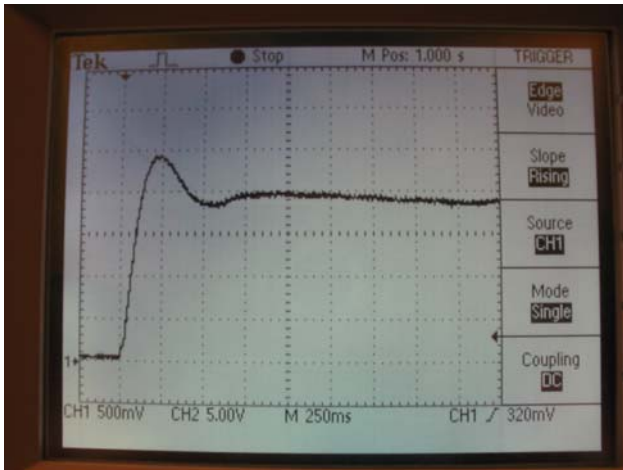
After tuning the Kp value for the proportional controller the new data indicates the proportional controller has significantly improved the system.



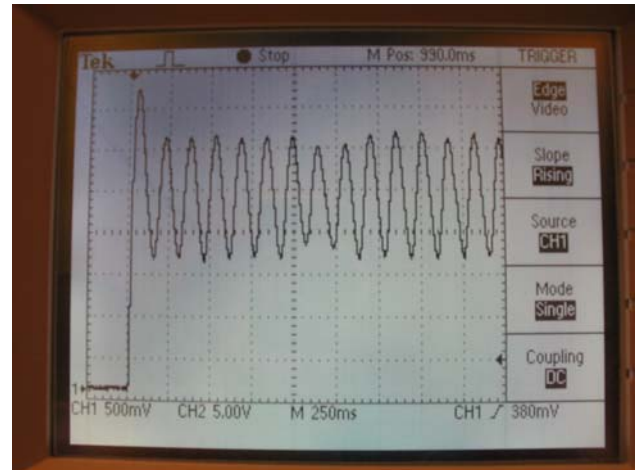
Step Response to Switch input 192 $K_p=1/7$



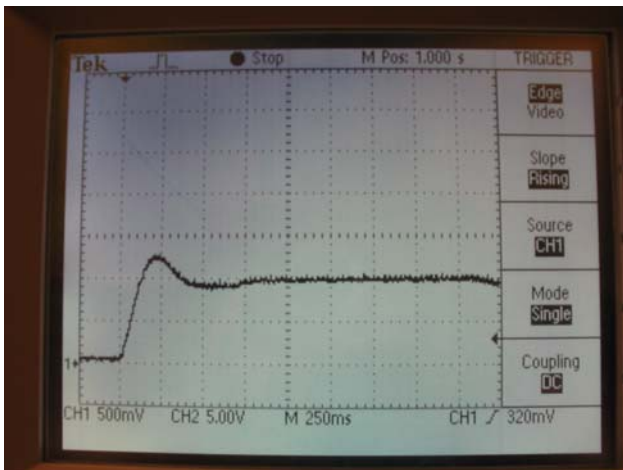
Step Response to Switch input 192 $K_p=1/2$



Step Response to Switch input 128 $K_p=1/7$



Step Response to Switch input 128 $K_p=1/2$



Step Response to Switch input 64 $K_p=1/7$

As can be seen the tuning process while observing the step response allowed us to achieve a desired step response that will meet the design parameters by trial and error.

Conclusions

The PWM controller laboratory was an awesome experience and allowed us to receive exposure to all of design aspects in very little time. The Xilinx FPGA used in this lab will personally be used in my senior project as a result of what I learned from this class. The utility of Microprocessor control in systems is very