MSE 352 - Digital Logic and Microcontrollers

Project Report: Design of a Servo DC motor speed controller using 8051 MCU

Group Number: 12
Report Due Date: December 2, 2019

Member - Student ID

[redacted]

Sepehr Rezvani - 301291960

Table of Contents

Introduction

In this project, the goal was to control a fan using an 8051 MCU and PWM to control the duty cycle. This cycle would vary based on a series of switches which would allow the MCU to determine the desired duty cycle. The RPM of the fan would also be displayed on three separate 7-segments (only showing the 3 most significant figures) and would vary based on the duty cycle. This displayed speed would be gathered by averaging values gathered from the fan and then displayed on the 7-segments. Ideally this project would be showing four 7-segments, but due to the limitations of the 8051 MCU and the lack of 7-segments available, only three 7-segments were used.

Design

The purpose of hardware design was to implement the 8051 Mikroelectronika microcontroller with a brushless DC motor fan and display the tachometer output. The hardware design for the circuit was fairly straight forward, but some challenges did occur when initially prototyping. It is noted that when using the recommended motor driver NMOS circuit, bench testing with a function generator showed an inverted duty cycle was affecting the motor. The wiring accuracy was confirmed and therefore was switched out with a PNP transistor, which showed no inverted effects but the code logic would need to be considered later. The use of a pull-up resistor for the tachometer stabilized the output and reduced the noise. For duty cycle input switches, the default was pull-down resistors but we were unable to use pull-down resistors for the 7-segment default inputs. The 8051 did not have the capability to pull-up default inputs, as per such it was left directly connected so it could be toggled. One of the major concerns for tachometer accuracy for 100% duty cycle (referenced at 0.09A), was poor grounding and fluctuating voltages that influenced the measurements. Due to breadboards having parasitic capacitance, fluctuation of voltages, and other noted issues we could not achieve the exact result. It is noted that when using an independent 3.3V supply for DIP switch input, instead of the 5V rail, there was minimal voltage fluctuations and the result appeared more accurately. Future designs should be completed with a properly grounded PCB design, using separate voltage regulators

and multiple test points for verification.

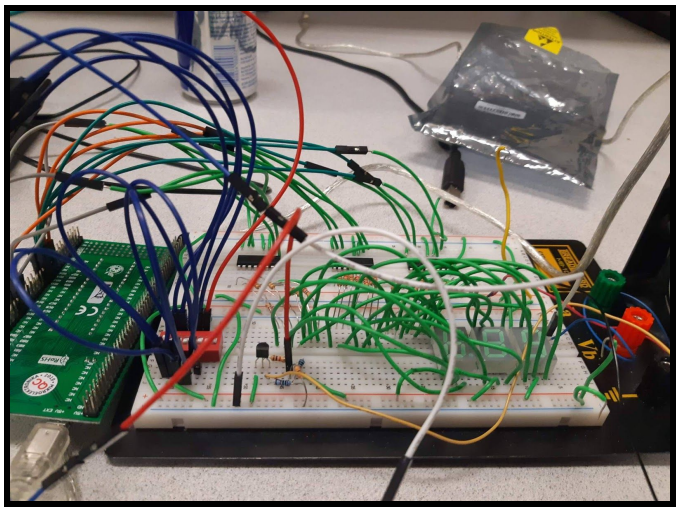

Figure 1

Schematic

The circuit schematic modeled in Proteus can be seen in the following figure.

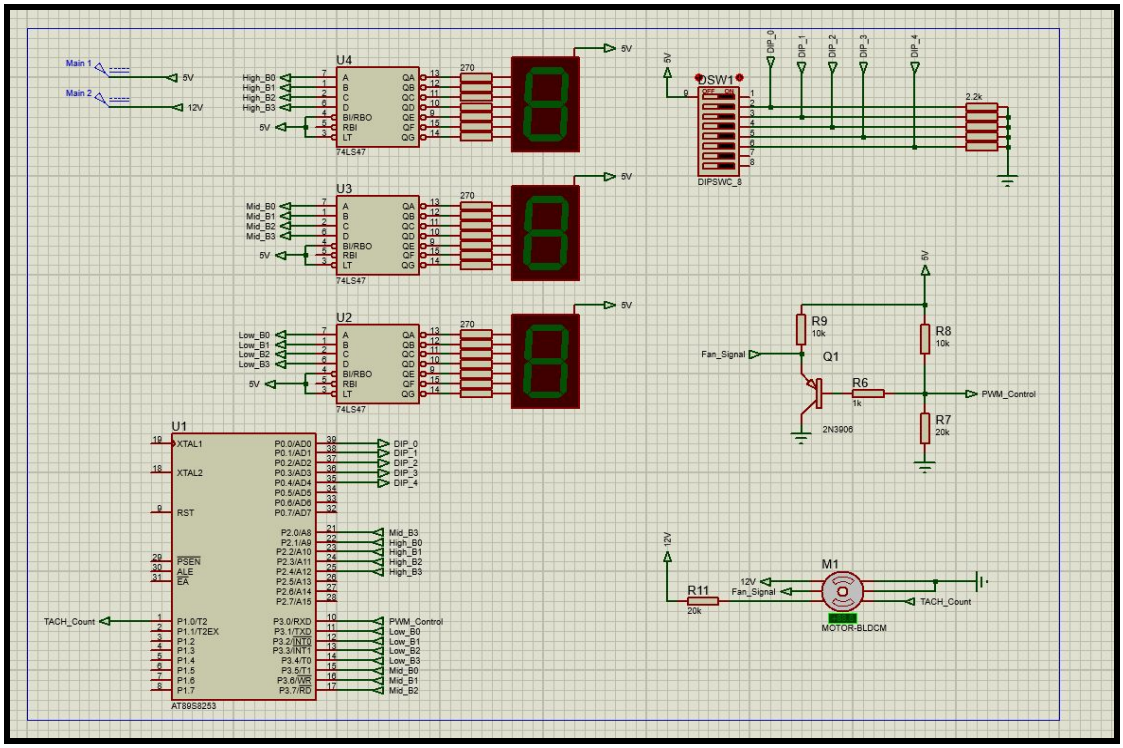

Figure 2

Starting from the top left of the figure, we have three 74LS47 IC's that have active-low outputs designed for driving a common-anode LED display via external current-limiting resistors. The resistors are equal and chosen to be 270 ohms. Seven-Segment-Displays from top to bottom are positioned from left to right on the breadboard and represent rotations per minute of fan connected to 8051 microcontroller. According to Superflow FAN's manual, the maximum speed is 2400 rpm, and the 7-Segment-Displays would show 2, 4 and 0, in order from left to right (in the case where the fan has 100 percent duty cycle).74LS47 IC's are connected to ports 2 and 3 of 8051 microcontroller. DIP switch is connected to 2.2 $k\Omega$ resistors and pins 2-6 of the switch are connected to P0.0 - P0.4 (port 0, pins 0-4) on 8051. Servo motor is connected 20 $k\Omega$ resistor and P1.0. Lastly, 2N3906 PNP transistor was used for switching purposes (explained in Code section) and was connected to RXD or P3.0

Code

The basic tenets of the code are to achieve PWM and provide open loop feedback. This is achieved through configuration code, average motor speed calculations, and using three timer peripherals. Timer0 is used for input control of the brushless dc fan, timer1 is used to determine when one second has occurred, and timer2 is used as an event counter. Each one will be discussed after the general objectives are noted. Objective one, achieve speed control of the DC motor without hardcoding delays that will affect performance of overall code. Objective two, read the output of the tachometer with reasonable accuracy independent of PWM control. General objectives are to maintain efficient configuration code with proper readability. For the second objective, timer1 and timer2 are used in conjunction to count the amount of tachometer pulses per one second.

Timer0 uses the basic concept of pulling the motor driver pin high for a given duty cycle $T_{on} = T_{duty\ cycle}$ and pull low for remaining time $T_{off} = T_{period} - T_{duty\ cycle}$ to achieve a specified average voltage. The speed of a DC motor is linearly related to the applied voltage, so this is how motor speed control was completed. It is noted that because the circuit uses a PNP transistor, the logic is inverted (such that $T_{on} = T_{period} - T_{duty\ cycle}$). The timer is setup to count

until the specified T_{on} or T_{off}, at which point it will overflow and call the declared interrupt service routine for that address.

The simplest method to determine RPM (as noted in motor datasheet) is to determine switching period or pulse frequency $n = 60 / T_s$. This can be achieved knowing when one second has occurred and counting the amount of triggered pulses during that duration. Timer2 was configured as a negative edge event counter
to track the amount of pulses, where we can obtain revolution frequency by dividing this number by two. Timer1 was configured as a 16-bit counter, to determine when one second has occurred so revolution frequency could be passed to a global variable. With an average of three samples, motor rpm is calculated from this. The calculations for number of timer1 ISR cycles is noted below:

$Machine\ Cycle\ -\ 1.046uS,\ 2^{16} = 65536,\ N_{cycles}\ =\ 1\ second / (2^{16} * (1.046 * 10^{-6}))\ =\ approx\ 14$

Some error may occur as a result of this, in which the writer recommends future projects use a timer with lower value to round the number more evenly. Other configuration items can be easily noted in the code comments section.

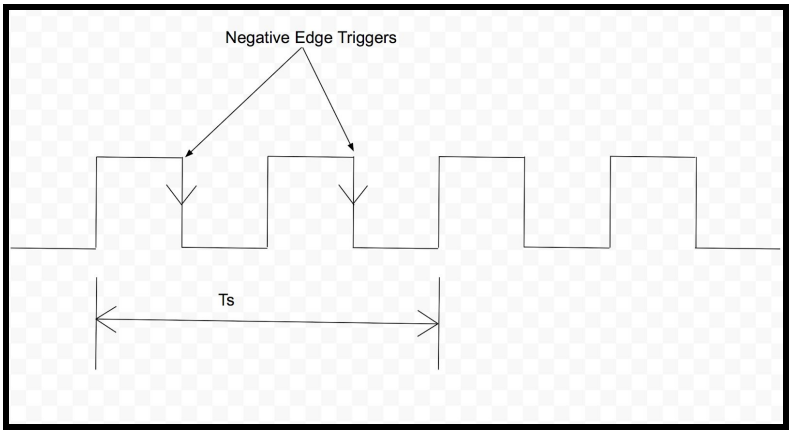

Figure 3

Conclusion

The purpose of this project was to control a brushless DC motor fan with PWM using the 8051 microcontroller and display the output on 7-segment LEDs. This was achieved through wiring a

breadboard with a PNP motor driver circuit, 7-segment LEDs that were connected to BCD-to-7-segment decoders, DIP switch user input, and default pull-up or pull-down resistors where appropriate. There was some hardware issues previously noted when bench testing with the NMOS driver circuit using a function generator and an oscilloscope to track output. The schematic in proteus notes the layout for the reader, as the provided image can be difficult to discern the topology. The main firmware implemented the use of three timer peripherals from the microcontroller. Timer0 and timer1 were configured as interrupt service routine functions, that would step into the function when their respective timer flags overflowed. Timer0 was used with basic PWM code, using inverted logic since we used a PNP transistor. Timer2 was used as a negative edge event counter, to determine the amount of pulses during a given period that could be passed through to a global variable. Timer1 was implemented to determine when approximately one second had occurred, so that revolution frequency could be calculated and average motor speed would reference such. This report recommends for future projects to implement a properly grounded PCB design to reduce noise and voltage fluctuations. Additionally, more accurate means to verify the max speed during benchtop testing should be defined.