

**University of Missouri - Columbia**  
**Electrical and Computer Engineering Department**

**2014 Fall**

# **3 Channel Pulse Width Modulation Measurement System**

**ECE 4220 Real Time Embedded System Final  
Project**

## **Abstract**

This project uses 3 GPIO pins of Port B from the TS - 7250 board. These GPIO pins are connected to the any kind of machine that uses pulse width modulation for motors or servos. A Kernel is created to monitor the input value of GPIOB pins, and send these data to the main program using real time FIFO. In the main program, expecting duty cycle, period, and the error tolerance will be asked to user. Once the user sets up the expected duty cycle, period and the error tolerance, then this program keep monitoring the value of duty cycle and period. If the duty cycle or the period is not what out of the tolerance, then error message is notified to the network server, and user can decide to turn off the certain PWM that error occurred.

## **Introduction**

In many industries, there are a lot of machines that using motors or servos. These motors and servos are usually controlled by PWM to change the speed or location. However, when the faulty PWM occurs, this can cause whole system failure, furthermore there could be human injury or death. There are some concept like watchdog timer, which is monitoring the software failure of embedded system, but the watchdog timer cannot measure the actual period and duty cycle for the PWM. So in this project TS-7250 board will monitor the 3 PWM lines (in this case I'll use Arduino to generate the PWM) through the 3 GPIOB pins. When the wrong PWM occurs, this system can send error to the network using sockets. So the server can decide either turning off the whole system, or restart the system again. This project will decrease risk that the system failure, or human injury and danger.

## Background

First of all, I had to assign the GPIO B resistor address to read the input value of data pins. Then in the kernel module, make a real time that measures GPIOB pins continuously. I assigned period for this real time task to be 1,000,000ns which is 1ms. I have tried 0.5ms for the period, but the real time function was not working properly, so I used 1ms for the period. After reading the data register for port B, using real time named pipes, I had to send these data to the main program. This main program reads value in the named pipes, and analyze the period and duty cycle. These analyzed data will be compared with the user input value. The user input value are expected duty cycle, and the error tolerance percentage. If the user defined duty cycle, and actual duty cycle is different, then the error will be reported to the server using sockets. The server will ask user how to handle the error. User can choose either stop the PWM program, or keep running the program.

## Proposed method/System description/ Implementation

TS-7250 Board

Server

GPIOB2

GPIOB1

Pin5

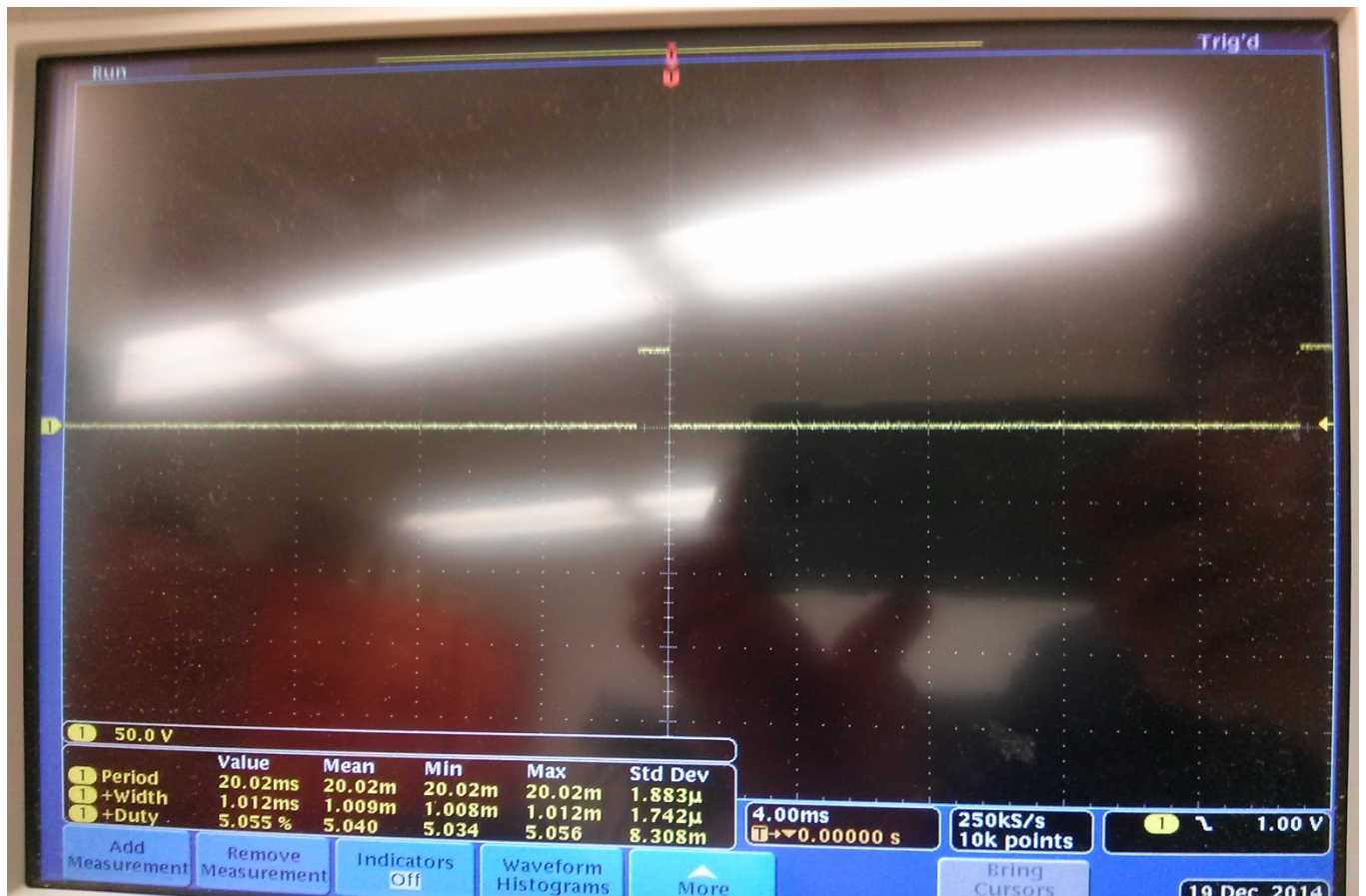
## Named Pipes

## Main Program

I generated PWM in the Arduino pin3, pin5, pin6, and to read the value of each pin, I used GPIOB0,1,2. Then the value goes to main program via named pipes, and the error will be reported to the server via socket. If the server sends data to reset the Arduino, the main program will trigger the reset pin on the Arduino to restart the process.

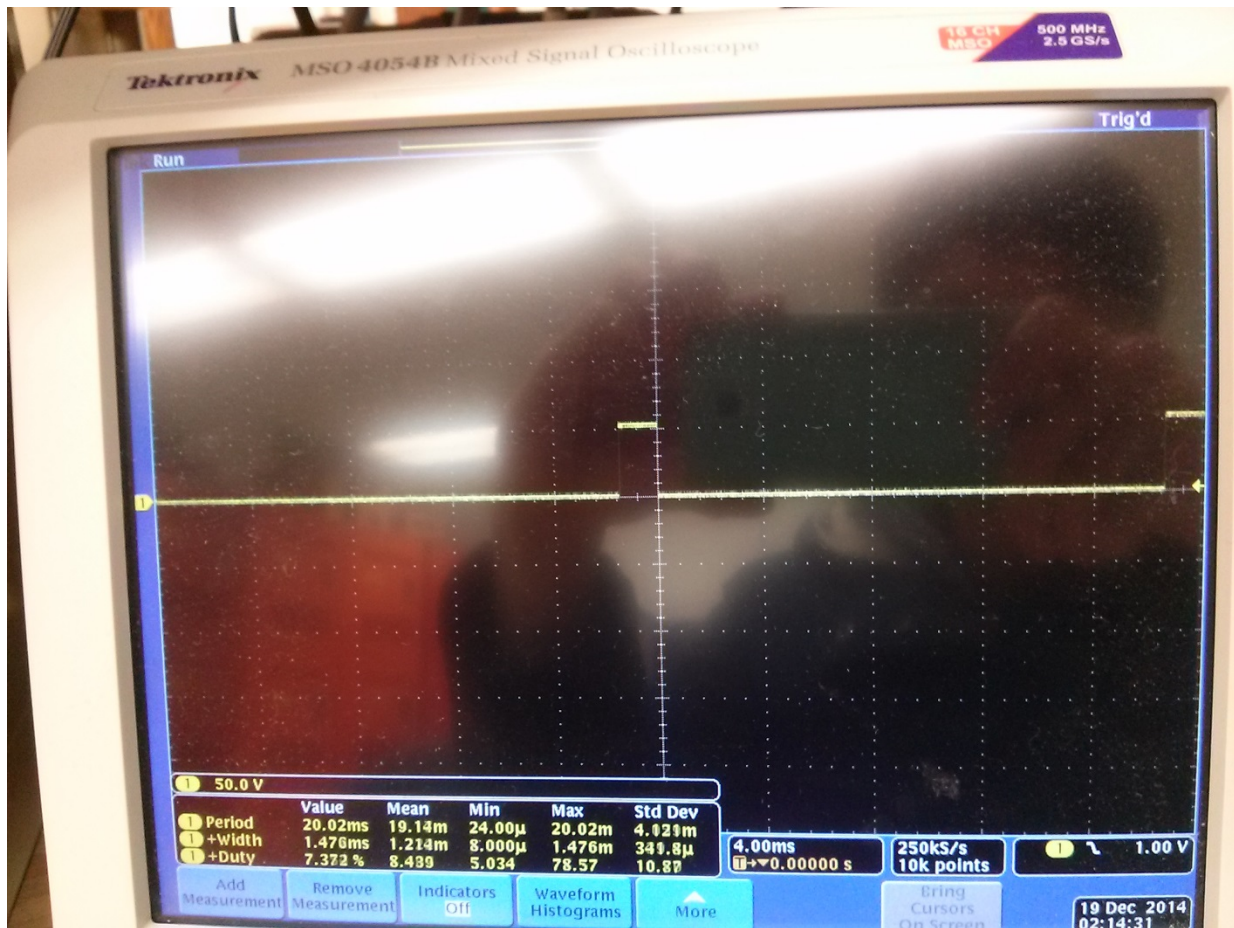
## **Experiments and Result**





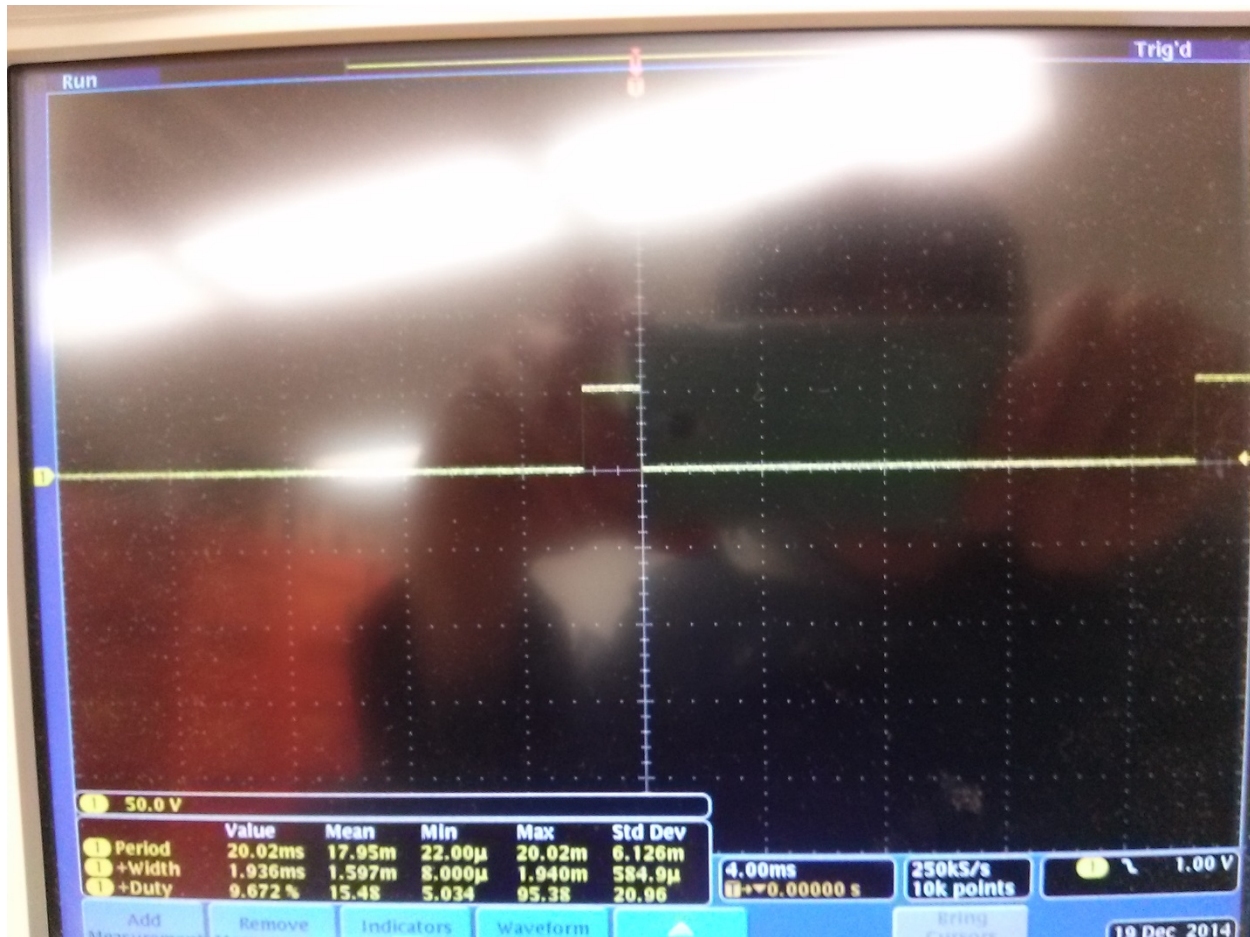
This image shows what Period and Duty cycle is setup for the Arduino pin 3.

Pin 3: Period: 20.02ms, Duty Cycle: 5.055% (1.012ms)



This image shows what Period and Duty cycle is setup for the Arduino pin 5.

Pin 3: Period: 20.02ms, Duty Cycle: 7.372% (1.476ms)



This image shows what Period and Duty cycle is setup for the Arduino pin 5.

Pin 3: Period: 20.02ms, Duty Cycle: 9.672% (1.936ms)

I had to measure the exact PWMs the Arduino is producing. The above images are showing the period, duty cycle, and positive pulse width period from the oscilloscope. Within these result, I had to put the user expected duty cycle, and the error tolerance. Like the image below.



```
[root@ts7250-15 /home/dymp8/project/Release]# ./project 999
Type the duty cycle for channel 0: 5.055
Type the duty cycle error tolerance percentage for channel 0: 20
Type the duty cycle for channel 1: 7.372
Type the duty cycle error tolerance percentage for channel 1: 50
Type the duty cycle for channel 2: 9.672
Type the duty cycle error tolerance percentage for channel 2: 50
```

As shown in the above image, I put first, second, and third channel duty cycle be 5.055%, 7.372%, and 9.672%. Also for the tolerance, I put 20% on channel 0, but channel 1, 2 has 50%. Channel 0 was accurate, but channel 1, and channel 2 were less accurate than channel 0.

```
pin0 Period: 19.98ms Duty cycle: 5.35
pin1 Period: 19.98ms Duty cycle: 4.35
pin2 Period: 19.97ms Duty cycle: 10.27
```

Above image shows the period, and duty cycles on each channel.

		Calculated period on the TS-7250	Actual Duty-cycle	Calculated Duty-cycle on the TS-720
	Actual period			
Pin0	20.02ms	19.98ms	5.055%	5.35%
Pin1	20.02ms	19.98ms	7.372%	4.35%
Pin2	20.02ms	19.97ms	9.672%	10.27%

This table compares the actual period, and calculated period on the TS-7250 board. And it was very accurate result. On the other hand, actual duty-cycle and calculated duty-cycle was not really accurate especially channel 1.

## Main Program

```
Waiting Permission to start this program from the server!!!
Type anything from the server!!!
start

Channel 0 is error. Sending result to the server
Y

Resetting the Arduino Board!!
Arduino is running again. Measuring the PWM again!!
Channel 0 is error. Sending result to the server
N
N

Channel 2 is error. Sending result to the server
Y

Resetting the Arduino Board!!
Arduino is running again. Measuring the PWM again!!
Channel 2 is error. Sending result to the server
^[A^[[BN

Channel 2 is error. Sending result to the server
N

Channel 2 is error. Sending result to the server
N\

Channel 2 is error. Sending result to the server
N

Channel 2 is error. Sending result to the server
N

Channel 2 is error. Sending result to the server
N

Channel 1 is error. Sending result to the server
Y

Resetting the Arduino Board!!
Arduino is running again. Measuring the PWM again!!
Channel 1 is error. Sending result to the server
```

## Server Program

```
start
This was received: start

This was received: Client: 15 PWM Channel 0 failed!

This was received: Do you want to reset the system? [Y/N]
Y
This was received: Y

This was received: Client: 15 PWM Channel 0 failed!

This was received: Do you want to reset the system? [Y/N]
N
This was received: N

This was received: Client: 15 PWM Channel 2 failed!

This was received: Do you want to reset the system? [Y/N]
Y
This was received: Y

This was received: Client: 15 PWM Channel 2 failed!

This was received: Do you want to reset the system? [Y/N]
N
This was received: N

This was received: Client: 15 PWM Channel 2 failed!

This was received: Do you want to reset the system? [Y/N]
N
This was received: N

This was received: Client: 15 PWM Channel 2 failed!

This was received: Do you want to reset the system? [Y/N]
N\
This was received: N\

N
```

When you look at the above image, the Main program waits until the server program gives permission. So the server program send start command, and there was no error, but when I took the channel 0 pin out, it states channel 0 is error and reported error message to the server. Server can reset the system, and it will reset the Arduino. 5 seconds after the restarting, the main program measures the PWM again. This time I plugged out the pin2 and pin2 shows error, This error reported to the server. I did not want to reset the Arduino again, so I send “No” command. So the program started again.

I have tried this program 8 times. And 5 out of 8 was working properly, but sometimes it failed because I put too low error tolerance.

## **Discussion and Conclusions**

The second Channel was not measuring the duty cycle correctly, while the first and third working right. I was trying to fix measuring the accurate result for the second channel, but could not figure it out. Other thing that I had trouble was when I implementing the socket, my program did not wait for the receiving socket function. There was an infinite loop. I realized I was keep sending data to myself. I fixed this problem by not broadcasting data to myself. Another problem is if I measure the data pin value more often in the kernel I could get the precise duty cycle, and period. Since I was measuring the data pin once in a millisecond, it was not accurate for this kind of fast PWM machines. Besides above problems, I had this program quite running well, as I show you in the demonstration.