**Timer0 Interrupt**

*Lab #6*

**Phillip May,** 1639294

**ABSTRACT**

*In this lab I programmed to the PIC16F887 to use tmr0 as a hardware interrupt.*

Experiment Performed on **2 November 2017**

Report Submitted on **12 November 2017**



Department of Computer

Engineering Technology

**INTRODUCTION TO MICROPROCESSORS (247-302)**

*Day Yann Fong*

Table of Contents

[Introduction 3](#_Toc498273303)

[Equipement 3](#_Toc498273304)

[Procedure 3](#_Toc498273305)

[Results and Discussion 4](#_Toc498273306)

[Conclusion 6](#_Toc498273307)

[Additional 7](#_Toc498273308)

[References 7](#_Toc498273309)

# Introduction

In this lab I learned about how to use interrupts on the PIC16F887. In particular, we looked at generating an interrupt on timer0 overflow. By making this occur once every second you can create a 1 second counter.

# Equipement

• PIC16F887

• Pickit3

• 2 100nF capacitors

• 4 LED

• 4 270ohm resistors

•2 10Kohm resistor

•1 push button

•Connecting wires

• 1 7 Segment display

# Procedure

The procedure is provided in the lab instructions.

# Results and Discussion

PART A: Animate a polling on TMR0 interrupt

As the animation shows, the overflow flag for timer 0 must always be scan until there is change. While this is a viable solution it can be problematic when you have lots of things to scan and is power inefficient for the PIC16F887.

PARTB: Animate an ISR for TMR0 interrupt

Code is provided with lab. The code for context saving was taken directly from the datasheet to avoid potential unnecessary complications. Because this is using interrupts instead of polling it’s possible for the main loop to be doing completely unrelated things that take up significant computation time without affecting response on inputs. For example is the PIC is busy calculating something with polling it can’t check to see if it received an input but with interrupts it will pause whatever it is doing and execute code relevant to the input.

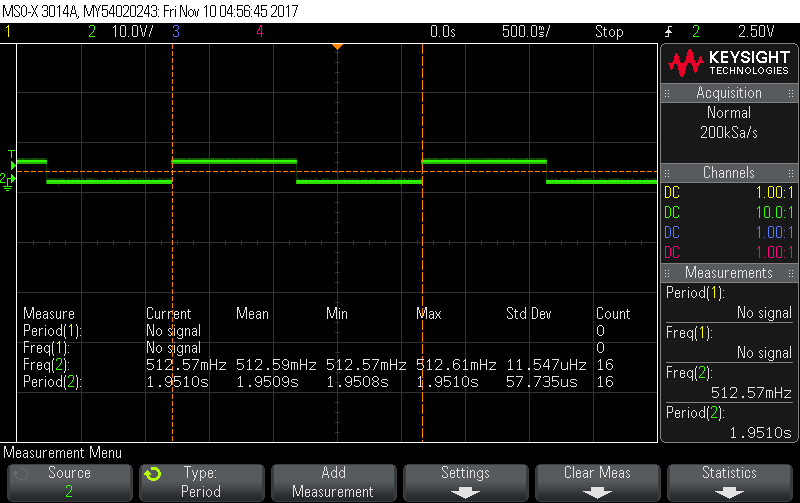
PARTC: 1 second single digit 7-segment counter

9.

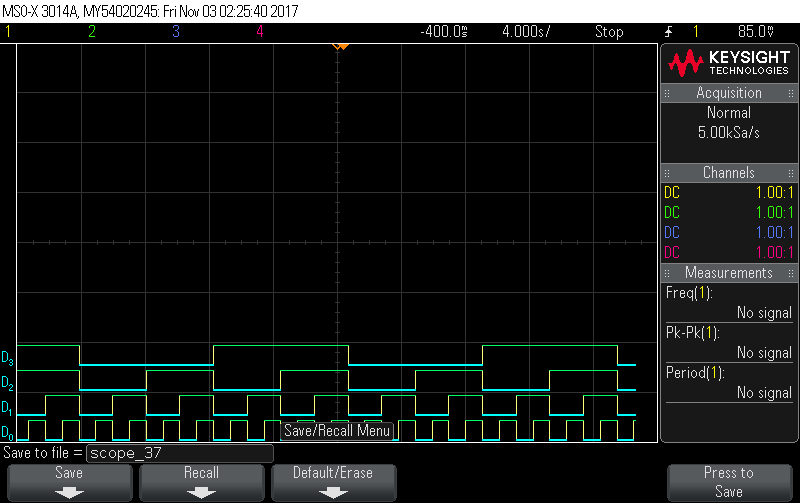
In order to get an interrupt from timer0 every 1 second I calculated a combination of clock cycle and timer prescaler that would cause an overflow every approximately 1 second. The combination I used is a 250 KHz oscillator speed combined with 1:256 prescaler. This results in a clock cycle of 16 micro seconds. 16uS\*256 (prescaler)\*256 (increments to overflow) is 1.0485seconds. Since this is within 5% of 1 second and the PIC16F887 does use the most accurate oscillator ever this is close enough.

10.

As the two scope shots illustrate, the period of the LED representing the least significant bit is 1.9510 seconds. This means 0.975seconds per cycle which is close enough to 1 second. More concerning is that my calculations would have estimated the pic being about 5% too slow instead of 2.5% too fast which goes to show that the PIC16F887’s internal oscillator is not super accurate.



The logic analyser shows that all the bits in the timer are operating as intended. Going from 0000 to 1111 and then overflowing back to 0000 thus repeating the cycle.



11.

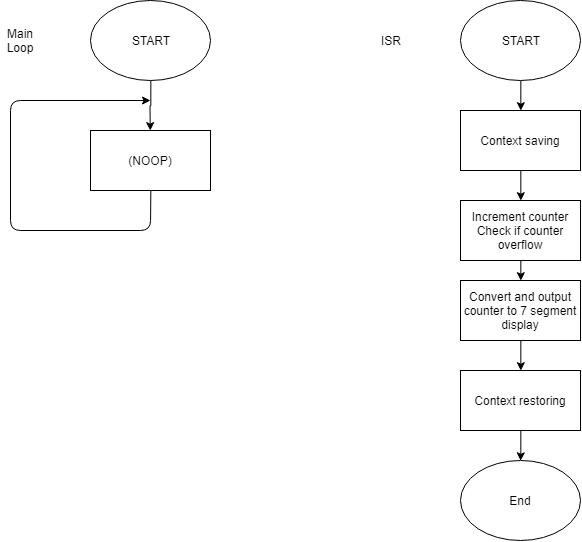
a) I received a common cathode 7 segment display.

b) I would need to request a common cathode decoder for 7 segment display.

c) I opted to use PORTC to drive the 7 segment display directly bypassing the need for a hardware decoder instead requiring a software lookup table that would convert 4 bit binary integers to the common cathode 7 segment display equivalent. The result of this lookup would be pushed to PORTC that was connected to the 7 segment display.

d) Included a copy of the code with the lab.

Flowchart for PARTC:



# Conclusion

In this lab I learn how to use the PIC16F887’s tmr0 as an interrupt that would trigger every desired amount of time. I then used this interrupt to increment a counter and output to 7 segment display. A bit of code optimization I could have done is the check for if the counter is too high for the seven segment. I could have put it in main instead of having everything in the ISR which is generally a bad idea as ideally the ISR should be short and quick.

# Additional

For additional challenge, I programed the PIC16F887 to use interrupts from timer0 to control a counter while also using PORTB as an interrupt that would switch the direction of the counter. This required code at the start of the ISR to identify the source of the interrupts and branch accordingly. I also programmed it to increment/decrement PORTA on overflow/underflow. I included a copy of the code with this report. The possible code optimizations are the same as the main lab, move overflow checks to the main loop instead of having everything in the ISR.

# References

|  |  |
| --- | --- |
| [1] | Microchip Technology Inc., "MPASM Assembler, MPLINK Object Linker, MPLIB Object Librarian User's Guide," Microchip Technology Inc., 2013. [Online]. Available: http://ww1.microchip.com/downloads/en/DeviceDoc/33014L.pdf. [Accessed 20 9 2017]. |