# 

# **Interrupt & Timer ISRs**

## *C8051 Microcontroller*

Jeffrey Pistacchio

Christopher Pybus

Microprocessor Systems

26 September 2016

**Introduction**

The purpose of this exercise is to demonstrate the purpose and use of interrupts and interrupt service routines (ISRs). This was accomplished by using a pushbutton to generate an interrupt which would be displayed on a computer screen. Furthermore, timer controls and configurations were explored by writing two scripts, each generating a stopwatch that counted in tenths of a second. One script employed rounding in its algorithm and the other was precise to the accuracy of the internal clock frequency. Finally, these two concepts were combined to measure the reaction time of a user by prompting the user to push a button after the computer screen flashes.

**Methods and Procedures**

Each of these sections represented a specific goal for the lab. Goal 1 included wiring a push button to generate an external interrupt and displaying the number of times that this interrupt is recorded on the computer screen. Goal 2 included using two different scripts to create two stopwatches. The first stopwatch simply needed to count tenths of a second by rounding interrupts. The second stopwatch needed to count tenths of a second exactly to the precision of the external clock (50 ppm). Goal 3 included making a reaction time tester by asking the user to push a button in response to a flash on the computer screen.

**Goal 1**

Goal 1 was achieved mostly through initializing ports and the cross bar. Specifically the crossbar was configured so that the external interrupt “/INT0” was routed to pin 0.2. Then Port 0 was initialized so that pin 2 was configured to open drain. From here one of the push buttons on the protoboard was connected in series to Pin 0.0 and Ground. In the software, every time the pushbutton was pressed an interrupt was generated and a variable containing the number of presses was incremented in the interrupt service routine. This value was printed to the terminal on the computer screen.

One key aspect to using push buttons is debouncing. Whenever a pushbutton is pressed the connection vibrates and opens and closes the circuit many more times than intended. In order to filter out these unintended presses, a debouncing loop is implemented in the main code. As soon as an interrupt is generated due to a pushbutton press a flag is triggered. This generates a waiting loop in the main code which forces the software to ignore the following fantom pushbutton presses.

**Goal 2**

Goal 2 was more software oriented. The stopwatch program required the setup of Timer 2 and the corresponding interrupt for Timer 2. This timer was configured to use an external crystal oscillator that used PLL to create a 49.7664 MHz clock signal. Timer 2 used this clock to increment a variable every 12 clock ticks. In other words, the timer used the SYSCLOCK/12 setting. Whenever this 16 bit variable overflowed, it would generate an interrupt. Timer 2’s interrupt service routine incremented global variable, ‘intrpt’, that was used to keep time for the stop watch.

Every six Timer 2 interrupts was about a tenth of a second. More accurately, a tenth of a second corresponded to about 6.328125 interrupts. This calculation was made by dividing the clock frequency (49,766,400/12 ticks per second) by the number of size of the Timer 2 variable (2^16 ticks per overflow). This results in the exact number of overflows per second. However, the 6.328125 overflows per tenth of a second was rounded to 6 overflows per tenth of a second, in accordance with the lab’s directions.

The main section of code of the software simply included an if-statement that incremented the time on the stopwatch when the intrpt variable reached a value of six.

In comparison, to create a stopwatch that was accurate to the precision of the clock hardware, the objective became to configure the Timer such that a tenth of a second was an integer number of overflows. The simplest way to do this was to adjust the number ticks per overflow. The number of ticks per overflow can be decided by selecting a start value for the Timer variable. For simplicity it was selected that one interrupt correspond to one one hundredth of a second.

Similar to the steps above, the number of ticks for one interrupt to be a hundredth of a second can be found by dividing the clock frequency by 100. Then subtracting the result from 2^16 (the size of the variable) gives the necessary start value. For this case the start value, set on TMR2H and TMR2L SFRs, was 24,064 in decimal or 0x5E00 in hexadecimal. This gave 41,471 clock ticks per overflow. Therefore, ten overflows corresponded to exactly a tenth of a second. In addition, this script used the same method as the previous script to increment and display the timer.

**Goal 3**

The reaction time test had a number of features. Rather than displaying time elapsed to the tenth of a second, time was displayed to the hundredth of a second. In order to get a nonsingular sample, the user was tested five times and the average reaction time was displayed. Furthermore, using ANSI terminal manipulation learned in Lab 1, a section of the screen changes color, from blue to yellow, to signal the user to press the push button. A fourth feature is that if the user tries to “cheat” and press the button before

In order to accomplish this, the functionality from the programs from the first two goals needed to be integrated. The main addition was using two interrupt service routines to increment the stopwatch and register button presses.

A for loop was used to give the user five tries to test their reaction time. The number of timer 2 interrupts between the screen flashing and the pushbutton being pressed is displayed and recorded as the reaction times. After every fifth try, the avertage of the previous five tries is displayed as the user’s reaction time.

In order to increase the difficulty of the test, the program waits a random amount of time, between two and seven seconds between tests. This prevents the user from predicting the flash of the screen and “cheating”.

**Results and Analysis**

**User tests:**

TEST #1

|  |  |
| --- | --- |
| # | Reaction Time (sec) |
| 1 | .38 |
| 2 | .20 |
| 3 | .30 |
| 4 | .19 |
| 5 | .25 |
| AVG | .246 |

TEST # 2

|  |  |
| --- | --- |
| # | Reaction Time (sec) |
| 1 | .40 |
| 2 | .30 |
| 3 | .20 |
| 4 | .26 |
| 5 | .24 |
| AVG | .28 |

TEST # 3

|  |  |
| --- | --- |
| # | Reaction Time (sec) |
| 1 | .23 |
| 2 | .22 |
| 3 | .29 |
| 4 | .25 |
| 5 | .27 |
| AVG | .252 |

The three tests conducted gave reaction times of 0.246, 0.28, and 0.252 seconds. According to backyardbrains.com, the average reaction time to visual stimulus is about 0.25 seconds. This test was conducted using multiple trials (three), and unremarkably average test subjects to eliminate bias and reduce statistical anomaly. Since the test results of this experiment were similar to the expected result of 0.25 seconds and evidence of bias in the analysis of the program cannot be found, the reaction time tester can be confidently evaluated as proficient.

**Conclusion**

The team successfully accomplished Lab 2 by making four programs that accomplished all goals. Dividing the project into separate goals made the exercise significantly more approachable due to implementing the divide and conquer mindset. Each goal was simple enough that each group member could complete a goal by the time they came to class, and spend lab time debugging code and building hardware.

Some changes that could be made to make the program more effective would possibly be to make a more efficient pushbutton debouncer. The length of the debounce loop was relatively arbitrary, determined primarily based on trial and error. A more effective method would be to use a debouncing circuit in series with the pushbutton.

**Appendices**

**Schematic**