**CMPE-250 Laboratory Exercise Ten**

**Time Driver Input Timing**

By submitting this report, I attest that its contents are wholly my individual writing about this exercise and that they reflect the submitted code. I further acknowledge that permitted collaboration for this exercise consists only of discussions of concepts with course staff and fellow students; however, other than code provided by the instructor for this exercise, all code was developed by me.

John Judge

Performed 4/14/16

Submitted 4/21/16

Lab Section 02

Instructor: Dr.Shaaban

TAs: Peter Muller

Stephen Moore

Connor Goss

Lecture Section 01

Professor: Allesandro Sarra

**Abstract**

In this exercise a time driver was developed for the Freescale Freedom KL46Z Microcontroller. The objective of the exercise was to implement interrupt-based timing measurements accurate within 0.01s. The exercise tested the knowledge and use of interrupt service routines. The program written in this exercise was a security measure for password protected information in which the user would be asked for a password and had 5 seconds to enter the correct password or they were denied access. Upon providing the correct input in the specified amount of time they were granted access. Otherwise, the user could continue to enter passwords. The timing was done using channel zero of the periodic interrupt timer (PIT). An interrupt service routine, *PIT\_ISR*, and its initialization subroutine, *Init\_PIT\_IRQ,* were written to keep track of the amount of time that passed from when the interrupt was enabled and when it was disabled. After the program was developed and tested it was found that the objective was achieved and the measurements were accurate. The exercise was successful and several lessons were learned.

**Procedure**

To begin the exercise the subroutine *Init\_PIT\_IRQ* needed to be written to initialize the PIT. In order to initialize the PIT, first, all interrupts needed to be disabled. The PIT was then initialized by logically ORing the SIM\_SCGC6 and SIM\_SCG\_PIT\_MASK equates. This set the 7th bit of SIM\_SCGC6 to 1 thus enabling the PIT. The result of the ORing operation was stored. From here, timer 0 needed to be disabled. This was achieved by using a bit clear operation between PIT\_CHO\_BASE and PIT\_TCTRL and storing the result as the new PIT\_TCTRL value. Next the PIT interrupt priority was set. Then all PIT interrupts were unmasked by storing PIT\_IRQ\_MASK as the NVIC\_ISER. Then the PIT timer module was enabled and a request for an interrupt was set to every 0.01 seconds. The PIT timer channel 0 was then enabled for interrupts using the PIT\_TCTRL\_CH\_IE mask. To end the subroutine any pending interrupts were unmasked and the used registers as well as the program counter were popped off the stack.

Next the interrupt service routine, *PIT\_ISR*, was written. To begin the ISR all pending interrupts were masked and a variable called *RunStopWatch* was initialized. This represented a stop watch being pressed to run. This ISR incremented another variable, *Count*, for each time the interrupt was requested. This meant that he stop watch increased for each 0.01 second that passed from when the stopwatch was initialized. For each time a request was generated the incremented *Count* value was stored. To end the subroutine the interrupt condition was cleared and the all pending interrupts were unmasked.

The main function was then written. This provided the functionality of the exercise. When run, the variables *Count* and *RunStopWatch* were initialized. Then the prompt for the user was printed to the console using the *LengthStringSB* and *PutStringSB* subroutines from previous exercises. In order for these to work correctly the *Enqueue, Dequeue, GetStringSB, LengthStringSb, PutNumHex, PutNumU, PutNumUB, DIVU, PutChar* and *GetChar* subroutines from the previous exercise also needed to be imported into the program. The main program would then move the console to the next line and print a carrot. From here, it polled for the user’s input of the access code. Once the user pressed the enter key the program would stop triggering the ISR and print the value of *Count* followed by “ x 0.01s” to show the amount of time that had passed while the user entered the password. If the code entered matched the constant *AccessCode* and the PIT\_ISR had not been triggered over 500 times then the program printed “—Access granted” to the console followed by “Mission completed!” on the next line. Had either on of the conditions not been achieved, the program would print “--access denied” followed by the prompt for the password.

**Results**

After the program was written it was tested. In order to ensure the program was written correctly the program was tested under the conditions of no code being entered, a wrong code being entered, the correct code being entered after more than 5.0 seconds had passed, and finally the correct code being entered in enough time. A capture of the terminal output from the exercise can be seen in Figure (1.0). The program correctly handled these cases by denying access to the user for the first three cases and then granting access to the user and terminating the program when the correct code had been entered in the right amount of time.

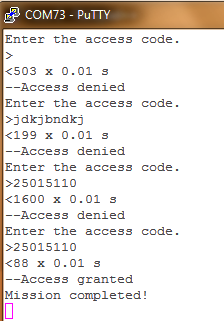


Figure (1.0): The Console Output

After the testing occurred, the memory ranges of the code were analyzed and a capture of the memory map can be seen in Figure (1.1). From the map file it was concluded that the total executable code was between the range 0x00000410 and 0x000008CF in ROM. The *PIT\_ISR* was located at 0x00000631. The constants in ROM were located between 0x00000204 and 0x00000264. These constants included the prompt string, the access code, the mission complete string, the access denied string, and the string showing the amount of time passed.

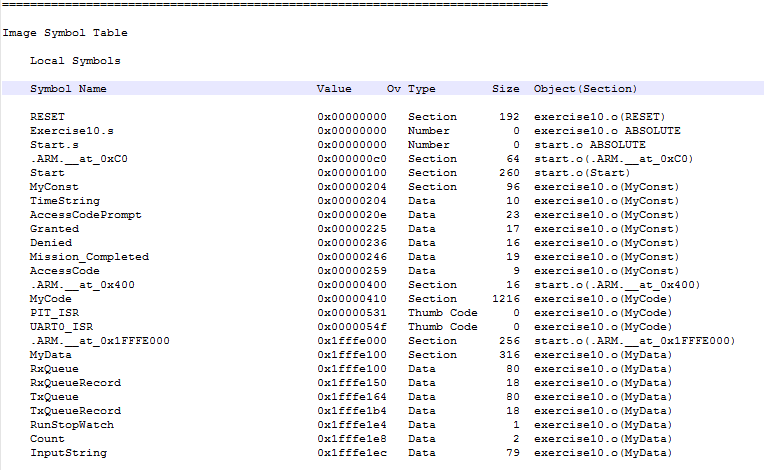


Figure (1.1): Memory Map of Program

**Conclusion**

In this exercise a time driver was developed for the Freescale Freedom KL46Z Microcontroller. The objective of the exercise was to implement interrupt-based timing measurements accurate within 0.01s. In order to achieve this objective the periodic interrupt timer was used to produce an interrupt every 0.01s. The subroutine *Init\_PIT\_ISR* was written to initialize all the starting conditions necessary for the PIT and the interrupt to function correctly. An interrupt service routine, *PIT\_ISR*, was written to increment a count variable from when the interrupt was enabled and when it was disabled. After the program was developed it was tested using several test cases to prove the code had been developed as intended and operated successfully. The program preformed as expected and the timing matched what was expected. As a result, the objectives of the exercise were met. The exercise was determined to be successful because the program operated as expected and the use of interrupt service routines as well as the use of the PIT were handled correctly. The implementation and use of the both of these proved that the theory had been learned. The exercise was then determined to be successful and the lessons learned during this exercise can be used in future exercises.