**CMPE-250 Laboratory Exercise Seven**

**Circular FIFO Queue Operations**

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.



Dean Trivisani

Performed October 17, 2017

Submitted October 24, 2017

Lab Section 01L2

Instructor: Melton

TA: Donald Mannise

Michael Baumgarten

Matthew Toro

Lecture Section 01

Professor: Melton

**Abstract**

This exercise utilized the Keil MDK-ARM Microcontroller Development Kit to write a Cortex-M0+ assembly language program (for the NXP Freedom KL46Z microcontroller with polled serial I/O and the universal asynchronous receiver/transmitter (UART) module) that implemented a functional interactive queue system operated through the command prompt. The purpose of this was to develop greater understanding of queue systems in Assembly Language programming, as well as increase familiarity with creating programs that interact with physical systems using polling. A project was created in Keil µVision and subroutines for all queue operations, as well as the subroutines required to ensure proper formatting and output of each operation, were written and integrated with subroutines used for interaction with the microcontroller and the command prompt. The main program code was then written to implement these subroutines in a program that would take instructions from the command prompt to operate the queue. The program was then run on the KL46Z board using the µVision debugger, and the program responded correctly for all the cases tested. Therefore, the program functioned correctly.

**Procedure**

A project directory and project files were created in µVision using the NXP MKL46Z256VLL4 microcontroller on the NXP FRDM-KL46Z Freedom development platform, and subroutine Init\_UART0\_Polling for initialization of the KL46 UART0 polled serial I/O operation at 9600 baud using eight data bits, no parity, and one stop bit, was implemented. Subroutine InitQueue was written to initialize the queue record structure at the pointer in register R1 at the pointer in register R0 and of size R2. The dequeue operation was written to remove a single character from the queue (the pointer to the record structure for which was contained in R1) to R0, clear the PSR Carry flag, and report success if the queue was not empty. If it was empty, the Carry flag would be set and report a failure. The enqueue subroutine attempted to add a single character from R0 to the queue and clear the Carry flag (also using the pointer in R1) if possible, and would report success. If not possible because the queue was full, the Carry flag would be set and failure reported. The PutNumHex subroutine was written to print an unsigned word value in R0 to the command prompt in hexadecimal format. PrintNumUB was written for a similar purpose, but instead of printing the hexadecimal form of the value in R0, it instead printed the decminal form. These were then used for the printing of the status of the queue in the STATPRINT subroutine, where the values of the InPointer and OutPointer would be printed in hexadecimal format and the number of enqueued characters would be printed in decimal format. Predetermined text outputs were defined as constants in the MyConst AREA, and the space for the queue buffer (QBuffer) and the queue record (QRecord) was allocated using constants defined with EQUates. Additional subroutines were written to improve readability (such as NEWLINE which would print a carriage return and line feed, and PRINTCMD which would print each command entered), however these were inconsequential to the operation of the program. The main program code was then written such that a prompt containing all possible queue commands was displayed to the user through the command line. If the command the user tried to input was not a valid command, no action would occur (this does not include incorrect command letter case, as the program checked the case of the input character using range checking and corrected it if necessary). Otherwise, the corresponding operation would occur using the appropriate subroutines. The program was then tested on the KL64Z board using the µVision debugger, where correct operation was ensured for different command character cases, empty queue operations, populated queue operations, full queue operations, and circular buffer operations. All were found to produce correct functionality.

**Results**

The program executed as intended, and the full functionality of a circular FIFO queue was able to be manipulated using command line inputs. The memory ranges given in Table 1 show the memory addresses and sizes of the executable main program portion of the code, the constants in ROM (the string constants used for printing to the command line), and the queue record structure and queue buffer in RAM. The queue record structure size is of note, as 18 bytes is the necessary space to support queue sizes up to 255, as the queue management data stored as InPointer, OutPointer, BufferStart, and BufferPast each require 4 bytes of space, and BufferSize and NumberEnqueued each require one byte, adding up to a total of 18 bytes. The queue buffer size is shown to be 4, which is set with an EQUates statement in the code and can be changed to allow for larger queues.

Table 1: Memory Ranges for Circular FIFO Queue Code AREAs

|  |  |  |  |
| --- | --- | --- | --- |
| AREA | Start | End | Size (Bytes) |
| Executable Code | 0x00000410 | 0x0000083B | 1068 |
| Constants in ROM | 0x000001FC | 0x000002A3 | 168 |
| Queue Record Structure (RAM) | 0x1FFFE104 | 0x1FFFE115 | 18 |
| Queue Buffer (RAM) | 0x1FFFE100 | 0x1FFFE103 | 4 |

The command prompt text shown in Figure 1 consists of various tests of the program the first two command input characters, ‘s’ and ‘S’, produce the same result, showing that the case of the command is unimportant as lonSg as the character is correct. Characters are then enqueued, with reports of success and the queue status, until the queue buffer size has been met and the queue is full. When another enqueue attempt is made, the operation reports a failure and the status of the queue. The print command is then given, and the contents of the queue are printed to the command line. The characters are then successfully dequeued (with print and status commands in between to show the progression of the dequeue operations) until the queue is empty. When the dequeue command is given again with an empty queue, the operation reports a failure and the queue status. Finally, the help command is input, and a list of the commands characters and their function is printed.

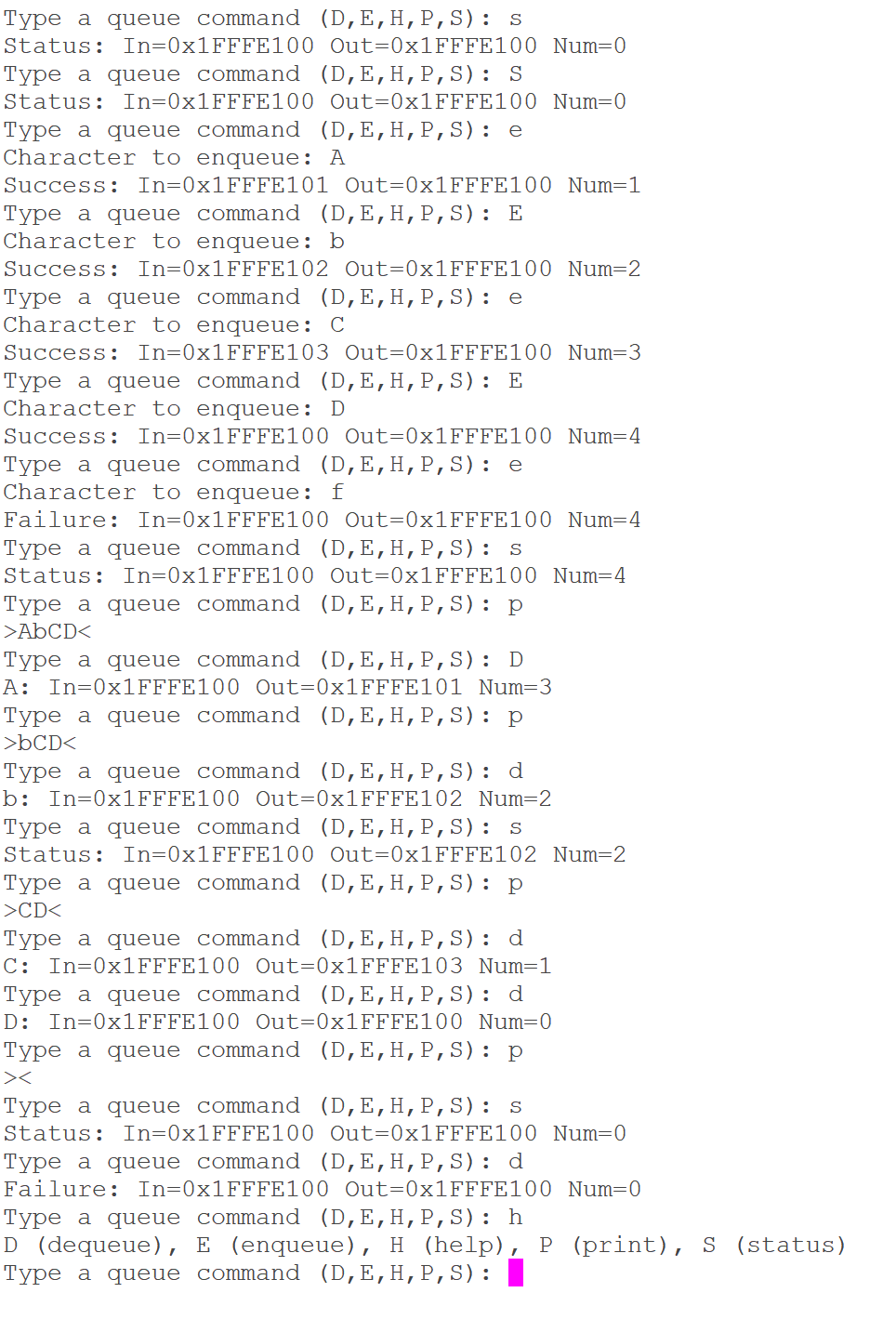


Figure 1: Command Prompt Test of Circular FIFO Queue Operation

**Conclusion**

This exercise gave insight into the functionality of circular FIFO queues, as well as how a basic menu-based polled I/O program is written and implemented. It also greatly improved familiarity with the debugger software, as a specific bug involving the inability to recognize the KL64Z device required numerous hours of troubleshooting and learning more about the hardware to resolve. This ultimately, however, led to the successful implementation and operation of the program. Knowledge of these things is essential to computer engineering, as they form the basis with which interactions between software and hardware are facilitated.