CMPE-250 Assembly and Embedded Programming

Laboratory Exercise Nine

Serial I/O Driver

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. Other than code provided by the instructor for this exercise, all code was developed by me.

Shubhang Mehrotra
Performed 1st April 2021.
Submitted 8th April 2021.

Lab Section 1
Instructor: Muhammad Shaaban
TA: Anthony Bacchetta
Aidan Trabuco
Sam Myers
Payton Burak

Lecture Section 1
Lecture Instructor: Dr. Roy Melton

Procedure Screen Capture

The activity explored the KL05Z Freedom Board further by writing a program to perform Circular FIFO Queue Operations upon interrupt-based serial communication with the UART instead of the previous polling-based implementation.

To initialize the connection to the UART with Interrupt based serial communication an Init_UART0_IRQ subroutine was defined which modified the previous Init_UART0_Polling to interrupt in the NVIC and initialize Receive and Transmit Queues. An interrupt service routine (ISR) was also written to handle interrupting the UART0 based upon whether the Transmit Interrupt as enabled or the Receive input was enabled.

GetChar and PutChar operations were modified to utilize the receive and transmit queues and work with the interrupt-based communication.

This connection to the board was again established via UART Polling at 9600 Baud rate using PuTTY and its “Serial” connectivity functionality. The code for interrupt-based queue operations was downloaded to the KL05Z board using the Keil IDE. It was then tested against the following cases:

- Uppercase and Lowercase commands
- Commands with an empty queue
- Commands with partially full queue
- Commands with a full queue
- Commands with Circular queue

The results of the test run are shown in Figure 1 and Figure 2.

```
Type a queue command (D, E, H, P, S): h
D (dequeue), E (enqueue), H (help}, P (print), S (status)
Type a queue command (D, E, H, P, S): p
><
Type a queue command (D, E, H, P, S): s
Status:     In=0x01FFFFD30   Out=0x01FFFFD30   Num= 0
Type a queue command (D, E, H, P, S): e
Character to enqueue: T
Success:    In=0x01FFFFD31   Out=0x01FFFFD30   Num= 1
Type a queue command (D, E, H, P, S): e
Character to enqueue: o
Success:    In=0x01FFFFD32   Out=0x01FFFFD30   Num= 2
Type a queue command (D, E, H, P, S): e
Character to enqueue: n
Success:    In=0x01FFFFD33   Out=0x01FFFFD30   Num= 3
Type a queue command (D, E, H, P, S): e
Character to enqueue: y
Success:    In=0x01FFFFD30   Out=0x01FFFFD30   Num= 4
Type a queue command (D, E, H, P, S): e
Character to enqueue: i
Failure:    In=0x01FFFFD30   Out=0x01FFFFD30   Num= 4
Type a queue command (D, E, H, P, S): p
>Tony<
Type a queue command (D, E, H, P, S): d
T:          In=0x01FFFFD30   Out=0x01FFFFD31   Num= 3
Type a queue command (D, E, H, P, S): d
o:          In=0x01FFFFD30   Out=0x01FFFFD32   Num= 2
Type a queue command (D, E, H, P, S): P
>ny<
Type a queue command (D, E, H, P, S): e
Character to enqueue: 3
Success:    In=0x01FFFFD31   Out=0x01FFFFD32   Num= 3
Type a queue command (D, E, H, P, S): e
Character to enqueue: 4
Success:    In=0x01FFFFD32   Out=0x01FFFFD32   Num= 4
Type a queue command (D, E, H, P, S): e
Character to enqueue: n
Failure:    In=0x01FFFFD32   Out=0x01FFFFD32   Num= 4
Type a queue command (D, E, H, P, S): P
>ny34<
Type a queue command (D, E, H, P, S): D
n:          In=0x01FFFFD32   Out=0x01FFFFD33   Num= 3
Type a queue command (D, E, H, P, S): d
y:          In=0x01FFFFD32   Out=0x01FFFFD30   Num= 2
Type a queue command (D, E, H, P, S): d
3:          In=0x01FFFFD32   Out=0x01FFFFD31   Num= 1
Type a queue command (D, E, H, P, S): d
4:          In=0x01FFFFD32   Out=0x01FFFFD32   Num= 0
Type a queue command (D, E, H, P, S): d
Failure:    In=0x01FFFFD32   Out=0x01FFFFD32   Num= 0
```

Figure 1. Results

```
Type a queue command (D, E, H, P, S): P
>ny34<
Type a queue command (D, E, H, P, S): D
n:          In=0x01FFFFD32   Out=0x01FFFFD33   Num= 3
Type a queue command (D, E, H, P, S): d
y:          In=0x01FFFFD32   Out=0x01FFFFD30   Num= 2
Type a queue command (D, E, H, P, S): d
3:          In=0x01FFFFD32   Out=0x01FFFFD31   Num= 1
Type a queue command (D, E, H, P, S): d
4:          In=0x01FFFFD32   Out=0x01FFFFD32   Num= 0
Type a queue command (D, E, H, P, S): d
Failure:    In=0x01FFFFD32   Out=0x01FFFFD32   Num= 0
Type a queue command (D, E, H, P, S): p
><
Type a queue command (D, E, H, P, S): e
Character to enqueue: a
Success:    In=0x01FFFFD33   Out=0x01FFFFD32   Num= 1
Type a queue command (D, E, H, P, S): p
>a<
Type a queue command (D, E, H, P, S):
```

Figure 2. Results

Memory Ranges

For Lab Activity 8, the Listing file and the Map file were generated and analyzed for the memory addresses of the items listed in Table 1, which lists the start addresses and the size occupied by the item in the memory in bytes.

Table 1. Memory Ranges

Object	Memory Address Start	Memory Address End (Start + Size - 1)	Size (bytes)
Executable code in MyCode AREA	0x00000410	0x0000097F	1392
UART0 ISR code	0x00000594	0x000005E1	78
Constants	0x000001C4	0x0000028B	200

Table 2. RAM Usage

Object	Memory Address Start	Memory Address End (Start + Size - 1)	Size (bytes)
Program Queue Buffer	0x1FFFFDC8	0x1FFFFE17	80
Program Queue Record	0x1FFFFE18	0x1FFFFE29	18
Receive Queue Buffer	0x1FFFFD00	0x1FFFFD4F	80
Receive Queue Record	0x1FFFFD50	0x1FFFFD61	18
Transmit Queue Buffer	0x1FFFFD64	0x1FFFFDB3	80
Transmit Queue Record	0x1FFFFDB4	0x1FFFFDB5	18

The Start Address for the objects were obtained from the Map file. Under the section marked “Image Symbol Table,” is a table which lists the execution address of each entity in the source file. Upon inspection of the table, the start addresses, and the size can be obtained. The End address, if required, can be obtained by adding the start address and the size and subtracting 1.