**Introduction**

The activity outlines an activity where we are required to interface with an IO peripheral, namely the UART peripheral, to showcase simple text processing and manipulation. The high level problem statement is simple enough. We are required to a) read input characters from the UART peripheral thru the RX FIFO, b) invert the cases of each letter that is encountered, and lastly, c) echo back all input characters with their transformation (if applicable) by writing them to the TX FIFO.

**Procedure/Discussion**

As initial mentioned, this activity essentially does simple text manipulation. There are however certain aspects that make this activity really challenging. They are enumerated below in the order they were encountered during the making of this activity.

1. The first one is that ARM assembly is the requirement and not high level language like C. Because of the nature of assembly language, the entire activity must be broken down into the most fundamental component steps - e.g. a simple one line variable assignment in C or in any high-level language, would need to be broken down into three or more lines of assembly code in order to work. The ARMv7 user manual came in very handy for this as it provided instruction listings and explanations that guide developers in how the CPU will behave when executing the code.
2. Familiarity with the peripheral behavior and how to use it. Accessing of data and status is usually done in the peripheral’s register space. More than one piece of data is usually cramped into a 32-bit register which means, more often than not, further bit-wise manipulation is needed to be done in order to extract or set the correct data. The website for JTAG UART [1] fortunately provides sample codes for basic reading and writing which provided the basis of my own algorithm when interfacing with the JTAG UART.

To prove correct program behavior, the screenshots with short paragraph description below are attached for review.

Graphical user interface, text, application, email

Description automatically generated

Figure - Before program run

The screenshot above shows the initial state of the program. The program counter is shown to be at the first instruction (highlighted yellow). Additionally, the RX FIFO has been inputted with an initial test value “%ArM 7” per requirements a) at least 1 lowercase, b) at least one uppercase, c) at least 3 non-letter character, and no more than 6 characters.

Graphical user interface, application

Description automatically generated

Figure - After program run

The screenshot above shows the final state of the program. The program counter is shown to be at the final “nop” (highlighted yellow). Additionally, the RX FIFO shows is it now empty since the program has successfully popped all values from it. Furthermore, the JTAG UART text window now shows the text manipulation applied to the input string “%ArM 7”, and it has now become “%aRm 7”. The Upper case ‘A’ and ‘M’ have been converted to lower case and the lowercase ‘r’ has been converted to upper case. No Transformation has been applied to ‘%’, ‘ ‘(space character), and ‘7’ since they are all non-letter characters.

**Analysis**

**Conclusion**

**References**

**[1]** JTAG UART <https://www-ug.eecg.utoronto.ca/desl/nios_devices_SoC/dev_jtaguart.html>

[2] ARMv7-M Architecture Reference Manual<https://documentation-service.arm.com/static/606dc36485368c4c2b1bf62f?token=>

* 1. load instructions and derivatives(ldr) - A7.7.43
  2. store instructions and derivatives (str) - A7.7..161
  3. mov instruction (mov) - A7.7.76
  4. branch instructions - A7.7.12
  5. orr instruction - A7.7.89
  6. conditional execution mnemonics - A7.3
  7. Application Program Status Register (APSR) - A2.3.2

1. **GNU Assembler User Manual:** <https://www.ndsl.kaist.edu/ee209/references/gnu-assembler.pdf>
   1. .equ symbol, 7.21, page 45
   2. Labels, 5.1, page 33
2. **Loading constants in assembly for Arm Architecture:** <https://community.arm.com/arm-community-blogs/b/architectures-and-processors-blog/posts/how-to-load-constants-in-assembly-for-arm-architecture>

**Appendix**

.equ JTAG\_UART\_BASE\_ADDR, 0xFF201000

.equ JTAG\_UART\_DATA\_REG\_OFFSET, 0

.equ JTAG\_UART\_DATA\_VALID, (1<<7)

.equ JTAG\_UART\_CONTROL\_REG\_OFFSET, 4

.equ JTAG\_UART\_CONTROL\_WRITE\_IRQ\_PENDING, (1<<9)

.equ JTAG\_UART\_CONTROL\_READ\_IRQ\_PENDING, (1<<8)

.global \_start

\_start:

/\* r0 is global defined to be the base jtag uart register from now on \*/

ldr r0,=JTAG\_UART\_BASE\_ADDR

/\* INITIALIZATION. we put the jtag uart peripheral into a known and stable state

\* 1) disable both read and write interrupts by writing zero to bits 1, 0

\* 2) clear all pending interrupts by writing writing 1 to bits 9, 8

\*/

ldrb r1, =0

orr r1, r1, #(JTAG\_UART\_CONTROL\_WRITE\_IRQ\_PENDING |

JTAG\_UART\_CONTROL\_READ\_IRQ\_PENDING)

strb r1, [r0, #JTAG\_UART\_CONTROL\_REG\_OFFSET]

/\* POLL FOR CHARACTER INPUT. \*/

poll: ldrb r1, [r0, #(JTAG\_UART\_DATA\_REG\_OFFSET+1)]

ands r2, r1, #JTAG\_UART\_DATA\_VALID

beq poll

/\* extract the number of characters to read, r2 will hold the max loop count \*/

ldrh r2, [r0, #(JTAG\_UART\_DATA\_REG\_OFFSET+2)]

loop:

ldrh r1, [r0, #(JTAG\_UART\_DATA\_REG\_OFFSET)]

mov r3, #('a' - 'A') /\* bias value for upper case to lower case \*/

/\* check input is a letter (only letters can have capital or small versions) \*/

and r1, #0xFF /\* isolate the byte field which contains the letter \*/

cmp r1, #'A'

blt print

cmp r1, #'Z'

blt change\_case

mov r3, #('A' - 'a') /\* bias value for lower case to upper case \*/

cmp r1, #'a'

blt print

cmp r1, #'z'

bge decrement

change\_case:

add r1, r3

print:

str r1, [r0]

decrement:

adds r2, #-1 /\* decrement the loop counter \*/

bgt loop

nop

b . /\* stop processing here. \*/