

Homework 4 - Resource Handling

Zackary McClamma^{*†}

October 28, 2019

1 Introduction

This homework assignment was focused on resource sharing and consisted of running the $\mu\text{C}/\text{OS-II}^{\text{TM}}$ operating system with two tasks, a mutex, and custom UART print function. Each task prints the alphabet, one in lowercase and the other in uppercase. The basic idea is that if the tasks have short enough delays after the print statements they will interfere with one another and cause the output to become jumbled. Adding the mutex fixes this issue as it locks the print resource down while a task is printing and the other task has to wait for the resource, in this case the UART port.

2 System Design

This is a fairly simple system the only peripheral is the UART port and a clock. The operating system does most of the work without much input. The UART port is used to connect the device to an external computer where a terminal can be opened to monitor the output of the development board. Writing to this port can be accomplished with the standard `printf` function, but, in this case to have more control over the resource a custom UART print function, which will be explained in the section 3 of this document, was created.

3 Theory of Operation

In this system the majority of the work is done in software since the only peripherals are the UART connection and a clock. The Nios-II processor is first loaded onto the FPGA along with the memory. The operating system is then loaded onto the processor and is used to create two tasks. The first task prints a lowercase alphabet and the second an uppercase alphabet. Initially the program was run with no mutex and the tasks were set to have a difference between delays of 5ms to check that the print statements did in fact interfere with one another. After that was tested the mutex was added to lock the resources when in use which should allow for the print statement to be executed without interference. The `uartPrint` function is another crucial part of this program not shown in the

^{*}University of Dayton

[†]Dept. of Electrical and Computer Engineering, University of Dayton, 300 College Park, Dayton, OH 45469-0226, U.S.A. E-mail: mcclammaz1@udayton.edu

diagram explicitly, it polls the UART register, checking the transmit status bit if the bit is a 1 then the register is available, if it is a 0 the function continues polling. Once the status bit is read as a 1 the function then writes a byte to the transmit data part of the register and then goes back to polling for the status bit until it is available and can write the next byte. This continues until the string reaches a null character since that is how C terminates strings.

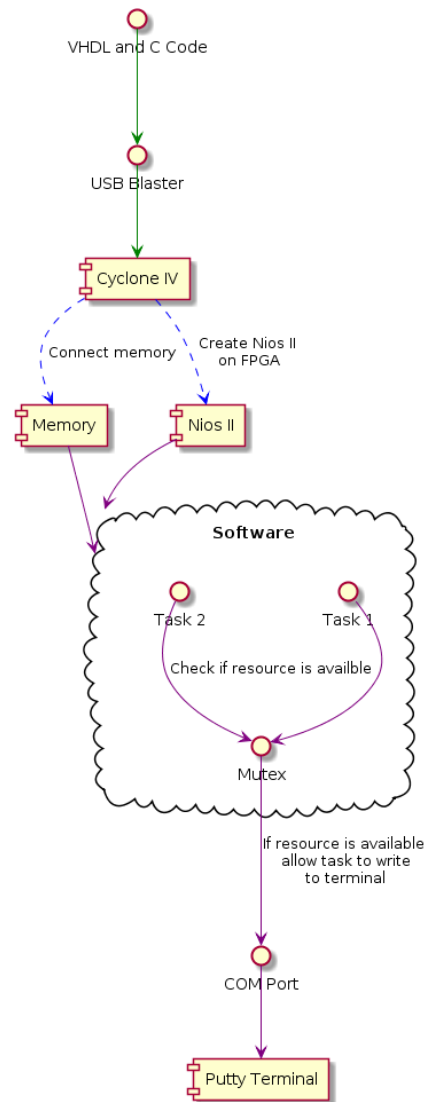


Figure (1) Embedded System Block Diagram

4 Results

In the screenshots below Figure 2 demonstrates how the program functions without a mutex. As can be seen in the screenshot the print statements are clearly

interfering with one another because there are capital letters being printed with lower case letters. Figure 3 shows the results when the mutex is added, and as can clearly be seen the mutex prevents the print statements from interfering with one another.

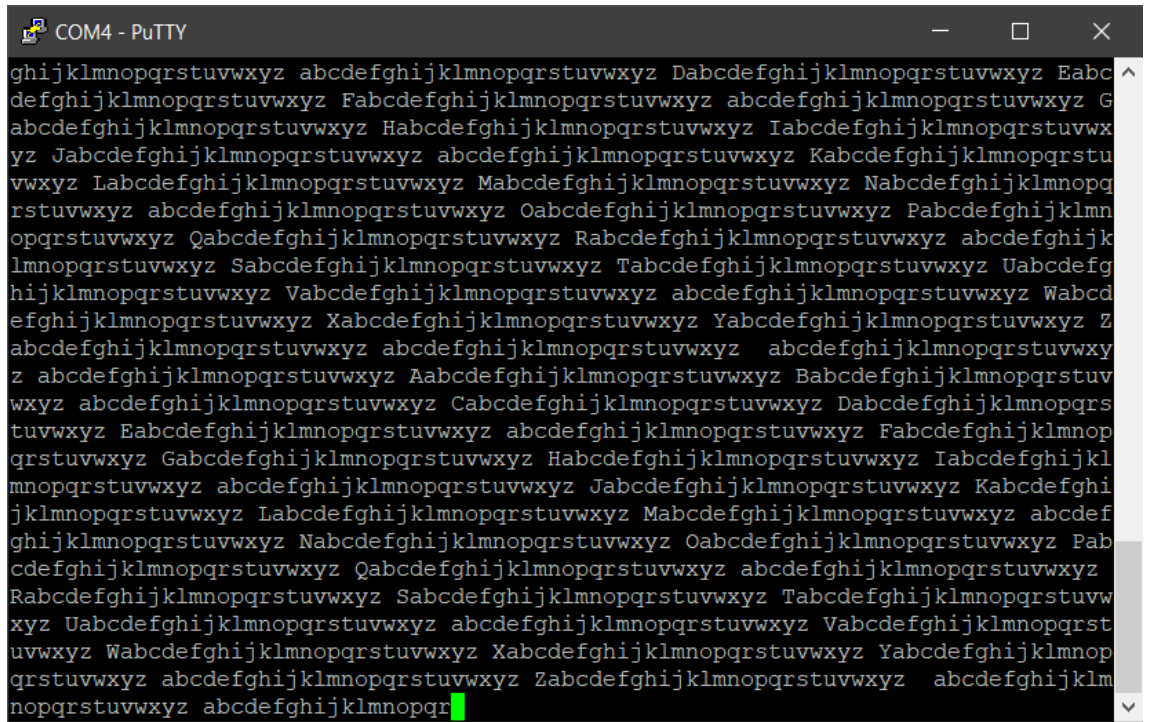
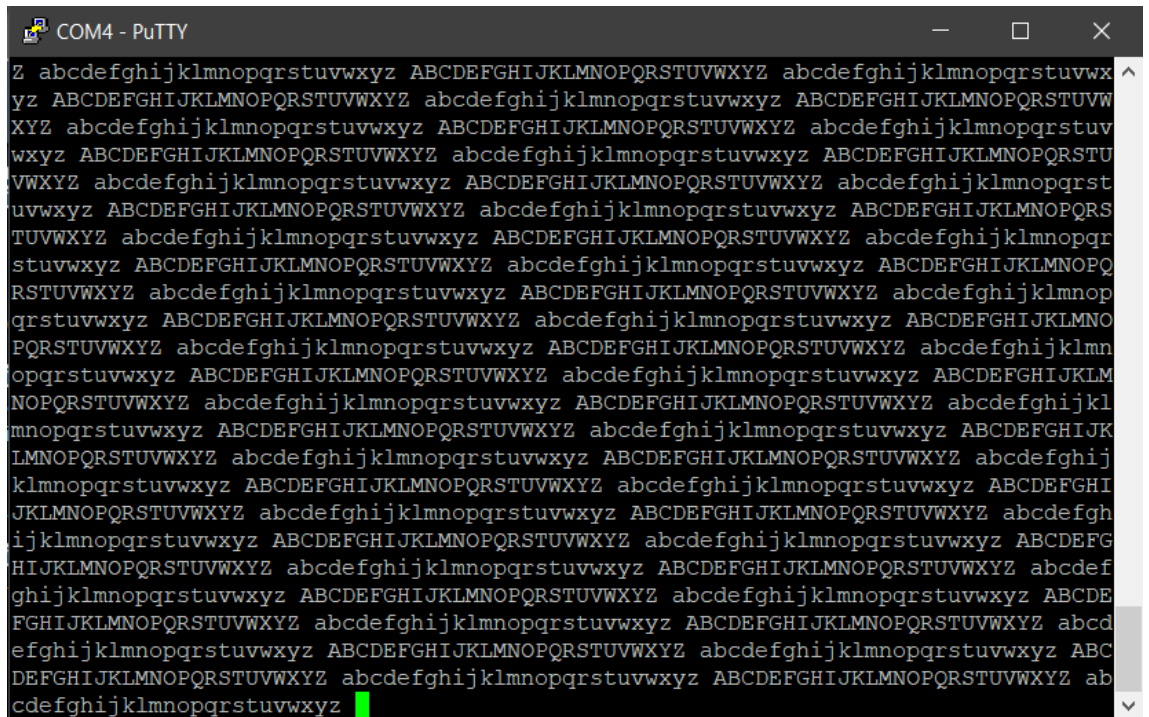


Figure (2) Terminal output without Mutex

A terminal window titled "COM4 - PuTTY" with standard window controls. The terminal displays a single line of text where lowercase and uppercase alphabets are interleaved. The sequence starts with 'Z' followed by 'abcdefghijklmnopqrstuvwxyz', then 'ABCDEFGHIJKLMNOPQRSTUVWXYZ', and continues with 'abcdefghijklmnopqrstuvwxyz' again. This pattern repeats for the entire line, ending with 'abcdefghijklmnopqrstuvwxyz'. A green cursor is visible at the end of the line.

```
Z abcdefghijklmnopqrstuvwxyz ABCDEFGHIJKLMNOPQRSTUVWXYZ abcdefghijklmnopqrstuvw
xyz ABCDEFGHIJKLMNOPQRSTUVWXYZ abcdefghijklmnopqrstuvwxyz ABCDEFGHIJKLMNOPQRSTU
VWXYZ abcdefghijklmnopqrstuvwxyz ABCDEFGHIJKLMNOPQRSTUVWXYZ abcdefghijklmnopqr
stuvwxyz ABCDEFGHIJKLMNOPQRSTUVWXYZ abcdefghijklmnopqrstuvwxyz ABCDEFGHIJKLMO
NOPQRSTUVWXYZ abcdefghijklmnopqrstuvwxyz ABCDEFGHIJKLMNOPQRSTUVWXYZ abcdefghij
klmnopqrstuvwxyz ABCDEFGHIJKLMNOPQRSTUVWXYZ abcdefghijklmnopqrstuvwxyz ABCDEFG
HIJKLMNOPQRSTUVWXYZ abcdefghijklmnopqrstuvwxyz ABCDEFGHIJKLMNOPQRSTUVWXYZ abcd
efghijklmnopqrstuvwxyz ABCDEFGHIJKLMNOPQRSTUVWXYZ abcdefghijklmnopqrstuvwxyz ABC
DEFGHIJKLMNOPQRSTUVWXYZ abcdefghijklmnopqrstuvwxyz ABCDEFGHIJKLMNOPQRSTUVWXYZ ab
cdefghijklmnopqrstuvwxyz
```

Figure (3) Terminal output with Mutex

5 Conclusion

Overall this was a fairly simple assignment, I have already worked with mutexes and semaphores in the past so I was familiar with them. The only difficult part about this project was getting the print function right and the only reason I got hung up on it was because I made a small logic error that I didn't notice until I had already ran the program about 100 times. The problem I had was that I was using the while loop that polls the status register as an if statement instead of a while loop so I was checking if the bit was 1 not 0 which caused my program to reach the polling loop and stall there because nothing was ever written to the transmit register the value was always 1.

A

VHDL Code

```
— ECE532 HW4
— Zackary McClamma
— 28-Oct-2019
```

```
library ieee;
  use ieee.std_logic_1164.all;

entity homework4 is
  port
  (
    i_clk           : in  std_logic := 'X'; — clk
    i_reset_n       : in  std_logic := 'X'; — reset_n
    i_uart_rxd      : in  std_logic := 'X'; — rx
    o_uart_txd      : out std_logic   — tx
  );
end homework4;

architecture sch of homework4 is

  component hw4_cpu is
    port (
      clk_clk           : in  std_logic := 'X'; — clk
      reset_reset_n     : in  std_logic := 'X'; — reset_n
      uart_rxd          : in  std_logic := 'X';
      uart_txd          : out std_logic
    );
  end component hw4_cpu;

begin

  u0 : component hw4_cpu
    port map (
      clk_clk           => i_clk,
      reset_reset_n     => i_reset_n,
      uart_rxd          => i_uart_rxd,
      uart_txd          => o_uart_txd
    );
end sch;
```

B

C Code

B.1 Headers

```
/*
 * Name: Zackary McClamma
 * Course: ECE 532
 * Assignment: Homework 4
 * Date: 28 OCT 2019
 * File: hw4.h
 *
 * */

#ifndef HW4_H_
#define HW4_H_

typedef volatile struct{
    unsigned int uart_rxddata;
    unsigned int uart_txddata;
    unsigned int uart_status;
    unsigned int uart_control;
    unsigned int uart_divisor;
    unsigned int uart_eop;
}uart_reg;

void uartPrint(char* input);

#endif /* HW4_H_ */
```

B.2 Source

```
/*
 * Name: Zackary McClamma
 * Course: ECE 532
 * Assignment: Homework 4
 * Date: 28 OCT 2019
 * File: main.c
 *
 * */

#include <stdio.h>
#include "system.h"
#include "includes.h"
#include "altera_avalon_uart_regs.h"
#include "hw4.h"
```

```

/* Definition of Task Stacks */
#define TASK_STACKSIZE 2048
OS_EVENT *tex;

OS_STK task1_stk[TASK_STACKSIZE];
OS_STK task2_stk[TASK_STACKSIZE];

/* Definition of Task Priorities */

#define TASK1_PRIORITY 2
#define TASK2_PRIORITY 3

/* Prints "Hello World" and sleeps for three seconds */
void task1(void* pdata)
{
    INT8U err;

    while (1)
    {
        OSMutexPend(tex, 0, &err);
        uartPrint("abcdefghijklmnopqrstuvwxyz_\n");
        OSTimeDlyHMSM(0, 0, 0, 5);
        OSMutexPost(tex);
    }
}

/* Prints "Hello World" and sleeps for three seconds */
void task2(void* pdata)
{
    INT8U err;

    while (1)
    {
        OSMutexPend(tex, 0, &err);
        uartPrint("ABCDEFGHILKLMNOPQRSTUVWXYZ_\n");
        OSTimeDlyHMSM(0, 0, 0, 10);
        OSMutexPost(tex);
    }
}

/* The main function creates two task and starts multi-tasking */
int main(void)
{
    INT8U err;
    tex = OSMutexCreate(1, &err);
    OSTaskCreateExt(task1,

```

```

        NULL,
        (void *)&task1_stk[TASK_STACKSIZE-1],
        TASK1_PRIORITY,
        TASK1_PRIORITY,
        task1_stk,
        TASK_STACKSIZE,
        NULL,
        0);

OSTaskCreateExt(task2,
    NULL,
    (void *)&task2_stk[TASK_STACKSIZE-1],
    TASK2_PRIORITY,
    TASK2_PRIORITY,
    task2_stk,
    TASK_STACKSIZE,
    NULL,
    0);

OSStart();
return 0;
}

void uartPrint(char* input){

    uart_reg *reg = UART_0_BASE;
    do{
        while((reg->uart_status & 0x00000040) == 0x00);
        reg->uart_txdata = *input;
        input = input + 1;
    } while (*input != '\0');

    reg->uart_txdata = '\0';
    return;

}

```