Lab 2: UART microserver

martedì 14 novembre 2023 14:34

The aim of this lab is to create a UART microserver, that receives messages from a UART connection, operates some processing on the received data and sends processed data in output back to the same UART port.

Vivado project project_nexys_complete.zip
Block design view design_1.pdf

Analyze the print primitive implementation

Create an application project using the HelloWorld template.
Analyze the print() function, as an example of blocking write to a UART controller.
Follow its implementation and the implementation of the internal primitives down to the level where access to registers is visible (holding Ctrl and clicking on a function name opens its implementation).

Check as a reference the ppt presentation about peripherals (slide 43) Peripherals_SMC.pdf

Implement a blocking read from UART and a blocking write to UART

Implement a blocking read_from_uart() function that reads data from UART as soon as there is valid data in the FIFO. Test it by sending characters to your board through UART with the SDK terminal and printing them back from the Board. Implement similarly a blocking write (it will be very similar to the print() function).

Implement a continuous loopback

Implement a loopback that sends back to the UART every received data.

Implement a simple UART-based shell

Implement a polling loop that listens on the UART and performs different actions on the LEDs depending on the single-byte commands received:

- 'a' - set all the 16 LEDs to 1
- Any number from 0 to 9 – sets to '1' the bits corresponding to '1's in the binary representation of the received number

Implement and use a non-blocking read from the UART

Implement an application to control three groups of 4 LEDs from the LEDs available on the Nexys board using polling of 4 buttons, 4 switches, and UART communication. The program should read input from switches, buttons, and UART and update corresponding groups of LEDs based on the input.

Consider the following code as a template:

```
#include <stdio.h>
#include "platform.h"
#include "xil_printf.h"
#include "xil_io.h"
#include "xuartlite_l.h"

#define UART (u8)2
#define SWITCHES (u8)1
#define BUTTONS (u8)0

u32 my_XUartLite_RecvByte(UINTPTR BaseAddress)
{
// insert your implementation of the non-blocking read here;
}

void update_leds(u32 data, u8 mode){
// insert your implementation of the led update here;
// mode indicates which input you are considering for the update;
}

}

int main()
{
    init_platform();
    u32 uart;
    u32 buttons;
    u32 switches;
    print("Hello World\n\r");

    while(1){
        buttons=Xil_In32(0x40010000);
        update_leds(buttons, BUTTONS);
        switches=Xil_In32(0x40020000);
        update_leds(switches, SWITCHES);
        uart=my_XUartLite_RecvByte(0x40600000);
        update_leds(uart, UART);
    }

    cleanup_platform();
    return 0;
}
```

1. The least significant 4 LEDs should be controlled by the rightmost 4 switches, other switches need to be ignored.
2. The next 4 LEDs should be controlled by 4 buttons, other buttons need to be ignored.
3. The next 4 LEDs should be controlled by the least significant 4 bits in the bytes received from UART.
4. Implement a polling mechanism for reading input from switches, buttons, and UART.
5. Implementing UART Reception using a non-blocking read primitive `my_XUartLite_RecvByte()`
    a. The function needs to indicate when nothing was received
    b. The function needs to discard CR character to avoid considering the one sent by the terminal to close the message
6. Implement a `led_update()` function based that changes the groups of bits based on the input without changing the uninvolved LEDs
7. Verify that the LEDs respond correctly to changes in switch positions, button presses, and UART input.