The Abdus Salam International Centre for Theoretical Physics

# *Advanced Workshop on modern FPGA-based technology for Scientific Computing*

# LwIP tutorial

Fernando Rincón
*fernando.rincon@uclm.es*

# Contents

- The guide not to get lost in the *hello world echo server* example
- The exercise:
  - Create a Vivado project with a GPIO IP core connected to the board leds and switches
  - Modify an UDP echo server to send/receive data to/from a remote client
    - The client will send a number that will be displayed in the leds
    - In return, the echo server will answer with the current combination of the switches
  - Test the server with netcat
  - Modify the server to use TCP
- Interesting reference:
  - https://www.xilinx.com/video/soc/networking-with-lwip-focused-free-rtos.html

# FreeRTOS network application

```c
int main()
{

    xTaskCreate((void (*)(void*))main_thread,"main_thread", THREAD_STACKSIZE,
                NULL, DEFAULT_THREAD_PRIO, &xMainTask);
    vTaskStartScheduler();
    while(1);
    return 0;
}
```

```c
int main_thread()
{

    /* initialize lwIP before calling sys_thread_new */
    lwip_init();

    xTaskCreate(network_thread, "nw_thrd", THREAD_STACKSIZE,
                NULL, DEFAULT_THREAD_PRIO, &xNetTask);

    vTaskDelete(NULL);
    return 0;
}
```

```c
void network_thread(void *p)
{
    struct netif *netif;
    /* the mac address of the board. this should be unique per board */
    unsigned char mac_ethernet_address[] = { 0x00, 0x0a, 0x35, 0x00, 0x01, 0x02 };
    ip_addr_t ipaddr, netmask, gw;

    /* initialize lwIP */
    lwip_init();

    netif = &server_netif;

    xil_printf("\r\n\r\n");
    xil_printf("-----lwIP Socket Mode Echo server Demo Application ------\r\n");

    /* initliaze IP addresses to be used */
    IP4_ADDR(&ipaddr,  192, 168, 1, 10);
    IP4_ADDR(&netmask, 255, 255, 255,  0);
    IP4_ADDR(&gw,      192, 168, 1, 1);

    /* print out IP settings of the board */

    print_ip_settings(&ipaddr, &netmask, &gw);
    /* print all application headers */

    /* Add network interface to the netif_list, and set it as default */
    if (!xemac_add(netif, &ipaddr, &netmask, &gw, mac_ethernet_address, PLATFORM_EMAC_BASEADDR)) {
        xil_printf("Error adding N/W interface\r\n");
        return;
    }

    netif_set_default(netif);

    /* specify that the network if is up */
    netif_set_up(netif);

    /* start packet receive thread - required for lwIP operation */
    sys_thread_new("xemacif_input_thread", (void(*)(void*))xemacif_input_thread, netif,
            THREAD_STACKSIZE,
            DEFAULT_THREAD_PRIO);

    xil_printf("\r\n");
    sys_thread_new("echod", echo_application_thread, 0,
        THREAD_STACKSIZE,
        DEFAULT_THREAD_PRIO);
    vTaskDelete(NULL);

    return;
}
```

Here the network interface and Ips are configured but still two more threads will be created:

- *xemacif_input_thread*
- *echo_application_thread*

They are the only two that will remain

# Deeper look into the *network_thread*

```
void network_thread(void *p)
{
    struct netif *netif;
    /* the mac address of the board. this should be unique per board */
    unsigned char mac_ethernet_address[] = { 0x00, 0x0a, 0x35, 0x00, 0x01, 0x02 };
    ip_addr_t ipaddr, netmask, gw;

    /* initialize lwIP */
    lwip_init();

    netif = &server_netif;

    xil_printf("\r\n\r\n");
    xil_printf("-----lwIP Socket Mode Echo server Demo Application ------\r\n");

    /* initliaze IP addresses to be used */
    IP4_ADDR(&ipaddr,  192, 168, 1, 10);
    IP4_ADDR(&netmask, 255, 255, 255,  0);
    IP4_ADDR(&gw,      192, 168, 1, 1);

    /* print out IP settings of the board */

    print_ip_settings(&ipaddr, &netmask, &gw);
    /* print all application headers */

    /* Add network interface to the netif_list, and set it as default */
    if (!xemac_add(netif, &ipaddr, &netmask, &gw, mac_ethernet_address, PLATFORM_EMAC_BASEADDR)) {
        xil_printf("Error adding N/W interface\r\n");
        return;
    }

    netif_set_default(netif);

    /* specify that the network if is up */
    netif_set_up(netif);

    /* start packet receive thread - required for lwIP operation */
    sys_thread_new("xemacif_input_thread", (void(*)(void*))xemacif_input_thread, netif,
            THREAD_STACKSIZE,
            DEFAULT_THREAD_PRIO);

    xil_printf("\r\n");
    sys_thread_new("echod", echo_application_thread, 0,
        THREAD_STACKSIZE,
        DEFAULT_THREAD_PRIO);
    vTaskDelete(NULL);

    return;
}
```

MAC address

Static IP address

Network interface configuration

LwIP background task

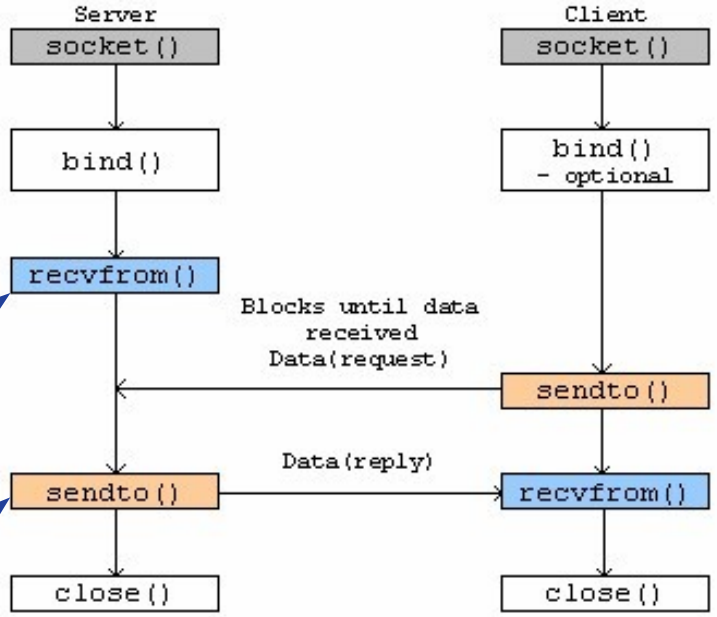
The echo server task

# The UDP echo server

```c
#define THREAD_STACKSIZE 1024

u16_t echo_port = 7;

void echo_application_thread()
{
    int sock;
    struct sockaddr_in address;
    struct sockaddr client_address;
    socklen_t client_length;

    int RECV_BUF_SIZE = 2048;
    char recv_buf[RECV_BUF_SIZE];
    int n, nwrote;

    memset(&address, 0, sizeof(address));

    if ((sock = lwip_socket(AF_INET, SOCK_DGRAM, 0)) < 0)
        return;

    address.sin_family = AF_INET;
    address.sin_port = htons(echo_port);
    address.sin_addr.s_addr = INADDR_ANY;

    xil_printf("binding socket to port %d\n", echo_port);
    if (lwip_bind(sock, (struct sockaddr *)&address, sizeof (address)) < 0)
        return;

    while (1) {
        if ((n = recvfrom(sock, recv_buf, RECV_BUF_SIZE, 0/*flags*/, &client_address, &client_length)) < 0) {
            xil_printf("%s: error reading from socket %d, closing socket\r\n", __FUNCTION__, sock);
            break;
        }

        /* break if client closed connection */
        if (n <= 0)
            continue;

        /* handle request */
        if ((nwrote = sendto(sock, recv_buf, n, 0, &client_address, client_length)) < 0) {
            xil_printf("%s: ERROR responding to client echo request. received = %d, written = %d\r\n",
                       __FUNCTION__, n, nwrote);
            continue;
        }

    }
}
```

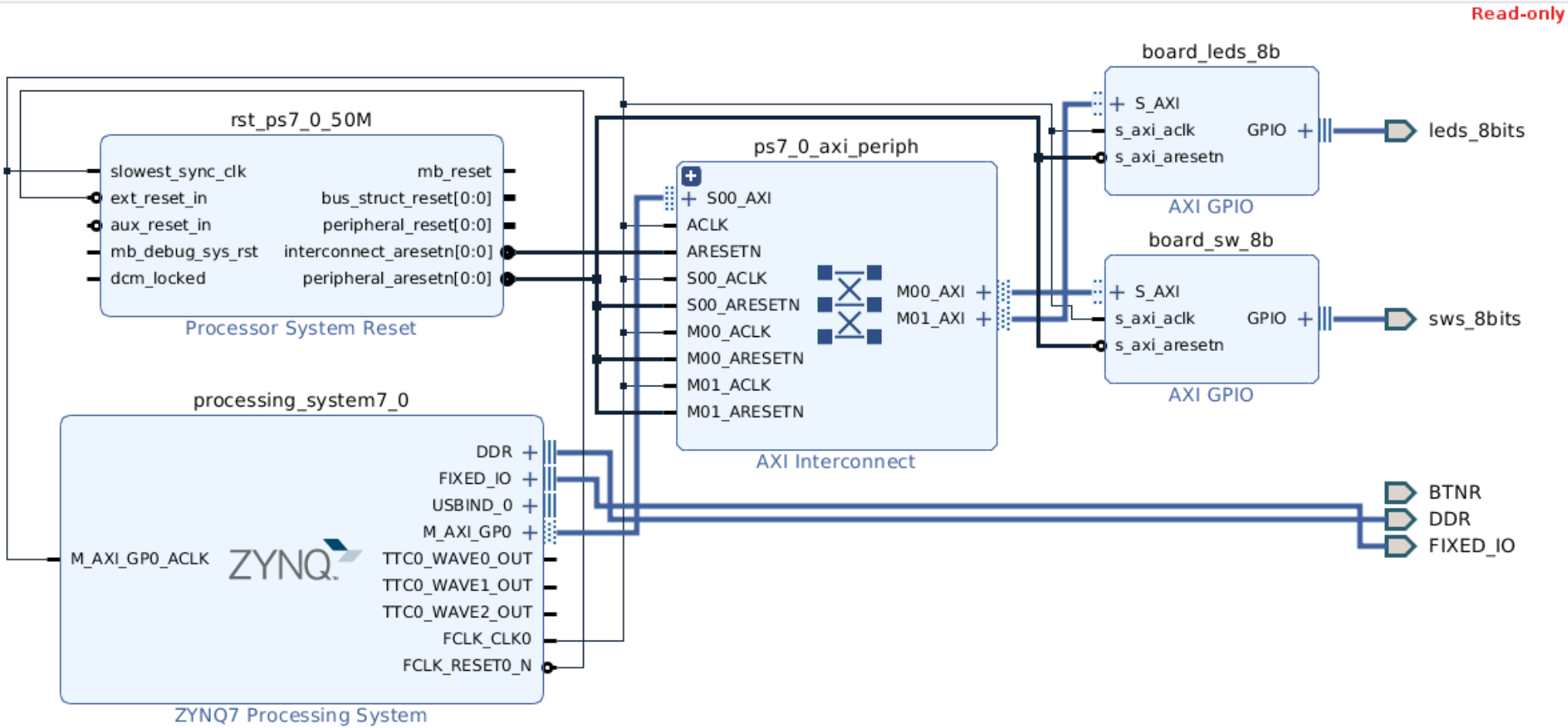
We need to keep track of the client sending data, since no connection is established and we need it for the reply

We send the reply to the client previously saved



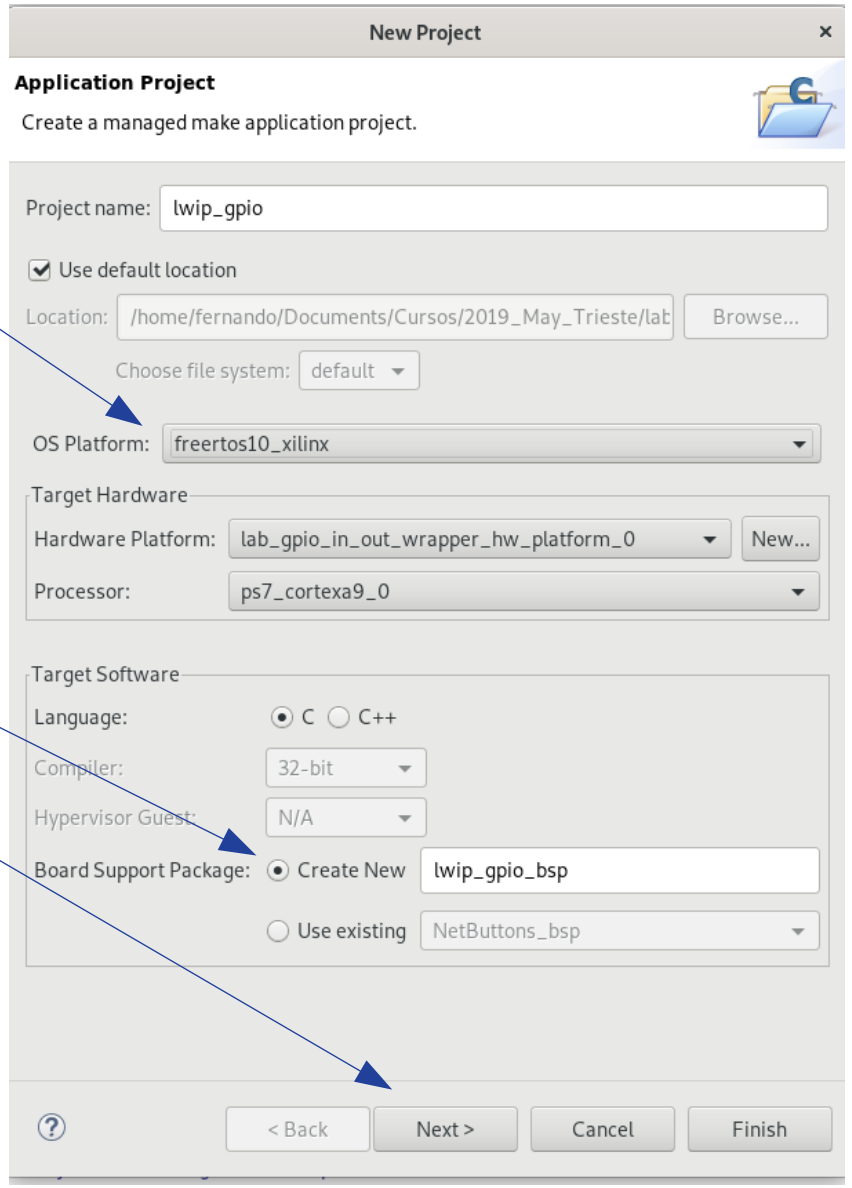
# The exercise: Step 1 – Vivado project

- First create a vivado project with a GPIO core connected to buttons and leds (could also be two different GPIOs)



# Step 2 – UDP echo server

- Export the vivado design to the SDK
- Create a **New→ Application Project**:
  - Select **freertos10_xilinx** as the OS Platform
  - The selection involves the generation of the FreeRTOS bsp
  - Then click on Next
  - In the next dialog box select:
    - **FreeRTOS lwIP echo server**
  - Then click **Finish**
- Now we'll do a little tuning of the generated code



# Step 2 – UDP echo server

- The generated code is using TCP for the echo implementation. We will replace with a simplified UDP version
- To do so, replace the contents of the main.c and echo.c files with the one provided in the shared folder.
- Then connect the board a run the code

# Step 3 – Test the UDP echo server

- Plug the Ethernet cable to the PC
- Open a terminal window in the PC (Windows Key + write “cmd”)
- Cd to the location of the netcat directory
- Run a netcat UDP client
  - the target IP is static: 192.168.1.10
  - The server is listening at port 7

```
nc –u 192.168.1.10 7
```

- If everything is OK the lines you type in the terminal window will be copied back

# Step 4 – server modification

- Now we will modify the server to implement the desired functionality this way:
  - We will use a queue to push the data received by the echo thread
  - Immediately after receiving data from a client, the echo thread will reply to that same client with the value read from the switches GPIO
  - A second task will receive data from the queue and write the value into the leds GPIO
- In the **main.c** at the *network_thread* function

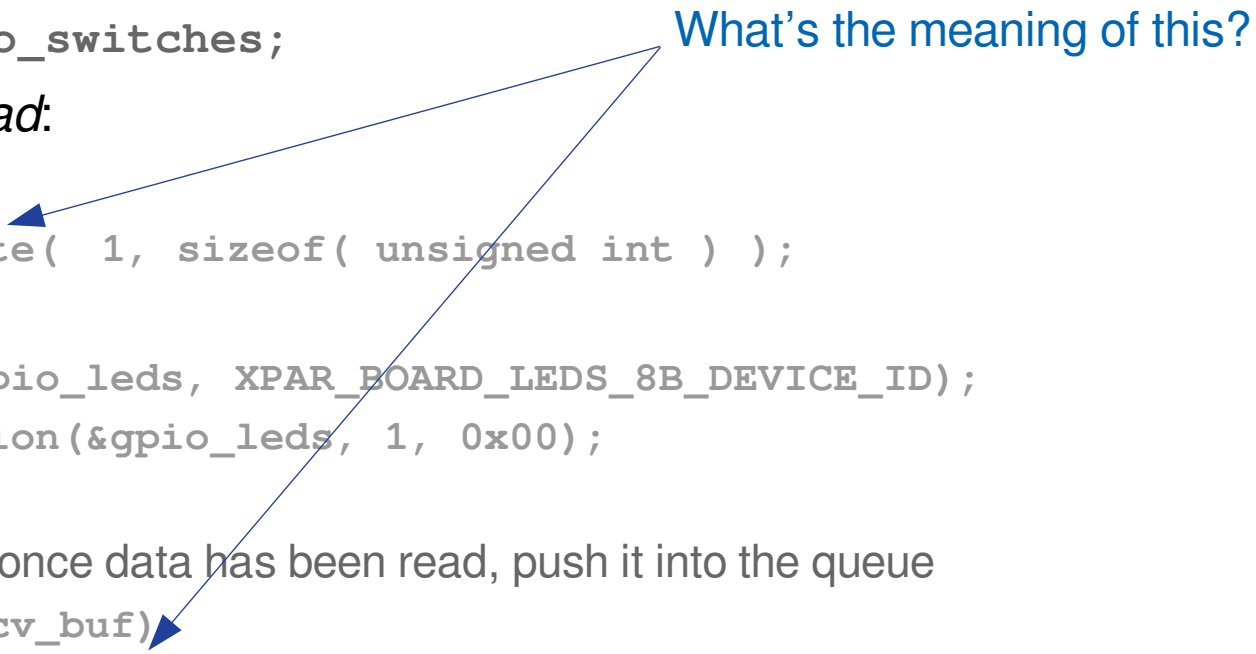
```
xil_printf("\r\n");
sys_thread_new("echod", echo_application_thread, 0,
    THREAD_STACKSIZE,
    DEFAULT_THREAD_PRIO);

vTaskDelete(NULL);
```

Create a new “leds_thread”

Remember to include the interface definition at the top of the file just like the echo application thread

# Step 4 – server modification

- The rest of modifications are performed in the **echo.c** file
- Declare the queue as a global variable:
  - `QueueHandle_t xQueue;`
- And also the two GPIO instances for leds and switches
  - `XGpio gpio_leds, gpio_switches;`
- In the *echo_application_thread*:
  - Create the queue:
    - `xQueue = xQueueCreate( 1, sizeof( unsigned int ) );`
  - Then initialize the GPIOS
    - `XGpio_Initialize(&gpio_leds, XPAR_BOARD_LEDS_8B_DEVICE_ID);`
    - `XGpio_SetDataDirection(&gpio_leds, 1, 0x00);`
    - ...
  - Inside the infinite while, and once data has been read, push it into the queue
    - `int value = atoi(recv_buf)`
    - `xQueueSend(xQueue, &value, 0UL);`
  - Finally read the value from the switches (XGpio_DiscreteRead(&gpio_switches, <channel>)), copy it into a buffer and send it, instead the copy of the received data



# Step 4 – server modification

- Finally create the leds_thread that reads from the queue and copies the value into the GPIO:

```
void leds_thread() {
     unsigned int value;
     for( ;; ) {
          xQueueReceive( xQueue,  &value, portMAX_DELAY );
          XGpio_DiscreteWrite(&gpio_leds, 1, value);
     }
}
```

What's the meaning of this?

# Step 5 – Test the server

- Use netcat again to test the server. The socket is the same one that we used previously

# Step 6 – Replace UDP with TCP

- This will only affect the **echo.c** file
- In the *echo_application_thread*:
  - Replace the parameter SOCK_DGRAM with SOCK_STREAM when creating the socket
  - After the binding put the socket to listen
    - `lwip_listen(sock, 0);`
  - And get the client_length that is required for the accept
    - `client_length = sizeof(client_address);`
  - Finally replace the infinite loop with the following one, but keep the previous code that we'll need in an extra thread:

```
while (1) {
    if ((accepted_sock = lwip_accept(sock, &client_address,
                                     (socklen_t *) &client_length)) > 0) {
        xTaskCreate(process_echo_request, "echos", THREAD_STACKSIZE,
                    (void*)accepted_sock, DEFAULT_THREAD_PRIO, &xEchoTask);
    }
}
```

# Step 6 – Replace UDP with TCP

- Create the new thread that will handle the connections accepted (process_echo_request)

```
void process_echo_request(void *param) {
    int sock = (int) param;

    /* here the while (1) loop previously saved*/
    ……

    /* close connection */
    close(sock);
    vTaskDelete(NULL);
}
```

# Step 7 – Netcat test

- Repeat the test with netcat but remember that now we are using TCP, therefore:
  - The server must be running before the client tries to connect (this is a connection oriented protocol)
  - The command line doesn't include the -u switch:

```
nc 192.168.1.10 7
```

  - Hitting Ctrl+C in the terminal window will close the current connection