Exercises FreeRTOS

Exercise 1:

- Open xsdk and create a new application Project.
- Provide Project name
- In OS platform select FreeRTOS823
- Press Next, select FreeRTOS_Hello_world. Press Finish.
- In project explorer right click on the project name - Build Project
- Open the freeRtos_hello_world file
- Identify the main function
- Identify the taskCreate functions
  - Which task has the highest priority?

- Identify the function xQueueCreate
  - How many elements can be stored in the queue?

Connect the serial port and Run the code

- Right click again on the project name
- Go to Run As – Launch on Hardware (GDB)

Exercises FreeRTOS

Exercise 2:

- Invert the priorities of the task
  - What should happen?

    Run the code and verify

  - Add some printf messages for debugging to know which task run first
- Add a vTaskDelay(x1second) to the RxTask to force it to go to the Blocked State
  - What should happen now?

    Run the code and verify
- Put the same prority to both tasks and run the code.

Exercises FreeRTOS

Exercise 3:

- Modify the number of items to be stored in the queue to five.
- Use the following function in the RX task to see how many messages are waiting in the queue

  - UBaseType_t uxQueueMessagesWaiting(QueueHandle_t xQueue);

1) Define task Tx with higher prority than Rx task
   1) Analize with your partner what should happen in the following cases:
      1) With the taskDelay() call in both tasks
      2) Without the taskDelay()

   2) Run the code and verify your analysis.

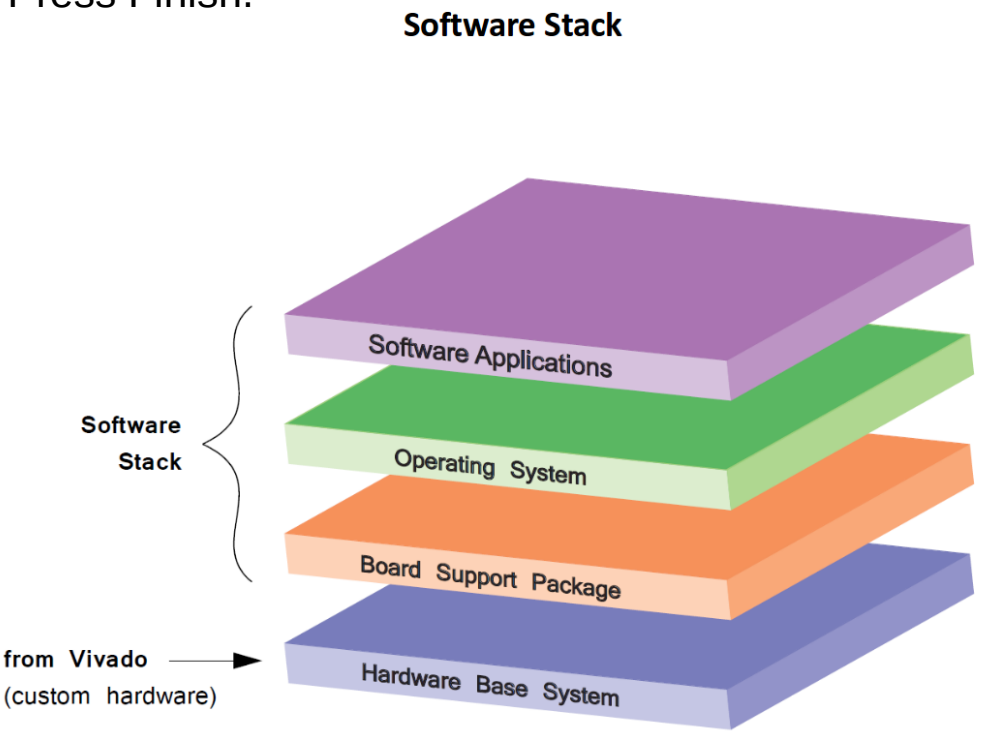
2) Repite part (1) but now with Rx task with higher priority than Tx task

3) Play with the function changing the values of the timer and queue.

Exercises FreeRTOS

Exercise 4:

- Create a project in Vivado with a Zynq processor and a GPIO to the leds
- Generate Bitstream and export hardware including bitstream
- Open xsdk and create a new application Project.
- Provide Project name
- In OS platform select FreeRTOS823
- Press Next, select FreeRTOS_Hello_world. Press Finish.



Exercises FreeRTOS

Exercise 4 (cont):

- Change the main function for

```
int main ( void ){
    // do needed Platform initialization
    // 1) Start LED 1 toogle
    xTaskCreate(Task_LED,  (signed char*) "LEDs", 1024, NULL, 1, NULL);

   //  2) printf
    xTaskCreate (Task_Print, (signed char*) “Print", 1024, NULL, 1, NULL);

   // Finally: Start FreeRTOS
    vTaskStartScheduler();

// Will only reach here if there was insufficient memory to create the idle task
    return 0;
}
```

Exercises FreeRTOS

Exercise 4 (cont):

- Add the corresponding tasks

```
int tick=0;
```

```
void Task_LED (void* p)
{
int tick;
  while (1)
  {
     Xil_Out32 (aGPIO, tick;
     vTaskDelay (100);
  }
}
```

```
void Task_Print (void* p)
{
  while (1)
  {
     printf („Tick is %d \n", tick);
     vTaskDelay (500);
     tick++;

  }
}
```

Default GPIO Base-Address: look in xParameters.h (0x4120_0000)

Configure the FPGA and Run the code

Exercises FreeRTOS

Exercise 5: Modify exercise 4 adding a GPIO to the switches, in such a way that the system shows in the Leds and in the serial port the value of the switches multiplied by 2. Create a new task to read the switches.

Exercise 6: Modify exercise 5 allowing task reading switches create the task to write in leds and destroy the task when finish.

Exercise 7: Modify exercise 6 stopping the scheduler until the event from switches occurs.

Exercise 8: Add a new GPIO for managing the set of 5 buttons of the board, and create two different tasks to attend, one the buttons and the other the switches. Both tasks should be send a message to the serial port indicating wich button and which switch was activated. Use a set of queue for managing the events.

Exercise 9: Add to the exercise 8 a task idle which perform a counting of ticks and show the value of the counting with the message.