ECE 315 Lab 1 Prelab – whitfor1

1. Imagine two different situations where the 7-segment LED display isn't functioning correctly:
    a. The digits are displayed on the incorrect side of the display.
    b. The digits are constantly flickering instead of being perceived as steadily lit.

   Identify potential causes for these issues. What might be going wrong in each scenario? Assume that the hardware platform is correctly configured.

   Answer:

   We will first establish as the question states that the hardware platform is correctly configured. Some common issues on why a or b may be experienced could be due to faulty hardware (even if the platform itself is correctly configured). Perhaps pin(s) are shorted or are improperly seated in between the JC & JD headers that could cause loss of power or data. We could also consider that perhaps the GIC, Central Interconnect that establish the bus and 'talking' between the APU (Zynq 7000) and the peripheral devices have malfunction or failed. These are all strictly hardware issues that *could happen* (though unlikely) and would be difficult to debug. Now towards the software side, we could have issues with the initial C template for Part 1 to initially build; this may inadvertently soft lock the board such as improper initialization of the `SSD` and setting the `GPIO direction` in the opposite way. If improper scheduling had occurred, there may be a starvation of cpu resources if any of the peripherals are waiting for their turn to communicate with the processor. Into the specifics of parts a & b:

   For a, we know that the digits on the SSD are interfaced specifically through the bit masking occurring at the hardware platform level to choose the appropriate 'segments' on each digit to power on, as such example, `[7:0] = x000 0110 where x=CAT bit`, which would display 1 on any of the digits, x choosing whether it is left or right digit. If we do <u>not</u> update the status of the `previous_key`, `current_key`, and `new_key` appropriately, that is, as a new key is pressed its digit should go to the right display, while the old digit goes to the left side of the display does <u>not happen</u>, then we have a logic error within the code that must be addressed. If the status function is also improperly implemented, it could soft lock the cpu as resources are tied to print the current status (though this would be *very unlikely* to happen).

   For b, this primarily deals with resource sharing and logic timings of the devices. We know that the SSD can only display one digit at a time on the single bus that communicates with the PMOD configured devices. To appear as both digits being on simultaneously, we will need to create pauses and resumes in the execution of tasks that are currently running, just started, or nearing their end. We are required to code a task named `xDelay`; this controls the frequency within which the LED digits will toggle on and off – essentially their switching frequency. As this is a class about working with embedded systems, we know that SWaP is essential. Power is the one we are most primarily concerned with, because a small digit synthesizer like we are designing in the lab, should have no perceivable flickering; setting the delay too low will eat into the power consumption of the system and congest the future queues and other process tasks; having the delay too high will create major dead time for the CPU where it will be doing nothing, and will only have one digit on at a time, as if covering the other one. Another function that could cause issues is when we decode the 7 segment value; if the base address for the GPIO or the call is incorrect from the keypad press, it may cause issues with `XGpio_DiscreteWrite` function thus improperly showing the correct digits and side. This ties in with `vTaskDelay` that also works with `xDelay`; should the value be too low, our resources will be over consumed and result in little headroom, too much delay and we will not eliminate the 'flickering.'

2. What are the steps required to establish an interface with a peripheral device on the Zybo Z7 board? This should be described assuming that the hardware platform is already in place and ready for configuration.

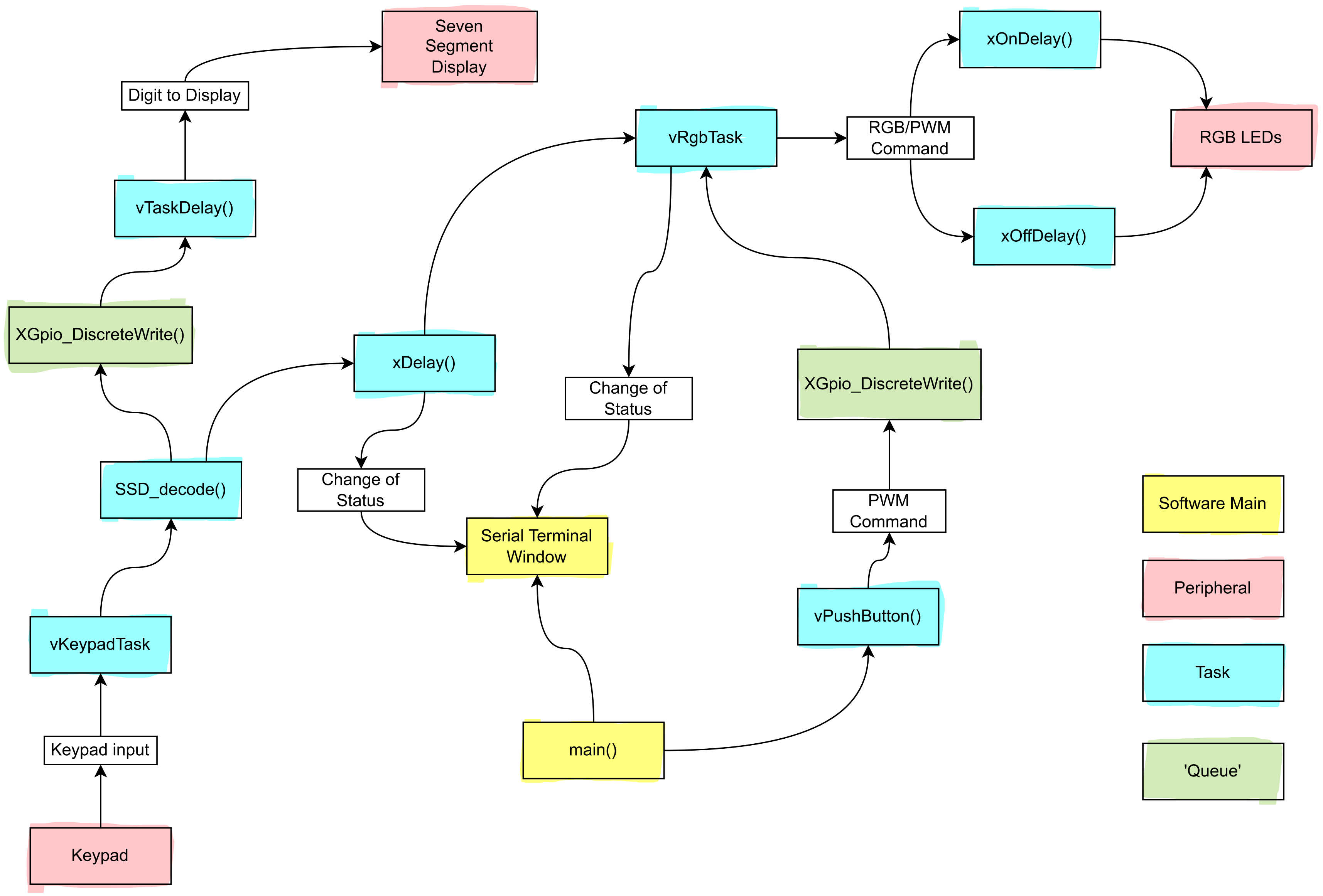
   Answer:

   We are told that the hardware platform is in place and ready for configuration; that is that the virtual circuit, bitstream, and RTOS has properly loaded onto the Zybo. We would also make sure that each peripheral is connected to the appropriate ports (channels) that we will be communicating with. By reviewing `lab1_part1.c` source code file, we can see that the bare minimum to begin communicating with the peripherals of the devices are to include some of the Xilinx Libraries, such as `xgpio.h`, `xparameters.h`, `xscugic.h`, and more. These libraries essentially acts as drivers on the `Hardware Abstraction Layer`, that allow our C program, through the use of `FreeRTOS`, to establish communication lines with the peripherals, and ensure that the data being received and sent between the CPU and devices can be read and executed or analyzed with.

   Once the device drivers (or libraries in our case are imported), we need to define the device IDs, this is essentially the specialized registers that hold the addresses that are used by the Zybos Hardware Platform to determine the channels and location that the software must access to begin communicating with the peripheral devices. With the base address now confirmed, we can initialize an instance of the peripheral (basically saying: "This is my display called 4K OLED and it is this TV."); thus confirming a 'hand shake' between the Zybo and the peripheral.

   At this point, the bare interface between the Zybo Z7 and the peripheral device has been essentially created, though how one would use the peripheral can be further implemented through creating various tasks for the device to perform, and starting the program itself through main.

3. Create a system architecture diagram for Part 2, including the Zybo Z7 board peripherals, the serial terminal window and the FreeRTOS tasks and others. Be sure to use arrows to show the relationships between components in your diagram.


Seven Segment Display
Digit to Display
vTaskDelay()
XGpio_DiscreteWrite()
SSD_decode()
vKeypadTask
Keypad input
Keypad
xDelay()
Change of Status
vRgbTask
RGB/PWM Command
xOnDelay()
RGB LEDs
xOffDelay()
Change of Status
XGpio_DiscreteWrite()
PWM Command
vPushButton()
Serial Terminal Window
main()
Software Main
Peripheral
Task
'Queue'