**Click three different green boxes and in your writeup, describe what configuration options are available and how they may be potentially useful in an embedded system.**

I/O Peripherals – Turn on and off the list of I/O devices depending on which one you would want to use. If the piece of hardware uses a certain I/O peripheral it would be useful to configure it and don’t use things you don’t need.

General Settings – Gives you a general description of what each setting does and gives you the option to change most of them based on what the user wants. If you need a certain setting for the embedded system, you can configure it and learn how it would works

DDR2/3, LPDDR2 – Controls a lot of aspects from the board’s memory. Such as the DRAM Bus Width, the memory type and memory part. Useful in an embedded application where you need to specify the memory needed for the system and how it functions

**Are these buttons, LEDs, and switches connected via the PS subsystem or the PL subsystem? Briefly defend your answer. Note also that all three peripherals appear to be the same exact IP type (axi\_gpio) – how can this be possible?**

PL Subsystem because the peripherals are connected directed to the AXI controller and not a CPU device. All three peripherals got the characteristic of being on or off which can lead them to have the same IP type

**Based on the datasheet and the address map shown in the “Address Editor” (mentioned in instruction 7 of Step 2: Use Designer Assistance), how would you (in software) read the current state of the switches? Be specific**

If the Master Address has the last 4 Bytes to be all 0 then it would be off and if the last 4 Bytes are all high then it would be on. You would set the GPIO registers to what you want to enable on and off and the corresponding one will interface with the GPIOs to do what is enabled.

**In your writeup, use this feature and describe what print() does, and how. Why do you believe this function is used by Xilinx for their Hello World application, as opposed to the more conventional printf() function?**

It works by going through a while loop that goes through the pointer and every bit sends it UART until every character gets printed out. Utilizes the hardware better than a traditional printf statement in C. printf doesn’t have to worry about the base address while print does.

