Introduction:

In this lab, you will be using the Fibonacci calculator from an earlier lab as a custom peripheral to the ARM processor on the ZedBoard. To enable communication with this peripheral, you will be implementing a custom memory map that associates different FPGA resources with a specific address. Although the AXI interface could provide this same functionality, in this lab we use a specialized abstract interface (i.e., a virtual board) with a memory-map interface that is built on top of the AXI interface. This virtual board enables us to potentially use the exact same code on different boards, instead of porting the memory-map code for different board interfaces.

You will again be working in groups of 2 on this project. Unless you are an EDGE student or have special permission from the instructor, you must work in a group of 2. Note that those in the overflow section are *not* EDGE students, and must work in a group of 2. Please only submit once per group.

EDGE INSTRUCTIONS (All non-EDGE students must do the entire lab):

EDGE students must complete part 1 and a simplified version of part 2. For part 2, you can email me for a C++ template that is close to being complete. You can then complete the code for full credit. Alternatively, you can do all of part 2 yourself for extra credit. I would recommend trying part 2 first and then emailing me for the template if you get stuck. After I send you the template there is no longer an option for extra credit.

Part 1 – VHDL

1) Download the provided code from the lab website. This code contains a solution to Fibonacci calculator in case you didn't get that lab fully working. However, I would encourage you to use your own code after testing everything with the provided code.
2) For this lab, I have provided an existing IP core that you will extend. This core is located in the accelerator_1.0/ directory. To instantiate this core in your Vivado project, move the folder into your IP repository that you used in lab2, or alternatively create a new repository. To use multiple repositories, right click in the block diagram (over several other places), and select "IP Settings." This will allow you to set the path to multiple repositories. As long as you put the accelerator_1.0/ folder into one of these paths, you should be able to instantiate it with the "Add IP" option.
3) Look over the provided code in the accelerator_1.0/src directory.
    - The top level is an entity called wrapper. This entity doesn't do much yet, but will be expanded upon in later labs to provide access to various resources (e.g., external memories). This is the entity that you would instantiate from within the auto-generated AXI peripheral code. However, I have already provided that code for you in the accelerator_1.0/hdl/ directory, which you do not need to change.
    - The wrapper entity instantiates the user_app entity, which is the abstraction you will be using for all future labs. Currently, user_app provides a clock, a reset, and a simplified memory-map interface that is

easier to work with (and can be made portable across different boards) compared to the AXI interface.

- config_pkg.vhd contains constants and types that are used for configuring the virtual board interface.
- user_app.vhd provides constants that are used throughout the application. Think of this file as a custom header file in C code. You should use the constants that I have already provided when creating your memory-map implementation. For future labs, you should use this file for your own custom comments and types.
- user_app_tb.vhd is a provided testbench that demonstrates how to send inputs and read outputs to/from the memory map. You do not have to change the testbench for this lab. However, I would recommend using this testbench as a template for all future labs. Note that the testbench outputs a score, which gives you an idea of your grade. The graders will use a similar testbench, so I can't promise your grade will be identical, but this at least gives a good idea of what to expect.
- memory_map.vhd implements the memory-map entity that enables application-specific communication with FPGA resources
- The fib/ directory contains the solution for the Fibonacci calculator. The top-level entity is called fib.

4) Modify the memory map entity to enable communication with the two application inputs: *go* and *n*. Do the same for the two outputs: *done* and *result*. This will be explained in more detail in class. I will also try to provide an exact timing diagram if I get a chance.

   The memory-map interface works as follows. When *wr_en* is asserted (active high), this means that the processor is currently providing valid data on *wr_data*. This data will only be valid for one cycle, so in the same cycle you must check *wr_addr* and then store *wr_data* into the corresponding resource, which for this assignment is just a register. For example, you will have separate registers for inputs *go* and *n*.

   When *rd_en* is asserted (active high), this means that the processor is requesting data from the address on *rd_addr*. The interface requires the data to be provided on the *rd_data* port **one cycle after** *rd_en* is asserted. If you provide the data at any other time, your code will not work.

5) Modify the user_app entity to instantiate your memory_map entity and the fib entity (either the provided one or your own). Connect them together appropriately.

6) Simulate user_app_tb to test the correctness of your design. Note that even if the testbench reports no errors, there could still be bugs in your code. In fact, I purposely created the testbench in such a way that it is unlikely to expose certain types of bugs. I leave it to you as an exercise to make the testbench better, but you do not have to submit the modified testbench.

7) Repackage the accelerator IP with your modified files, add it as a peripheral for the Zynq, and generate the corresponding bitfile (see lab 2 for the exact steps).
8) Rename the bitfile to lab3.bit and upload it to your project directory on reconfig.ece.ufl.edu.

Part 2 – C++

In part 2, you will be creating a software program to communicate with the peripheral. Use lab 2 as a reference for the board API. Make sure to define constants for the memory-map addresses that match those in user_pkg.vhd.

The C++ code should calculate the first 30 Fibonacci numbers using the FPGA circuit and output them to the screen. To verify correctness, you should also calculate each Fibonacci number using software using the same algorithm that was specified in lab 1. The output of the program should look **exactly** like this:

```
1: HW = 1, SW = 1
2: HW = 1, SW = 1
3: HW = 2, SW = 2
4: HW = 3, SW = 3
5: HW = 5, SW = 5
6: HW = 8, SW = 8
7: HW = 13, SW = 13
8: HW = 21, SW = 21
9: HW = 34, SW = 34
10: HW = 55, SW = 55
11: HW = 89, SW = 89
12: HW = 144, SW = 144
13: HW = 233, SW = 233
14: HW = 377, SW = 377
15: HW = 610, SW = 610
16: HW = 987, SW = 987
17: HW = 1597, SW = 1597
18: HW = 2584, SW = 2584
19: HW = 4181, SW = 4181
20: HW = 6765, SW = 6765
21: HW = 10946, SW = 10946
22: HW = 17711, SW = 17711
23: HW = 28657, SW = 28657
24: HW = 46368, SW = 46368
25: HW = 75025, SW = 75025
26: HW = 121393, SW = 121393
27: HW = 196418, SW = 196418
28: HW = 317811, SW = 317811
29: HW = 514229, SW = 514229
30: HW = 832040, SW = 832040
```

IMPORTANT: Make sure to test your output with the provided grade.pl script. Copy grade.pl to your project directory on reconfig and then do the following when you run your application:

chmod +x grade.pl (you only have to do this once)

zed_schedule.py ./zed_app lab3.bit > test.txt

./grade.pl test.txt

If you get errors on this last command, try running "dos2unix grade.pl" first.

Note that the reported score only applies to part 2, so a perfect score is 50%. Again, the grader will use a similar script to grade the projects, so this will give you an idea of your grade.

SUBMISSION INSTRUCTIONS (One submission per group)

Make sure the names of all group members are at the top of every file.

Create a directory with the name of your server account. Use the following structure:

```
server_account_name/
        readme.txt              // Group members, anything that the grader needs to
                                // be aware of
        lab3.bit
        accelerator_1.0/        // IP core from repository with this exact name
                                // Make sure all VHDL is included
        sw/
                All C++ code    // All .cpp, .h files, including the Board files from lab 2
                Makefile        // Your own makefile, extend the one from lab 2
```

Zip the entire directory and submit the zip file.

EXTRA CREDIT OPPORTUNITIES

Instead of having the software perform two writes for go=1 and go=0, there is a trick you can do to create assert and de-assert go with only one write from software. If you can figure this out, you will receive extra credit. Make sure to specify in your readme.txt file if you do this.