PROJECT 2: RTL Design

Material covered:

- Finite State Machines
- RTL Design
- Custom Processor Architecture
- System Design
- VHDL

Partners:

If desired, you may work with a partner on the project. However, it is imperative that you work collaboratively, and each understand the whole process. During check-ins TAs and instructors will expect both partners to be able to discuss the implementation.

Project submission:

1) VHDL Demonstration (Check-Offs): Due by Tuesday December 09 at 5pm.
2) Project Report: Due by Friday December 12 at 9pm.

Project Evaluation:

- VHDL Verification: 40 pts
  - Task 2 – 15 pts
  - Task 3 – 15 pts
  - Task 4 – 10 pts
- Project Report: 60 pts
  - Part 1 – 5 pts
  - Part 2 – 10 pts
  - Part 3 – 20 pts
  - Part 4 – 15 pts
  - Part 5 – 5 pts
  - Technical Communication – 5 pts

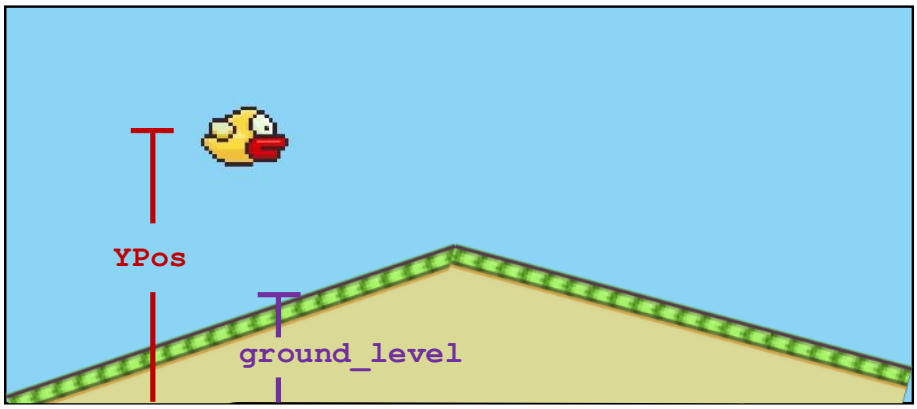
Equipment:

1) Vivado

When you are asked to verify your results with a TA/Instructor, you must show working results, not screen shots. Include screenshots only to document your results in your report.

Overview

For this project we will be designing a custom processor which will implement the beginnings of a game similar to the single button game Flappy bird. (link to example game). This type of game would be relatively simple to implement using software, but we will be building it using logic gates!



Ideally a game like this would be able to handle many mechanics including a variety of obstacles, keeping track of the score, etc.

For our project we will be focusing on the following capabilities:

- Make a character have a positive upward when a button is pushed,
- Implementing basic free fall (accelerating downward)
- Having character die when they hit the ground.

NOTE: Make sure to work carefully and neatly as you complete this design process. You will be evaluated on how well you are able to document your process in your project report.

Task 0: Preparation

Before beginning design and implementation, please create a new. empty Vivado project. Don't forget to save this project in a location outside of OneDrive or other cloud backup services.

I have provided you with a large collection of building blocks, module stubs, and testbenches to help you complete you implementation. Please make sure you have downloaded and imported all of the following files (which have been provided on Brightspace):

Building Block Modules (should not need to change):

- `dvlipflop.vhd`
- `lessthan_6bit.vhd`

Module Stubs (for you to complete):

- `flappy_datapath.vhd`
- `flappy_nextstate_logic.vhd`
- `flappy_output_logic.vhd`
- `flappy_controller.vhd`
- `flappy_processor.vhd`

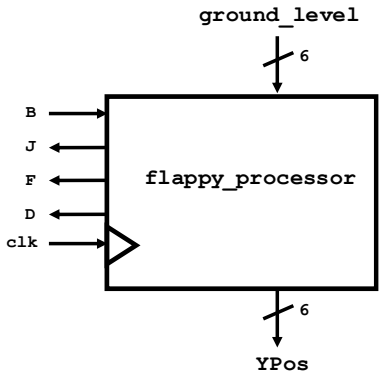
Testbench modules (to evaluate your designs):

- `flappy_datapath_tb.vhd`
- `flappy_nextstate_logic_tb.vhd`
- `flappy_output_logic_tb.vhd`
- `flappy_controller_tb.vhd`
- `flappy_processor_tb.vhd`

NOTE: You will likely need to import other modules to complete your designs (mostly modules from previous laboratories).

Task 1: High Level State Diagram

Your job is to design a custom processor will determine the position of the character in the vertical direction and what action it is taking (jumping, falling, or being dead).



The processor should have the inputs and outputs shown above and described below

Inputs:

- `clk` – system clock
- `B` – single bit input that is HI when the user pushes a button to jump
- `ground_level` – 6-bit input that specifies the current height of the ground in the Y direction (up and down)

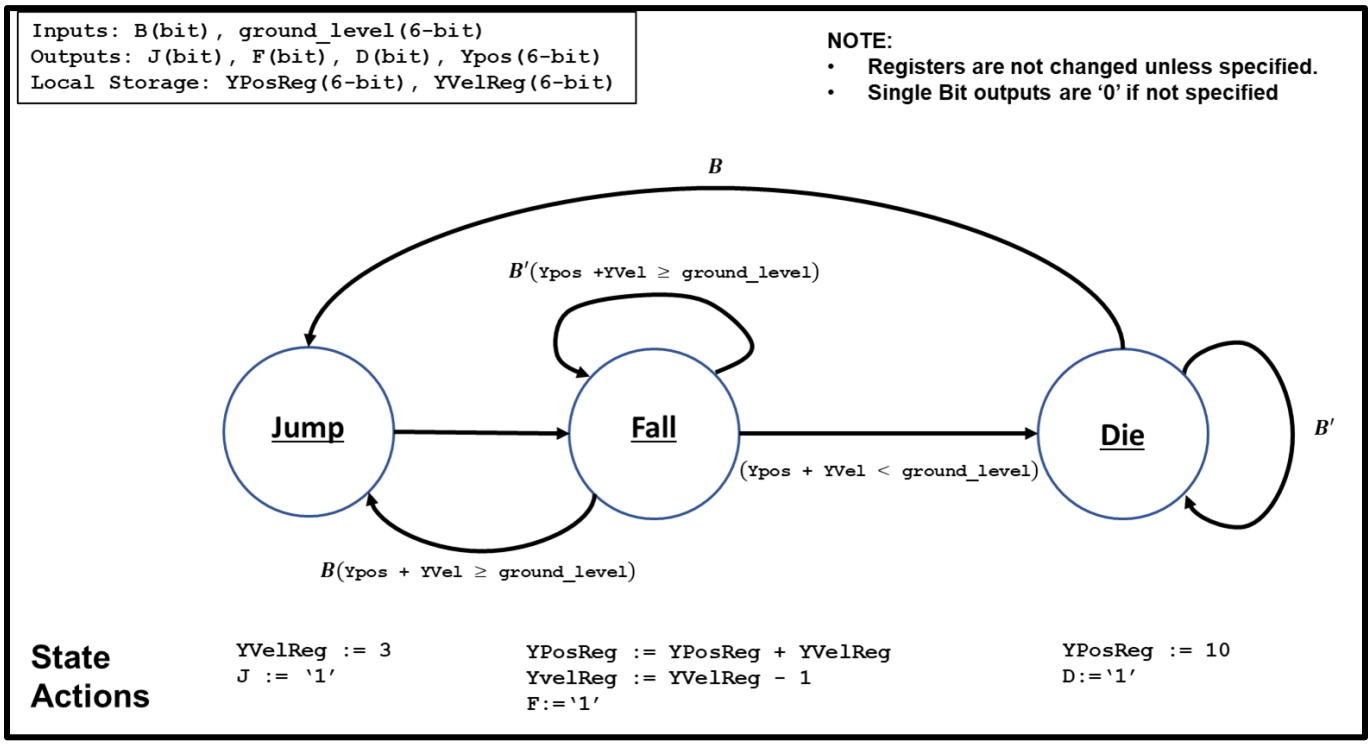
Outputs:

- `J` – single bit output which is HI when the character is executing a jump
- `F` – single bit output which is HI when the character is in free fall
- `D` – single bit output which is HI when the character is dead (game over)
- `YPos` – 6-bit output which outputs the current position of the character in the Y direction (up and down).

Local Storage:

- `YPosReg` – 6-bit register to store the current Y position
- `YVelReg` – 6-bit register to store the current Y velocity

The system should implement the following behavior outlined by the high level state machine provided to you below:

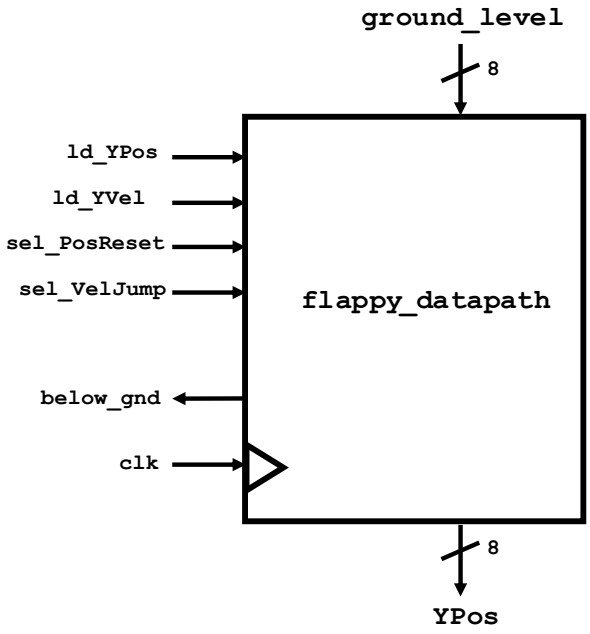


1. Describe the behavior of the custom processor **in words**. You can do this in a bulleted list or in sentences. It should be clear that you understand all of the functionality we want our lemming to have. This will be included as part of your functional description in your report.

I strongly recommend review your description with TA/SA or instructor before continuing to ensure you don't have any misunderstandings.

Task 2: Datapath

Now we need to design and test a datapath that can implement the mathematical and storage capability necessary for our processor.



1. Based on the HLSM, come up with a schematic for the processor datapath using the input / output names shown above.
2. First include registers for any local storage you might need.
3. Then add multi-bit blocks for any computations or comparisons needed and connect them appropriately.

Now prepare to implement the schematic of your datapath module in Vivado (not the whole processor yet). Start by thinking through which modules you will need. You may need to create or import some additional modules to implement your schematic.

Additionally, you will need a less-than component that returns one only when one 6-bit input is less than another. This has been provided for you on Brightspace.

Before beginning to implement anything: make sure to have created a final schematic drawing for the datapath.

Make sure your schematic has the following labeled with an identifier name:

- Each input and output
- Each component
- Each wire that is not an input or an output

I strongly recommend having a TA/SA or instructor review your schematic before proceeding.

Now implement your datapath using a **structural** architecture in VHDL by directly implementing the components and connections specified on your schematics.

When creating the entity for `flappy_datapath`, use the following entity declaration exactly (you can copy it from the lemming_datapath.vhd shell file found on Brightspace).

```
entity flappy_datapath is
  Port (clk:  in std_logic;
        ld_YPos: in std_logic; -- Updates YPosReg when 1
        ld_YVel: in std_logic; -- Updates YVelReg when 1
        sel_PosReset: in std_logic; -- Select bit: 1 means reset to 10
        sel_VelJump: in std_logic; -- Select bit: 1 means jump: Vel=3
        below_gnd: out std_logic; -- Character is below ground (YPos < ground_level)
        ground_level: in std_logic_vector (5 downto 0);
        YPos: inout std_logic_vector (5 downto 0);
        testport: out std_logic_vector (5 downto 0));
end flappy_datapath;
```

NOTE: the module contains an additional output called `testport`. This will allow us access to internal nets for debugging if needed. Speak to an instructor or TA if you have any questions about this.

In order to validate your module, test it with the testbench file found on Brightspace: `flappy_datapath_tb.vhd`.

Please be aware that this component will NOT test the full capability of the game, only that the actions are executed when the proper control lines are selected.

If your simulation does not work correctly the first time, compare it carefully to your schematic. If needed it might be necessary to pass internal nets out to your testbench to debug various points of your circuit. A TA / SA can help with this if you think you need it.

Look over your inputs and outputs closely to convince yourself that that the datapath is performing its job correctly.

TA Verification (Task 2): Demonstrate successful implementation of your lemming controller to a TA / Instructor. Explain why the testbench demonstrates successful implementation of our desired behavior.

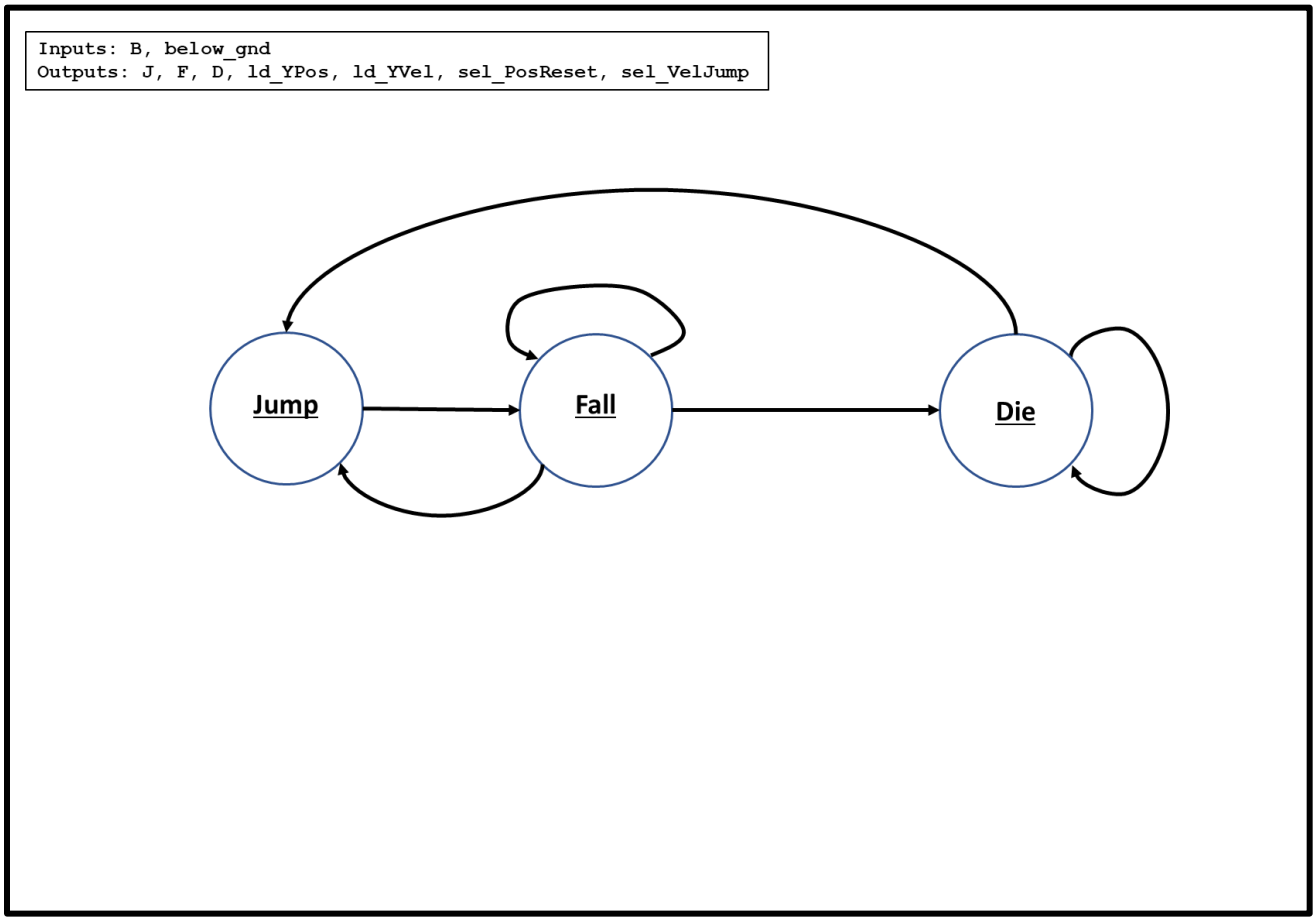
Make sure to take screenshots of your simulated output and be prepared to discuss the results in your report.

Task 3: Controller

Now we need to update our HLSM into a finite state machine (FSM) so that we can implement the controller using flip flops and logic gates.

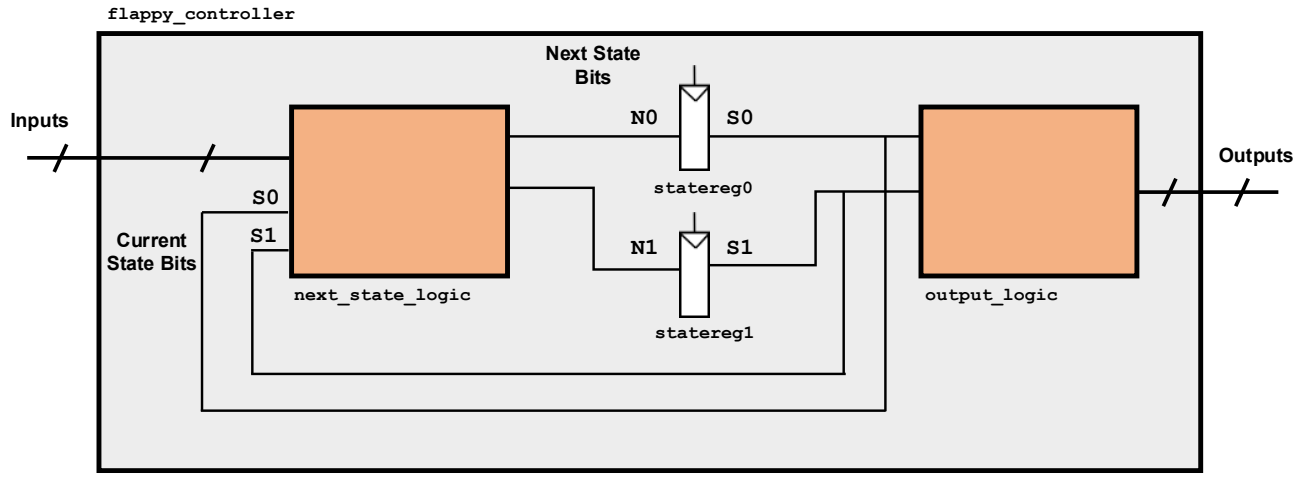
1. Create an updated diagram which describes the behavior of your controller FSM. Remember, any control inputs to your datapath are control outputs from your controller.
    a. This should have the same structure as your HLSM (same states and transitions) A blank version of the state diagram is listed below.
    b. Determine the value of all single bit outputs for each state.
    c. Update any multi-bit conditions to be single bit conditions based on the control outputs of your datapath.

Incomplete FSM diagram (You should complete it!!



We can now begin implementing the FSM using the techniques from earlier in the course. Remember an FSM is implemented using the controller architecture, which consists of a state register, some next state combinational logic, and some output combinational logic.

In order to make implementation and debugging more straightforward, you will be provided with a controller implementation as shown here:



The module `flappy_controller` has been **provided** to you on Brightspace and you should not need to change it. However, the `next_state_logic` and `output_logic` modules are **empty** and must be designed and implemented by you.

2. Based on your updated FSM, create a next state table mapping current state and inputs to next state. I recommend using the state encoding shown below:

State	S_1	S_0
Die	0	0
Jump	0	1
XXXX	1	0
Fall	1	1

3. Create an output table mapping current state to outputs.

4. Based on your two tables, write the minimized Boolean expression which will compute your next state and outputs.

5. Draw two separate schematics based on your Boolean expressions:
   a. `next_state_logic`
   b. `output_logic`

Now prepare to implement your combinational schematics in Vivado. Start by thinking through which modules you will need. You may need to create some additional modules to implement your schematic (NOT gate, multiple input AND / OR gates, etc.).

Before beginning to implement anything: make sure to have created a final schematic drawing for both sub-modules of your controller.

Make sure your schematics have the following labeled with an identifier name:
- Each input and output
- Each component
- Each wire that is not an input or an output

I strongly recommend having a TA/SA or instructor review your schematics before proceeding.

6. Now implement your schematics using a structural architectures in VHDL by directly implementing the components and connections specified on your schematics.

Shell files for both `next_state_logic` and `output_logic` have been provided for you on Brightspace. Make sure you do NOT modify the entity statements for these.

7. Additionally, testbench files for EACH of these modules should allow you to confirm that you are getting the desired outputs for all possible combinations of inputs. These testbench files have been **provided** for you on Brightspace

8. Carefully review the results of your testbench simulation and confirm that your responses are as desired.

9. Once you are happy with your combinational logic results, you should be able to validate the **full** controller module with the testbench file found on Brightspace: `flappy_controller_tb.vhd`.

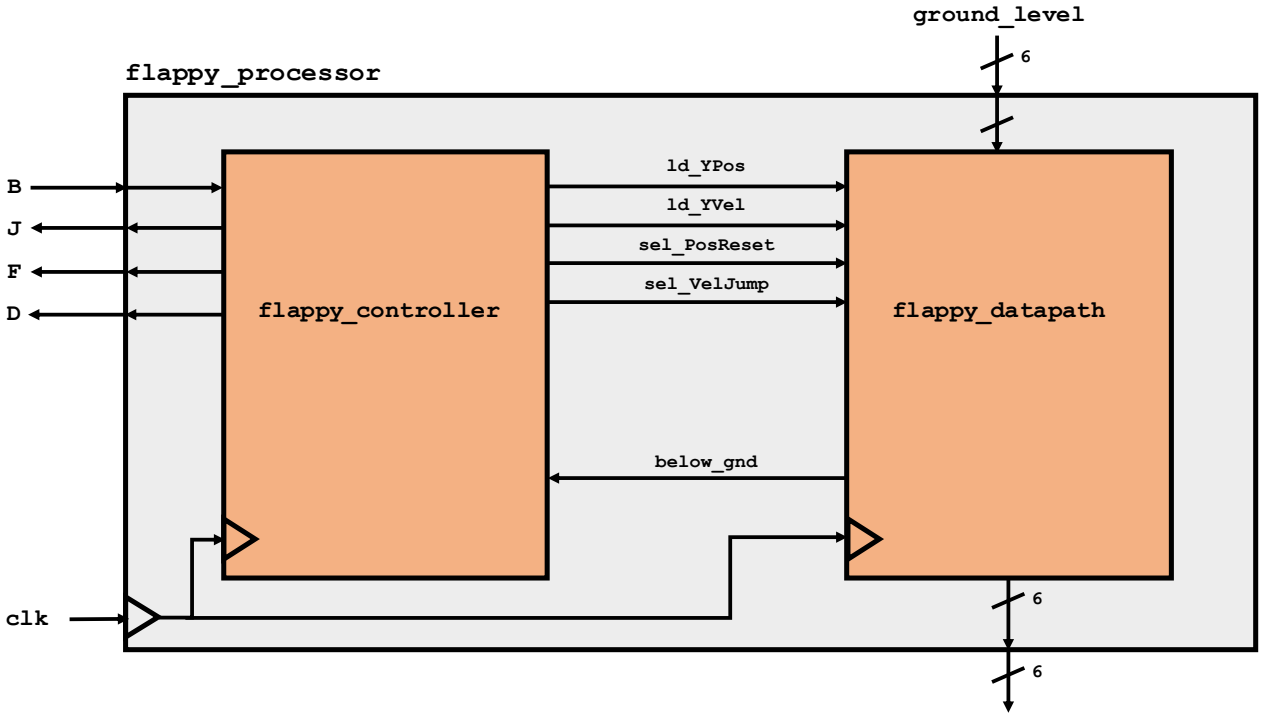
Look over your inputs and outputs closely to convince yourself that that the controller is performing its job correctly.

TA Verification (Task 3): Demonstrate successful implementation of your lemming controller to a TA / Instructor. Explain why the testbench demonstrates successful implementation of our desired behavior.

Make sure to take screenshots of your simulated output and be prepared to discuss the results in your report.

Task 4: Flappy Processor: Complete System

Our last task is to combine update our controller from Task 3 with our datapath from Task 2 to create our final system: `flappy_processor`



The schematic for the processor is shown above. You will be PROVIDED with a **complete** processor module in `flappy_processor.vhd` found on Brightspace.

Additionally, you have been PROVIDED with a testbench which should demonstrate the full capability of the custom processor (`flappy_processor_tb.vhd`).

NOTE: You may find issues in your controller or your datapath when testing with the full processor. Make sure to go back and fix any bugs. You can use the testport outputs to help you gather information about what is or is not working.

TA Verification (Task 4): Demonstrate successful implementation of your full custom processor to a TA / Instructor. Explain why the testbench demonstrates successful implementation of our desired behavior.

Make sure to take screenshots of your simulated output and be prepared to discuss the results in your report.

Report Structure / Guidelines

Unlike previous laboratories, you will be expected to create a formal lab report documenting your process. Being able to clearly communicate technical ideas in writing is a critical part of being an effective engineer. I should be able to pick up your report and be able to completely understand what your goal was, how you attempted to accomplish it, and what your results were.

The report should focus on the development of the full flappy processor. Discussion of any sub-parts should be placed in context of their purpose in the overall system.

Your report should have the following sections:

Part 1 - Project Summary: Summarize your project. Write a brief overview answering the following questions: What were you attempting? What was your general approach? What did you end up implementing and how well did it work?

Part 2 – Functional Description: Clearly describe the purpose of the custom processor you are building. What are the inputs and the outputs? What should their relationship be? You should include the high level state diagram here. (NOTE: you may use the provided HLSM, but make sure you pair it with a clear and concise description in words).

Part 3 – Circuit Design: Clearly describe how you took the functional description and translated it into a circuit. This should be done in three sections for the three primary sections of your design. Each section should include the schematic of that module. Most importantly, you must clearly document all of the work done to complete your schematic. You must demonstrate that you are capable to complete a design using the custom processor architecture.

> **Part 3.1 – Datapath -** make sure to discuss your process for designing the datapath based on the HLSM.

Part 3.2 – Controller – This section should include the FSM state diagram for your controller and a discussion about how the FSM is related to the original HLSM

Part 3.3 – Processor – For this section include the schematic of your processor and describe how it is implemented using the custom processor architecture.

Part 4 – Simulation Results: For each of the three parts of your circuit use results generated from Vivado to demonstrate whether your circuit did (or did not) work. Add annotations to your results illustrating how the timing diagram demonstrates the behavior described in the functional description. Make sure to reference specific events within the simulation by their time of occurrence.

Part 4.1 – Datapath

Part 4.2 – Controller

Part 4.3 – Processor

Part 5 – Conclusion: Provide another brief description of the project, your approach, and your results. Discuss what you learned during the project. Give specific examples of what you might do differently next time you go through a digital design process like this..

Appendix A: Include screenshots of your VHDL module code for Task 2, Task 3, and Task 4 (You do not need to include code for all your modules, just the three main ones).

Appendix B: Include screenshots of both prompts AND results for any generative AI tools you used in the development of this report. Make sure to review the AI policy to confirm what is and is not acceptable.

Technical Communication:

You will be evaluated on formatting, grammar, and clarity of language. Make sure you follow the guidelines below:

- Your report should look professional.
- Your report should begin with a title, author(s), date, class.
- Each section should be clearly identified and have section headings.
- **Any images / figures / tables that are included should have a number and a caption so they can be referred to in the text of your report.**
- Everything should be written in complete sentences, with proper grammar, with no spelling errors.
    - I strongly recommend you go to the UAlbany Writing Center to help improve your writing before submitting your report.