CS 4341 Digital Logic and Computer Design
Semester Project
Fall 2020

1 Objective:

The project is to build a system in a virtual hardware design language, such as Verilog, in a structural style. Such languages have a "behavioral style," which is more akin to C and C++, and a "structural style" which is more akin to hardware. For this semester, there are two approaches to the project.

Dr. Becker tends to use iVerilog for his lecture examples. Why? It's free. System Verilog and other languages are also available. In addition, various Verilog websites have libraries to connect to JavaScript or portals to other language compilers. There are features beyond the basics. If you want to use them, you may do so as long as you meet the requirements of the *approach*.

If you are a member of ***a cohort of three to five students***, your cohort is expected to take a structural approach. This means the project must be done in a manner that simulates the hardware more closely. Memory areas, for example, would be constructed of flip-flop modules instead of using built-in variables.

Due to the pandemic and the asynchronous teaching requirements, we will have ***an individual option***. If you do the individual option, your project may be done in the behavioral manner. Memory areas would be multi-bit, built-in register variables. Clock signals and timing will still be the same as the structural approach.

2 Topics:

Three topics are available to choose from form the project this semester: The classic ALU, a robot control system, or a design of your own.

2.1 The Arithmetic Logic Unit

The first topic for the project is to create an Arithmetic Logic Unit, using a structured approached with a Virtual Hardware Design Language such as Verilog. Mainly, the program is very close to a simulator for a programming calculator.

An ALU typically has the following operations

Math Functions: ADD, SUBTRACT, MULTIPLY, DIVIDE, MODULUS
Logic Functions: AND, OR, XOR, NOT, NAND, NOR, XNOR
Error Modes: DIVIDE BY ZERO, OVERFLOW
Support Functions: NO OPERATION, SHIFT LEFT, SHIFT RIGHT, RESET

An ALU has the following additional abilities
To store the first input
To store the second input
To store the results of the operation and use them for an output.
And to use the results of the operation to be one of the inputs.

Addition, subtraction, multiplication for basic math are typical. Long division and modulus are more of a challenge. In addition, an ALU can shift a binary value left or right, with or without a ring behavior. And ALU can have an error state on an overflow or a divide by zero error. And that means an ALU would also have a reset or clear command. And importantly, an ALU can be at rest, with a "no-operation." The data can be integer, or may include floating point, and can use various methods of sign. It is recommended that an ALU should operate on at least 16-bit integers. And that means the result of an operations should have room for at least 32 bits. The ALU should have at least ten "mode of operation", including error states, and supporting functions.

2.2 The Ultra-Violet Microbe Disinfecting Robot.

MIT, Japan, and other areas of the world have been showing off their robotic solutions to disinfect large areas of buildings, such as shopping malls. Instead of spraying chemicals, various robots have mounted ultra-violet lightbulbs that wipe out viruses and bacteria in public places after the crowds have gone home. Such a robot would have to be controlled. Basic functions would include movement: forward, back, left, and right. Additional abilities would be a motion detector, so that if something bigger than a microbe is moving in the area to not blast the area with ultraviolet light. The on-off switch for the ultra-violet light would be on a timer, so that the control circuit would have to pause the robot, check the area, then turn on the light for say, 60 seconds. All the while, another counter-timer circuit would be a check on the battery-life on the robot. When the robot reaches a certain level, it should trundle back to its recharging station. Once all these tasks, timekeeping, and energy-tracking are included...the robot would be performing math, logic, and must go through multiple operational modes. An excellent control system to simulate for the project.

2.3 Your Own Topic

Or, if a cohort is inclined, they can submit their own device they for with to build control circuitry. A basic ALU would have at least "10 operations and modes." A cohort's original project should be at least as complex.

3 Deliverables

The project will be worth 25 percent of the semester average. To that end, there are 5 deliverables to the project: Cohort Charter, Project Update 1, Project Update 2, the Final Project, and the Bonus.

Grading Scale

Deliverable	Due Date	Project Weight	Semester Weight
Cohort Charter	September 4th	10%	2.5%
Project Update 1:	September 25th	10%	2.5%
Project Update 2:	October 30th	10%	2.5%
Final Project	November 20th	70%	17.5%
Bonus:	November 18th	extra 20%	extra 5%

3.1 September 4th, Cohort Charter (10%)

The first deliverable for the project is the Cohort Charter. If you are working alone, then your charter should be very short with only a few details. If you have a Cohort of 3 to 5 members, you must include this information in the charter and how you will handle the tasks and communication.

Your Cohort Charter should contain the following parts:

- Cohort Name: The name of your Cohort
- Cohort Reason: Why did you pick this name for your cohort
- Cohort Members: Who are the members of your cohort? What is their name, and what is their official e-mail?
- Cohort Description:
    - What project do you plan to build? The ALU? The Robot? Something of your own? A short description in the cohort.
    - How will your cohort handle:
    - Communication? Particularly what software or messaging service this semester.
    - The addition of a new member?
    - The resignation of a member?
    - Conflict between the members?
    - Recognizing of the work performed by its members?
- Will you include Peer Grading? How will you implement it?

Note: If the cohort does not consider these situations, then Dr. Becker will handle any conflict as he sees fit. Typically, he dissolves cohorts that cannot work together into cohorts of one student each.

Turn in a PDF file of the Cohort Charter. Please name the file in the form of:

<CohortName>.Charter.PDF.

3.2 EXAMPLE CHARTER

COHORT CHARTER

The ITAON Chapter!

Cohort Name
Our Cohort is ITAON!!!
ITAON is an acronym for Instructors, Teaching Assistants, and Other Nightmares!

Cohort Members

- Dr. Eric Becker ewb123456
- The Black Knight tbk123456
- The Green Knight tgk123456
- Tim the Enchanter tte123456

Topic
A Dragon Incubator!

Description
Our project for this semester is to create the control circuitry for a dragon cloning facility. We intend to develop a simulator for the circuitry that will include incubator controls for the regulation of temperature for dragon eggs and young hatchlings. Such a system will require to be able to do both math and logic and to handle error states. Components will include the temperature maximum setting, the temperature minimum setting, and the current temperature. Commands will allow the temperature to be directly inputted by loading variables, or by having a +1/-1 increment/decrement in the controls. Tracking will include the number of eggs, the number of hatchlings, and the amount of energy needed to keep the incubator running. In addition, error states will exist for when the temperature gets to hot or to cold. In addition, our design will incorporate logic statements to safeguard the baby dragons from hostile temperatures. The system will also include a tracking of adding eggs, when the eggs hatch, and the eventual leaving of the incubator of a healthy baby dragon.

Charter

Yay, verily, we are the masters of dragons and their hatchlings, but we must be humble and agree on working with each other to overcome the struggles of the universe, and its Dragonbane Overlord, Dr. Becker.

To that end.

- The cohort will work together as a group for the entire semester for all parts of the project. We all stand together, or all fall together.
- The course has a group turn-in on blackboard/e-learning. We understand we need to turn in one complete copy of all parts before the due date closes. We shall meet once before each due date to make sure it is uploaded correctly.
- We will keep a journal on the DragonHub server, where we will host all official team communication and shared documents and code. We are keeping one working journal so we can check each other's work and help with participation.
- We do not make any member pay for food, or pay a fine, or surrender their belongings just because they are a member of our group. This is an academic exercise, not an industrial one.
- We will meet online during class cohort time and set aside time outside of class for communication. When we meet, we will make a log of everyone present on DragonHub note-taking service.
- If a student wants to join our cohort in mid-semester, all members must all agree, and must let the instructor know to update the blackboard/e-learning group.
- If a member wants to leave our cohort, we must inform the instructor, and make sure that the documents and code of our project remain intact and shared up to that date.
- No cohort member may alienate or harass another member.
- If a member never participates, and never shows up, we will contact them two times, and document the attempt by carbon-copying the instructor, the grader, and all cohort members to maintain transparency. After two *documented* attempts, we will ask the instructor to remove that member from the cohort.
- We want to have peer grading as part of our project, and we want to use Blackboard to be able to do so. We will ask the instructor to give us a grading option. We want the peer grading to count as a 100% reduction of the project grade for anyone not working on the project.
- If we disagree, and cannot work together during the semester, we will ask the instructor to break us into separate cohorts, and we will each turn in our own copy of the assignment. We understand that we can still work together, but each will be responsible for submitting their own copy.

3.3 September 25th, Project Update 1 (10%)

For the Project Update 1, your report should include the following:

- An updated description of your project. If it is the same, some minor edits should be in the description. Text that is removed should be marked with a ~~strikethrough~~. Material that has been added should be **blue and bolded.** If your cohort has worked on the project, there should be a minor change somewhere, even if it is just grammar. If it is a major change, the entire topic may be crossed out and an entire new topic description should be in blue.
- What hardware development language software have you settled on? iVerilog? System Verilog? Something different? A freeware Synopsis Synthesizer! What language and platform will your cohort be using? Cite the source.
- What tasks has the Cohort members been working on? Has anyone done something worth a "gold star?" Include a graphic, such as a pie-chart or bar-chart to show the group participation.
- A list of the actual required tasks that your project must perform to fulfill its description.
- A list of the commands that your system must recognize to achieve these required tasks.
- A list of the inputs that your system must recognize to achieve these required tasks.
- A list of the outputs that your system must produce to achieve these required tasks.

Turn in a single PDF file of the Project Update 1. Please name the file in the form of: <CohortName>.Update1.PDF.

3.4 Examples of List of Tasks, Commands, Inputs, and Outputs

For the ALU topic, it can be assumed that one possible task is addition. Result=2+5.
The result is stored in an accumulator register. From that one operation, we can imagine and design the tasks and parts needed, by thinking about how calculators work.

List of Tasks

- Clear the accumulator
- Input an integer
- Perform a sum.
- Store a result.
- Display the result.

List of Commands:

- Reset - clear and go back to idle and set the current value to 0.
- Addition – add the input value to the current value.
- No-Operation –be idle or return to idle without resetting the current value to 0.

List of inputs:

- Integers
- Addition command

List of outputs:

- The sum of the integers.

3.5 October 30th, Project Update 2 (10%)

For the Project Update 2, your report should include the following:

- What tasks has the Cohort members been working on? Has anyone done something worth a "gold star?" Include a graphic, such as a pie-chart or bar-chart to show the group participation.
- Have you started programming? Have you gotten a testbench and a breadboard module put together?
- By now, you should be able to list out the combinational logic components you need for your system. What are they? What are their symbols?
- Your cohort should have enough knowledge to start assigned digital abstraction labels to the project's commands. The result should be a command table.

Turn in a single PDF file of the Project Update 1. Please name the file in the form of: <CohortName>.Update2.PDF.

3.6 Examples of Components and Operations

Let us look at the ALU:

The ***combinational*** logic components needed for the ALU would include a multiplexer and a decoder as pieces needed to do commands. Individual tasks would to be done for Logic and for Mathematics. For Logic, the project might have an AND module and an OR module. For the Mathematics portion, your system may have a multi-bit adder/subtractor combinational logic circuit to handle addition and subtraction.

In addition, the project would have certain commands, and have created binary labels that can be recognized by the system.

Command	Binary Label	Description
No-Operation	0000	System is Idle, and stays in a ready state
AND	0001	perform an AND operation the current value with the input value
OR	0010	perform an OR operation the current value with the input value
ADD	1001	add the next input value to the current value
SUB	1010	subtract the next input value from the current value
RESET	1111	Clear all values to 0, and go to a ready state

3.7 November 20th, Final Project Due (70%)

The Final Project is the complete document. It should contain at least the following material:

Time to complete the project, and to get the program working. Your project must have all the matching material for a working system.

This is a warning. This is important.

Every year, I have cohorts that write the code first, then go back and try to write their design from their code. And every year, those cohorts realize that they "missed a bunch of stuff." Please, double check the material before you turn it in. I have had cohorts lose 20 points in one go because they left off a section of their turn-in.

What you need to turn in

- A List of Parts
- Circuit Diagram
- List of Opcodes (Triggers) and Operating Modes (States)
- State Machine, at least a top-level state machine.
- Verilog Code
- Verilog Output
- The Bonus

3.7.1 A List of Parts

Given the system you are building, it will be constructed of input wires, output wires, interface wires, gates, combinational logic components, and sequential logic components, and Datapath modules. Such modules include the testbench module and maybe as breadboard module. Inputs are the stimulus given in the testbench. Outputs are the results displayed in the test bench. All the parameters that go between the digital logic components are the interfaces. Sometimes individual gates are needed. Combinational Logic components are those modules that are not based on the passage of time. Sequential Logic components are those modules that are based on the passage of time.

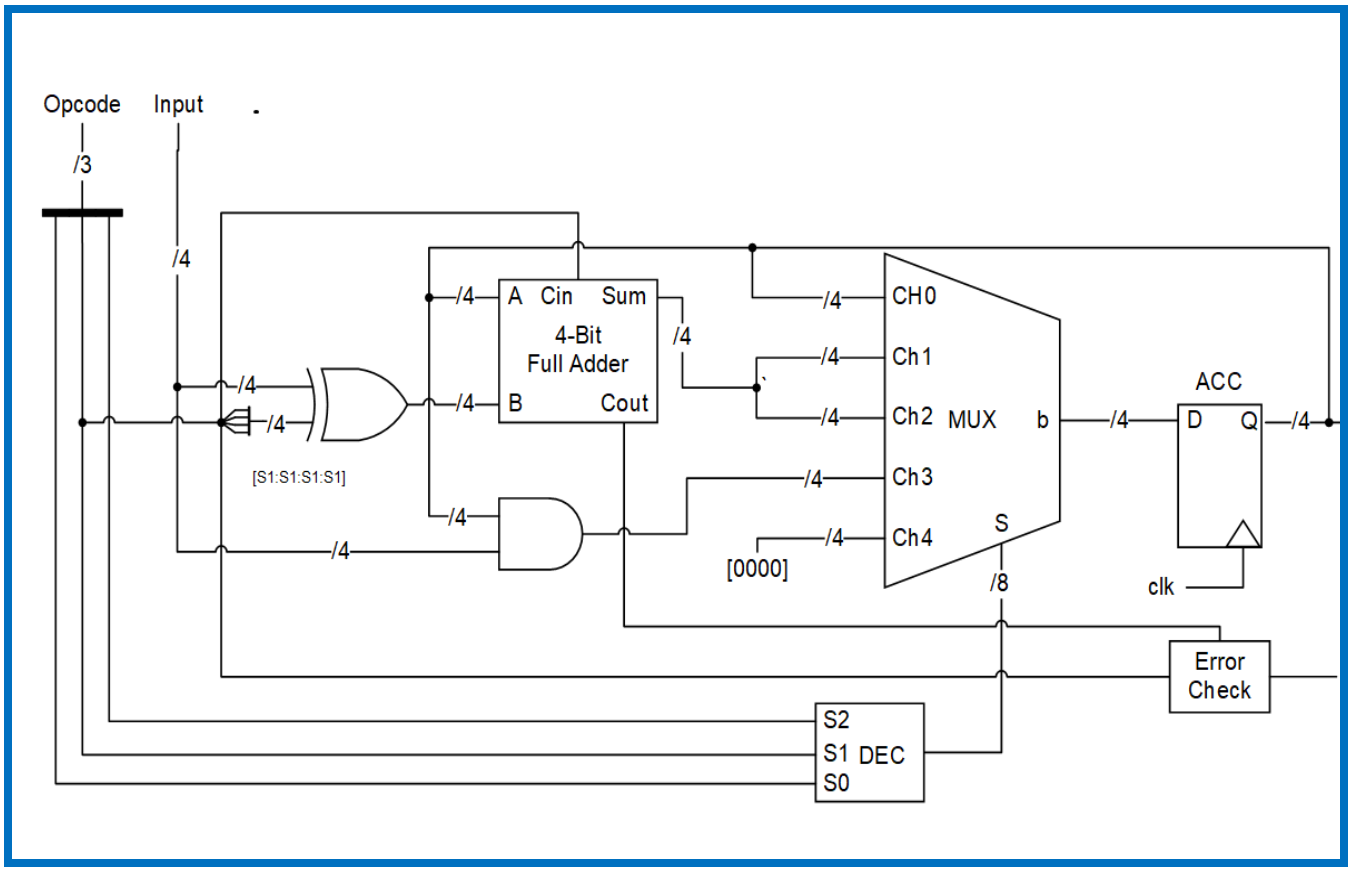
3.7.2 Example List of Parts

- Inputs
  - Clock-1 bit, the clock, feeds into the clock on the register Accumulator
  - Input-4-bits, binary integer for input
  - Opcode-4-bits, binary code to indicate the operation
- Outputs
  - Error-1 bit, true if an error state has occurred
  - Value-4 bits, the current output of the register Accumulator
- Interfaces
  - ACC.D, 4-bits, the next value into the register
  - ACC.Q, 4-bits, the current value from the register
  - Decoder.S0, 1 bit, lowest bit from the opcode
  - Decoder.S1, 1 bit, next lowest bit from the opcode, used for controlling add/subtract
  - Decoder.S2, 1 bit, highest bit from the opcode
  - FullAdder.A, 4-bits, takes in the current value of ACC
  - FullAdder.B, 4-bits, takes in the value of the XOR'd input
  - FullAdder.Cin, 1 bit, used for the +1 in a 2's compliment
  - FullAdder.Cout, 1 bit, the last carry, used for the Error Check
  - FullAdder.Sum, 4 bits, the actual sum, feeds into CH1 for addition and CH2 for subtraction
  - Multiplexer.b, 4 bits, output of the multiplexer into ACC
  - Multiplexer.Ch0, 4 bits, feedback for the current value of ACC into multiplexer
  - Multiplexer.CH1, 4 bits, results of addition into the multiplexer
  - Multiplexer.Ch2, 4 bits, results of subtraction into the multiplexer
  - Multiplexer.Ch3, 4 bits, results of AND into the multiplexer
  - Multiplexer.Ch4, 4 bits, 4-bit 0 used to reset the ACC
  - Multiplexer.Ch5, CH6, Ch7-4bit channels, unused.
  - Multiplexer.S, 8-bits, the one-hot control into the multiplexer
- Gates
  - XOR Gate, 4 bits, one channel is the input, the other channel a concatenation of Decoder.S1 4 times.
  - AND Gate, 4 bits, one channel is the input, the other the current ACC Q value
- Combinational Logic Components
  - Decoder, 3-to-8 bits, converts opcode into Multiplexer selection
  - Multiplexer, 8-channel, 4-bit multiplexor for result operation
  - Full-Adder, 4-bit, Full adder
- Sequential Logic Components
  - ACC, 4-Bit D Register that contains the current system value
- Modules
  - Error Check-Tests for an error in subtraction, inputs S1 and Carry out.
  - Test Bench-Main module, runs clock and stimulus. (not shown)
  - Breadboard-convenient module to allow easy assembly. (not shown)

3.7.3 Circuit Diagrams

Your project should include a circuit diagram, containing wires, components, registers and Datapath modules. The circuit diagram should be complete and have good labels. Wires of one bit typically do not have a bus size, but any group of wires greater than 1 will need to be marked. For a fan in/fan out, this is often skipped. In the example below, a fan out has been included. If your system is more complex, you may have more than one diagram.

Here is an example of a partial ALU that matches the List of Parts.



3.7.4 List of Opcodes (Triggers) and Operating Modes (States)

For the system to function, the program will run against a system clock. Your system should be capable of doing processes and operations against time, which requires a sequence. Sequences require states and triggers and transitions, which make it a finite state machine, or automata. A zero state, an idle or ready state always exists. And as time passes, one option is that nothing will happen. That option is a trigger referred to as no-operation. Your project should include a list of the active states, called Modes of Operation, a set of commands referred to as operation codes, or OpCodes, and you should have a state table to indicate who the modes and opcode are connected.

3.7.5 Example of Opcodes, Modes of Operation, and States

Op-Code Table

Command	Opcode	Description
No Operation	000	No action on this clock tick
Add	001	Add Input to ACC
Subtract	010	Subtract Input from ACC
AND	011	AND Input with ACC
Reset	100	Reset Register to all 0's

Modes of Operation

MODE	Code	Description
Ready	000	System is idle
AND	001	Performing an AND operation
ADD	010	Performing addition
SUB	100	Performing subtraction
ERR	111	An error has occurred, and will not return to ready until reset.

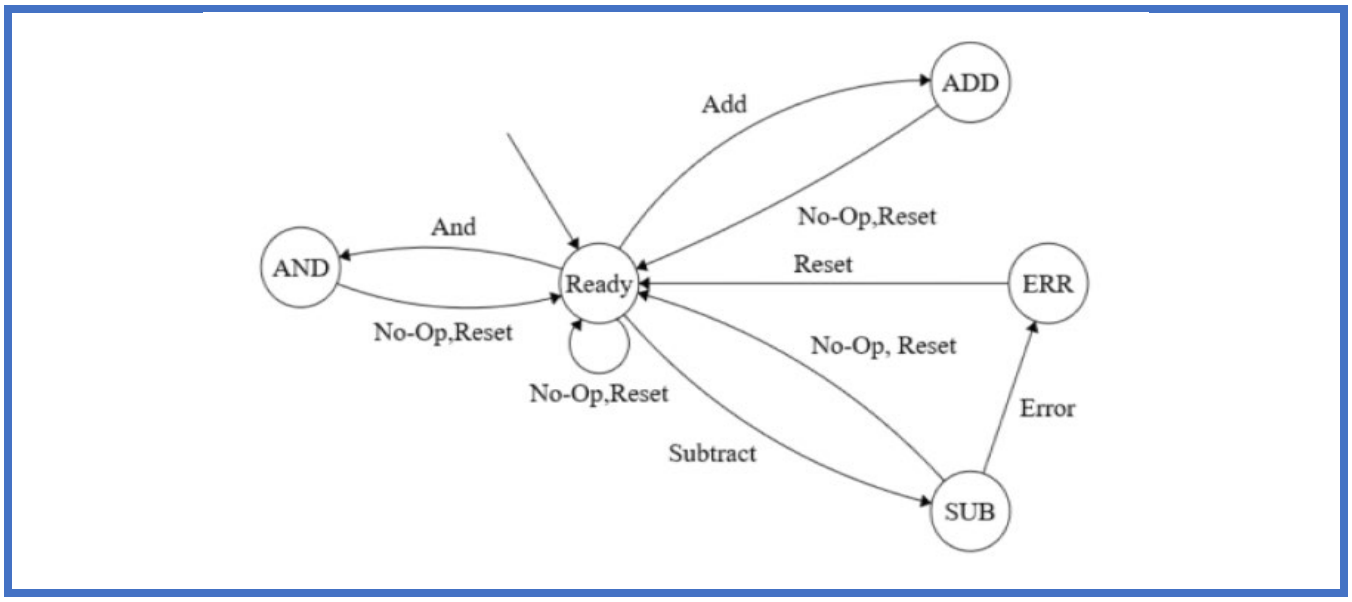
State Table

Current State	Command	Next State
ADD	No-Op, Reset	Ready
AND	No-Op, Reset	Ready
ERR	Reset	Ready
Ready	No-Op, Reset	Ready
SUB	No-Op, Reset	Ready
Ready	And	AND
Ready	Subtract	SUB
Ready	Add	ADD
SUB	Error	ERR

3.7.6 State Machine

Your system will now contain one or more state machines. You should turn in at least the top-most levels state machine for your system, the one that works on the main opcodes of the stimulus of the testbench. An ALU may be in a logic state or math state or error state or ready state. A robot could be in an idle state, or travelling, or turning left or turning right, or contacting a satellite. These different states are visible on the state machine that goes with the opcodes and modes.

3.7.7 Example State Machine, Top Level



3.7.8 Verilog Code

The next section is your Verilog code for your project. One thing you must make sure is that your previous sections match your code: parts, operational codes, and modules actually match the code that you turn in. Many cohorts in the past have written the code first, then gone back to do the diagrams and then realize they must rewrite their code.

Verilog has several styles of coding. Two are of interest for this project. Structural programming means the Verilog is written with the view to being a simulator, in which components are made of bitwise operations using the timing constraints in parallel programming style. The other form is behavioral, in which sections of the code are not in parallel, but are written sequentially, more like C++.

3.7.8.1 For Cohorts:

For this project, for cohorts, you are to turn in code that is manly structural in nature. The clock, the registers of flip-flops, and the test bench with a stimulus need to work correctly. For the subtasks, you are expected to also have structural code, using binary operations to create components, and then combining components into larger Datapath modules.

If a particular Datapath module is a problematic module for your cohort*, it is fine if it is done in a behavioral style*. For example, everyone should have done an adder/subtractor on their homework in a structural style. Using a behavioral with a flat addition operation + is not acceptable. Long division and modulus, however, are a multi-step task in themselves, and those would be fine as behavioral components for a project doing an ALU. Having everything 100% behavioral is not acceptable.

3.7.8.2 For Individuals:

If you do not have a cohort, and are working alone, because of this year's pandemic or other complications, you may use behavioral programming style without restrictions.

3.7.9 Verilog Output

Your output for your system must be complete enough to show your system in operation. The testbench stimulus would start to run, and the output would show the behavior of the system in operation. The output should be readable. Various ASCII and formatting may be used to make an output that can be read. Your output should also use enough operations to show a complete run of the circuit. For example, a rover would have to start, stop, start, turn left, turn right, go back to stop.

3.7.10 Example Verilog Output

Example:
In this example, the clock is shown to demonstrate the timing behavior. Input is the variable from the testbench, shown as an integer and as a binary. ACC is the current value of the register, shown as an integer and a brinaey. Instruction shows the command as a string label, and the opcode as a binary number. Next is the result, the next state of the system at the end of the clock tick. And the final column is a one-bit display to show if an error has occurred.

```
C|       |      |           |      |
L|Input  |ACC   |Instruction|Next  |
K|#|BIN  |#|BIN |CMD|OpCode |#|BIN |Error
-|-|----|-|----|------|----|-|----|-----
0|0|0000|X|XXXX|Reset | 100|X|XXXX| X
1|0|0000|X|XXXX|Reset | 100|0|0000| 0
0|0|0000|0|0000|No-Op | 000|0|0000| 0
1|0|0000|0|0000|No-Op | 000|0|0000| 0
0|5|0101|0|0000|ADD   | 001|0|0000| 0
1|0|0000|5|0101|ADD   | 001|5|0101| 0
0|0|0000|5|0101|No-Op | 000|5|0101| 0
1|F|1111|5|0101|AND   | 011|5|O101| 0
0|A|1010|5|0101|ADD   | 001|5|0101| 0
1|A|1010|5|0101|ADD   | 100|F|1111| 0
```

3.7.11 The Bonus

Verilog and building a circuit and being able to generate an output that uses time constraints is all well and good. But… it is plain text on a screen. But I want to see something better. Take your project and turn it into something cool. Different library packaged exist to connect Verilog to other languages. Some versions of Verilog have libraries that connect to JavaScript. Other methods include file dumps and screen scrapes. Other bonuses have been to create an ALU out of Minecraft© Redstone.

For this semester, build something cool on top of Verilog, (Or at least using your design in a super-cool medium) and create a video or presentation that can be shown to the class online. Due to the truncated semester, the day to show the presentation on or before Wednesday, November 18th. Why not after the Fall Break? The pandemic has changed the regular calendar, and the 18th is the last formal day that we meet for something other than exams.

Good luck, everyone!