CS 4341 Fall 2021
Project, Part 3
Assigned Week of February 21st
Early Due Date on March 11th, 11:59 pm
(for +15 points Part 3)
Expected Due Date March 25th, 11:59 pm

1 Objective

For this semester, you are to form a team, and build a simple processor, such as an ALU, in a hardware design language. For the third part of the project, cohorts must have sequential logic components incorporated into a circuit diagram and have a matching Verilog code and output. In addition, the cohort should have their first draft of a state machine that describes the behavior of the circuit.

2 Tasks

The following tasks to complete for this project will be:

- Update the Description and Cohort Information from Part 1.
- Read the description of the sequential circuit and update the combinational circuit from Part 2. This includes:
    - Update the system design
    - Update the circuit diagram
    - Include a state-machine diagram with state table
- Construct this next circuit in Verilog
- Run the Verilog simulation and capture the output.
- Keep a journal of meeting, tasks, and progress
- Submit a Peer Participation Report

2.1 Update Description and Cohort Information

Now half the project has been submitted, changes may have occurred in the cohort membership, description, new ideas about the project, or even new software, including the name of the cohort and its description, its members, the software, and the description of what you are to build. Anything removed from Part 1 of the projects will be marked with a ~~strikethrough~~. Anything added to the update will be marked in **bolded, blue text**. It is fine if there are no further changes in this description. Turn this in as a file named <cohort>.update.3.pdf.

2.2 Memory, Control, Logic, and Arithmetic

For this next part, the cohort will update the circuit from part 2. Now that sequential logic is involved, several key changes must be made to the control of the circuit.

2.2.1 Memory and Control

- First, add a 32-bit memory register to the circuit between the output of the circuit and the output of the multiplexer. A memory register will be identified by a flip-flop with a multi-bit input and multi-bit output bus. This register is called the ACCUMULATOR or ACC.
- Next, a clock signal will be added as a new input into the memory register. The clock signal is a single-bit square wave that drives the flip-flops of the memory register.
- Then, the memory register output will feed into one of the multiplexer channels. This feedback channel, typically channel 0, will represent the “no-operation” or no-op.
- One channel of the multiplexer will be nothing but 0, representing the reset of the circuit.
- One channel of the multiplexer will be nothing but 1’s, representing the pre-set of the circuit.
- Finally, the lower 16 bits of the memory register output will feed back to replace the second input into the system.
- Include a table of operational codes that has the memory and control operations: No-Op, Reset, and Pre-set

2.2.2 Logic

The multiplexer should have arithmetic channels for addition, subtraction, multiplication, division, and modulo from part 2. From Memory and control, the no-operation, reset, and preset will take up another three channels.

Now, time to add the logic from part 1. Put in modules for controlling set for another 7 channels. Each of these operations will have two 16-bit inputs, and a 32-bit output padded to fit the multiplexer.

Update the table of operational codes to have the logic operations: AND, NAND, NOR, NOT, OR, XNOR, XOR

2.2.3 Arithmetic

Currently, the cohort should have addition, subtraction, and multiplication as structural combinational logic components. Division and Modulus would be behavioral logic components. Memory is now available to the cohort, and with memory comes the SHIFT operator. Researching and implementing a shift-based multiplication operation is worth +10 points on part 3.

Update the table of operational codes to have the math operations addition, subtraction, multiplication, division, and modulus.

2.3 Update System Design Description

The System Design description has six tasks to be completed: the list of inputs, the list of outputs, the list of interfaces, the list of parts, the top-level circuit diagram, and a state machine.

2.3.1.1 The List of Inputs

The inputs for this circuit should include 4-bit opcodes, a single 16-bit integer, and a clock signal. The second input from part 2 becomes an interface from the memory register.

2.3.1.2 The List of Outputs

The output for the circuit includes the error codes from part 2. The output for the circuit will change since the source of the output is now the memory register instead of the multiplexer.

2.3.1.3 The List of Interfaces

The interfaces will change from part 2 of the project. The feedback from the lower 16 bits of the output will become the second input to all the modules. New interface lines will connect each of the logic modules to the multiplexer.

2.3.1.4 The List of Parts

New parts are added into the system. For the combinational logic, the logical operation modules will be added. A new section will include sequential logic, which will have the description of the 32-bit memory register.

2.4 Update Top-Level Circuit Diagram

Only the top-level circuit diagram is needed for this portion of the project. The interior workings of the arithmetic modules, logic modules, memory registers, the decoder, and the multiplexor are not required. The overall layout of the circuit begins with the inputs far left, next the arithmetic modules and logic modules near left, then the multiplexor and decoder near right. Next will be the memory register, and finally the output on the far right. Each line should be connected to the correct component, labeled with their identifiers, and marked with a bus-size shown as a slash on the line, with the size beneath.

Save this as a file called <cohort>.SystemDesign.3.pdf

2.5 New: The State Machine

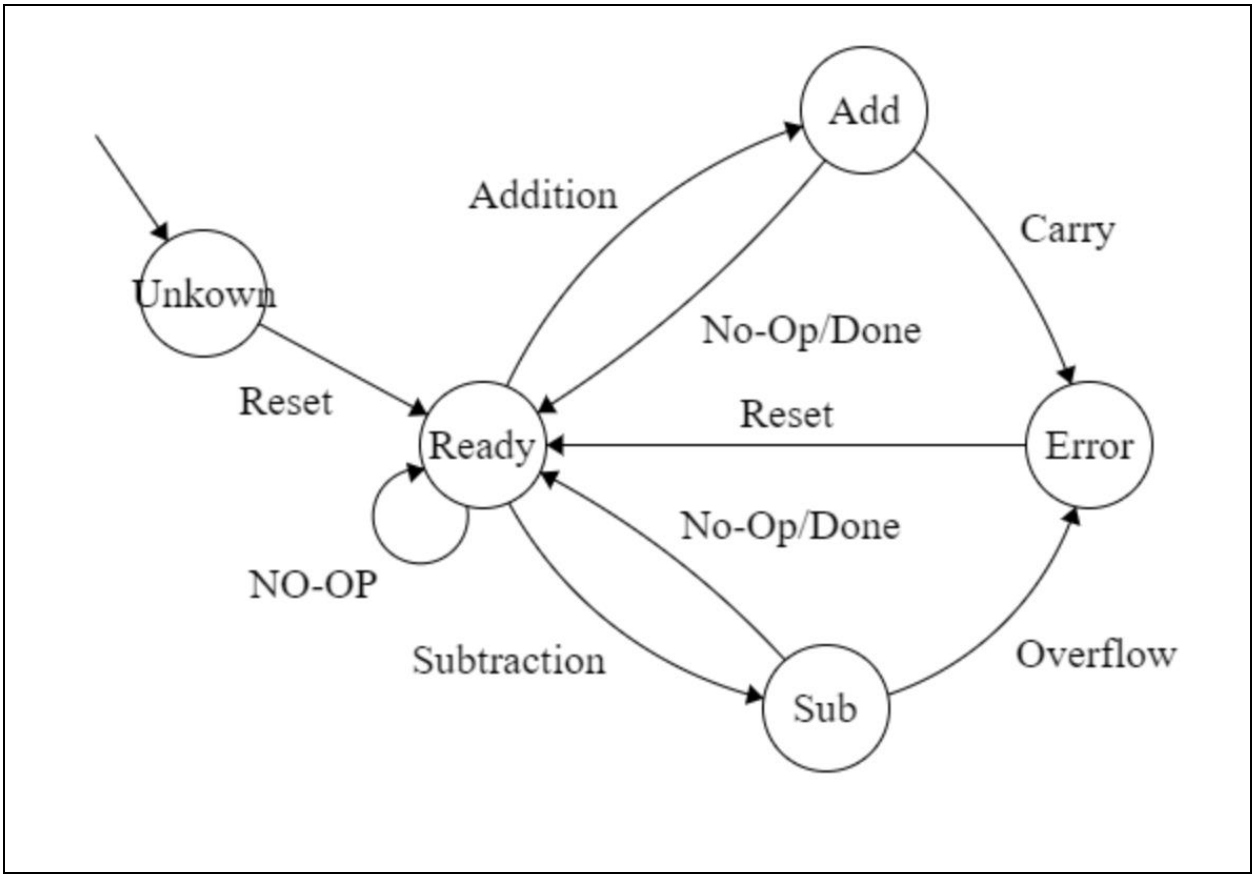
For the last item in this document is the State Machine with State Table. Please read carefully. This state machine is ***not*** part of a Verilog program. The state machine is for testing *use cases* now that time is involved in the circuit. A use case is one possible path of execution through the circuit. Combining all the use cases in a single diagram becomes a state machine, and even further, a reduced state machine.

Instead of the circuit simply being on or off, it has some specific behaviors. Let's look at subtraction.

- When the circuit starts, the circuit is in an unknown state.
- Once the circuit is reset, then the circuit is in a ready state, with the memory register at all 0s and the error signal is 00.
- So long as no operation is run against the clock, the circuit remains in the ready state.
- Then the first number 7 is inputted with the opcode for Addition. The circuit operates, adding 7 to the memory register, and when done, returns to the ready state and the error signal is 00 and the output is 7.
- Next, the second number 5 is inputted with the opcode Subtraction. The circuit operates, subtracting 5 from the memory register, and when done, returns to the ready state and the error signal is 00 and the output is 7.
- If a carry or overflow occurs in the circuit, the circuit is in an error state until the circuit is reset and sent back to the ready state.

Two possible state machines can meet these criteria. The first is an unreduced state machine, with one state per operation. The second has states where the inputs and outputs match, reduce to a single state with multiple triggers on the transition. These examples are only for Add and Subtract. The state machine for the project will have to handle all the operations. (Hint: Try the solving for the reduced state machine...it will be easier to draw.)

2.5.1 Unreduced State Machine



State Table

Current State	Trigger	Next State
Unknown	Reset	Ready
Ready	No-Op	Ready
Ready	Addition	Add
Ready	Subtract	Sub
Add	done	Ready
Add	Carry	Error
Sub	Done	Ready
Sub	Overflow	Error
Error	Reset	Ready

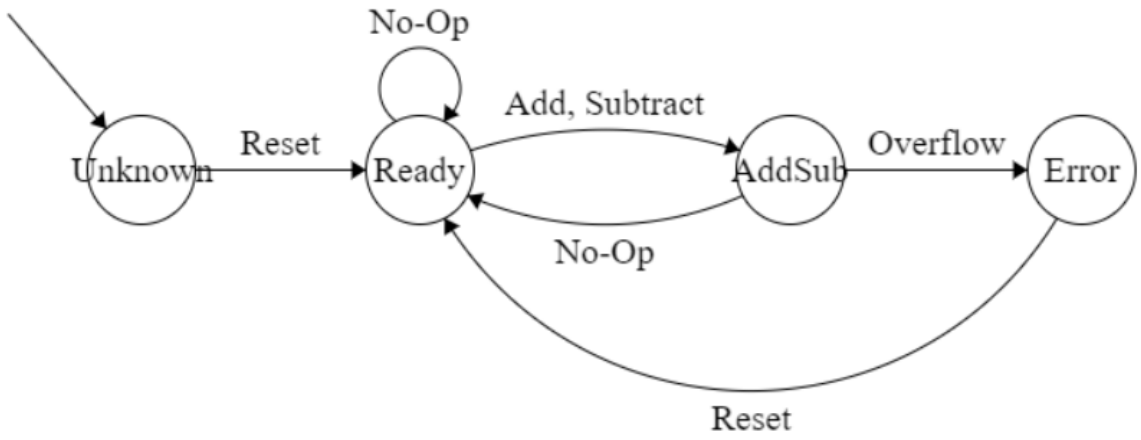
2.5.2 Reduction

Look at the state table:

- Add and Sub have the same next states on Ready and on Error.
- Ready goes to Add and Sub on a parallel trigger
- Done and NO-OP always return to Ready.
- Overflow and Carry are the same value for an Adder/Subtractor combinational component..

Current State	Trigger	Next State
Unknown	Reset	Ready
Ready	No-Op	Ready
Ready	Addition**/Subtraction**	Add **and Sub**
~~Ready~~	~~Subtract~~	~~Sub~~
Add **and Sub**	done No-Op	Ready
Add **and Sub**	Carry or Overflow	Error
~~Sub~~	~~done~~	~~Ready~~
~~Sub~~	~~Overflow~~	~~Error~~
Error	Reset	Ready

2.5.3 Reduced State Machine



Reduced State Machine

Reduced State Table

Current State	Trigger	Next State
Unknown	Reset	Ready
Ready	No-Op	Ready
Ready	Addition	AddSub
Ready	Subtract	AddSub
AddSub	No-Op	Ready
AddSub	Overflow	Error
Error	Reset	Ready

2.6 Write the Verilog Code

The next part of the project is to write the Verilog code that matches the circuit you designed in section 2.2. As with Part 2 of the project, the code begins with a testbench and a breadboard. Also, new modules will be created for a memory register made of Flip-Flops, and the 16-bit Logical Operators. ***The identifiers of these modules must match the identifiers chosen in the system design list of parts.***

The Verilog for a single D Flip-Flop is:

```
//=============================================
// DFF
//=============================================
module DFF(clk,in,out);
  parameter n=1;//width
  input clk;
  input [n-1:0] in;
  output [n-1:0] out;
  reg [n-1:0] out;

  always @(posedge clk)
  out = in;
 endmodule
```

The Verilog for a Register of D Flip-Flops is:

```
//---------------------------------------------
// instantiate state register
//---------------------------------------------
DFF #(6) state_reg(clk,  next, state) ;
```

All of these modules will be instantiated in the breadboard module and connected using parameters. The names of the parameters between those modules must match the identifiers chosen in the system design list of interfaces.

The testbench module will control the input and output in the stimulus. The stimulus will have hard-coded values to be the inputs to the circuit. The stimulus will also display the matching output. ***The names of the input and output variables in the TestBench must match the identifiers chosen in the system design list of inputs and list of outputs.***

What will be seen in the Verilog code is the following. All the variables are created first. Next, the modules are instantiated with those variables being the parameters. Then down in the initial portion of the TestBench, chose the input, and the command, do a delay (such as #60;), then display the output. Then do the next instruction.

Turn in a file named <cohort>.Part3.v

2.7 Run the Code and Capture the Output

Once the code is running, the cohort must demonstrate that the modules work, and that the output be clearly readable. What is more, your test bench will now have to include a ***Clock signal to indicate when the system will change.***

To do so, do the following 15 operations in the TestBench:

No-Operation, Reset, Preset, Add, Subtract, Multiply, Divide, Modulus, And, Or, Not, Nand, Nor, Xor, Xnor,

Turn in a capture of this file in the form of <cohort>.part3.pdf

The output might be something like:

I	Input	F	Feedback	Operation	OpCode	Output	Output	Error
0	0000000000000000	X	XXXXXXXXXXXXXXXX	RESET	1000	0	[0000000000000000] [0000000000000000]	00
0	0000000000000000	0	0000000000000000	NO-OP	0000	0	[0000000000000000] [0000000000000000]	00
7	0000000000000111	0	0000000000000000	ADD	0001	7	[0000000000000000] [0000000000000111]	00
3	0000000000000011	7	0000000000000111	SUB	0101	4	[0000000000000000] [0000000000000100]	00
250	0000000011111010	4	0000000000000100	MUL	0010	1000	[0000000000000000] [0000001111101000]	00
32	0000000000100000	1000	0000001111101000	DIV	0011	31	[0000000000000000] [0000000000011111]	00
4	0000000000000100	31	0000000000011111	MOD	0100	3	[0000000000000000] [0000000000000011]	00
32	0000000000100000	3	0000000000000011	AND	1001	0	[0000000000000000] [0000000000000000]	00
42	0000000000101010	0	0000000000000000	OR	1010	42	[0000000000000000] [0000000000101010]	00
0	0000000000000000	42	0000000000101010	NOT	1100	65493	[0000000000000000] [1111111111010101]	00
65535	1111111111111111	65493	1111111111010101	XOR	1011	42	[0000000000000000] [0000000000101010]	00
65525	1111111111110101	42	0000000000101010	XNOR	1111	32	[0000000000000000] [0000000000000000]	00
65535	1111111111111111	32	0000000000100000	NAND	1101	65503	[0000000000000000] [1111111111011111]	00
65535	1111111111111111	65503	1111111111011111	NOR	1110	0	[0000000000000000] [0000000000000000]	00
0	0000000000000000	0	0000000000000000	PRESET	0111	4294967295	[1111111111111111] [1111111111111111]	00

2.8 Journal

A calendar of events, meeting, discussion, choice of software, writing code, that has the date, a description of the work done, and who was doing the work. Turn in the file called <cohort>.Log.3.pdf

Date	Task	Who
October 9	Work out draft circuit diagram	Fred, Daphne, Velma, Norville
October 16	Update system design	Velma, Fred
October 18	Adding Logic modules to Verilog	Daphne, Norville
October 20	Start work on State Machine	Velma, Fred
October 22	Check to make everything matches	Fred, Daphne, Velma, Norville
October 24	Solves mystery of working feedback	Norville
October 26	Get all controls working in Verilog	Norville, Daphne
October 28	Edit everything to make sure all names match	Fred, Daphne, Velma, Norville

2.9 Peer Grade

A separate assignment is available on blackboard, a Peer Participation Report. This document is required for every member of the cohort individually. See blackboard for details.

3 Turn-In

- The updated Cohort and Project description, named <cohort>.update.3.pdf
- A file called <cohort>.SystemDesign.3.pdf that contains
    - Description of all inputs
    - Description of all outputs
    - Description of all interfaces
    - Description of all parts, combinational or sequential.
- A correctly labelled Top-Level circuit diagram
- A correctly labelled State Machine diagram with it state table.
- A text file, named <cohort>.Part3.v, that contains the Verilog code
- A text file, named <cohort>.Part3.txt, that contains the Verilog output
- The Work Log named <cohort>.Log.3.pdf that contains the Journal
- A Peer Participation report will be separate on Blackboard.