## 18-240: Structure and Design of Digital Systems

Electrical & Computer ENGINEERING

# HW6 Solutions [6 problems, 64 points]

*Covers lectures L12 – L14* *Due: 30 October 2023*

## Drill Problems [28 points]

1. [12 points, Lecture 12] You've now learned the **`parameter`** keyword in SystemVerilog, which is used to specify signal sizes at the time a module is instantiated. Use this knowledge to collect and update a bunch of SystemVerilog modules for commonly used datapath components. We are also including the sequential components and other often-used building blocks.

   In a file named **`library.sv`**, build the modules shown on the following page. You may update the **`library.sv`** file you turned in for previous homework exercises. Again, you are free to copy code you have found in lecture slides, homework problems or the textbook.

   **`library.sv and library_tests.sv`** are posted on Canvas.

2. [8 points, Lecture 12] Design and implement a serial magnitude comparator. As your inspiration, use the serial adder from Lecture 12. If implemented in SystemVerilog (which I'm not asking you to do, yet), your circuit would have the following header:

   Here's a design using 2 FFs. You can use register in similar fashion. The gates on the left are looking at the current inputs **`A`** and **`B`** to see how they compare (i.e. is one greater than the other?).

   The FFs are used to keep track if they have ever seen a situation where one input is larger than the other. The OR gate is used to ensure that if the FF is currently storing a 1, then the D input will be a 1.

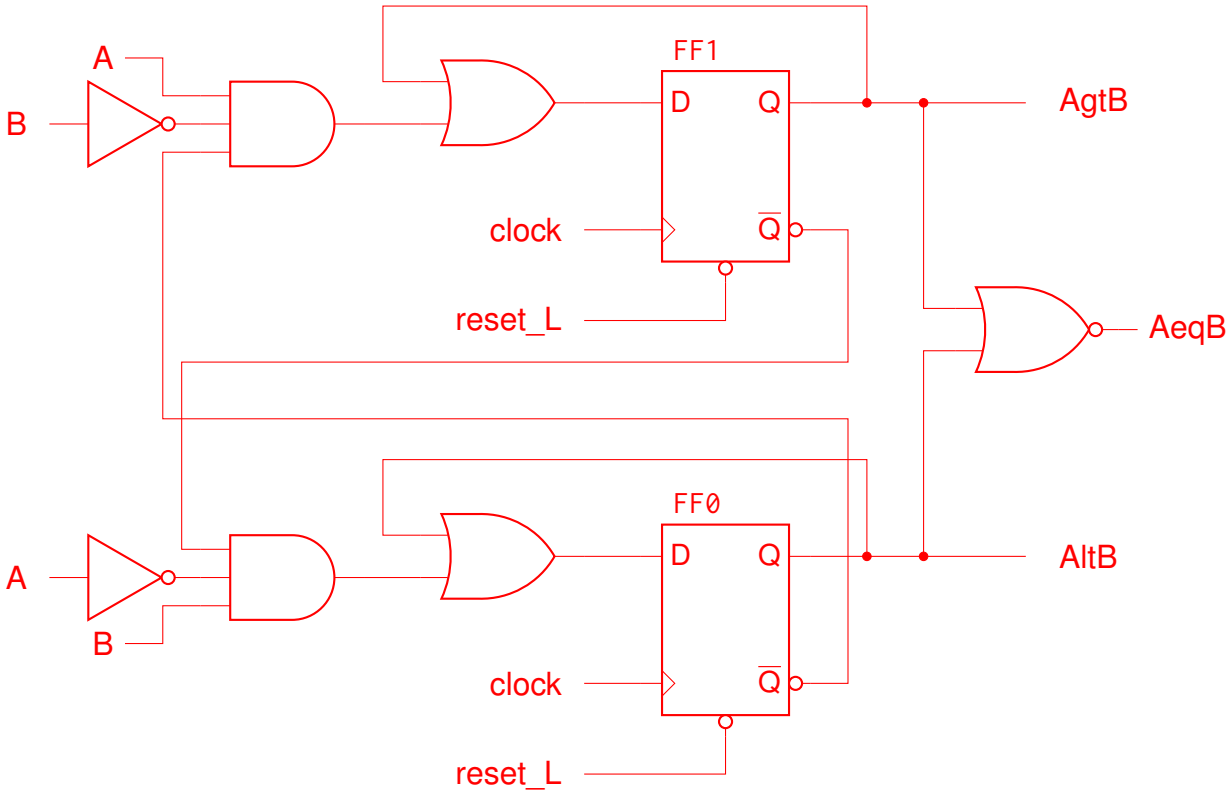
   So, if we look at **`FF0`**, it will be reset and hold a zero. Then, if **`A`** and **`B`** are both zero, then the D-input to **`FF0`** will be a zero (zero in the FF OR zero from the AND gate). And, Q stays at zero.

   But, if the next clock period, **`B`** is a one and **`A`** is a zero, that means that **`AltB`** is true. **`FF0`** will have a one on it's D-input (zero in the FF OR one from the AND gate) and thus the **`AltB`** output will be a one.

   At any point after that clock period, it doesn't matter what value is on **`A`** and **`B`**, as **`FF0`** holds a one and thus the D-input is a one (one in the FF OR **`x`** from the AND gate).

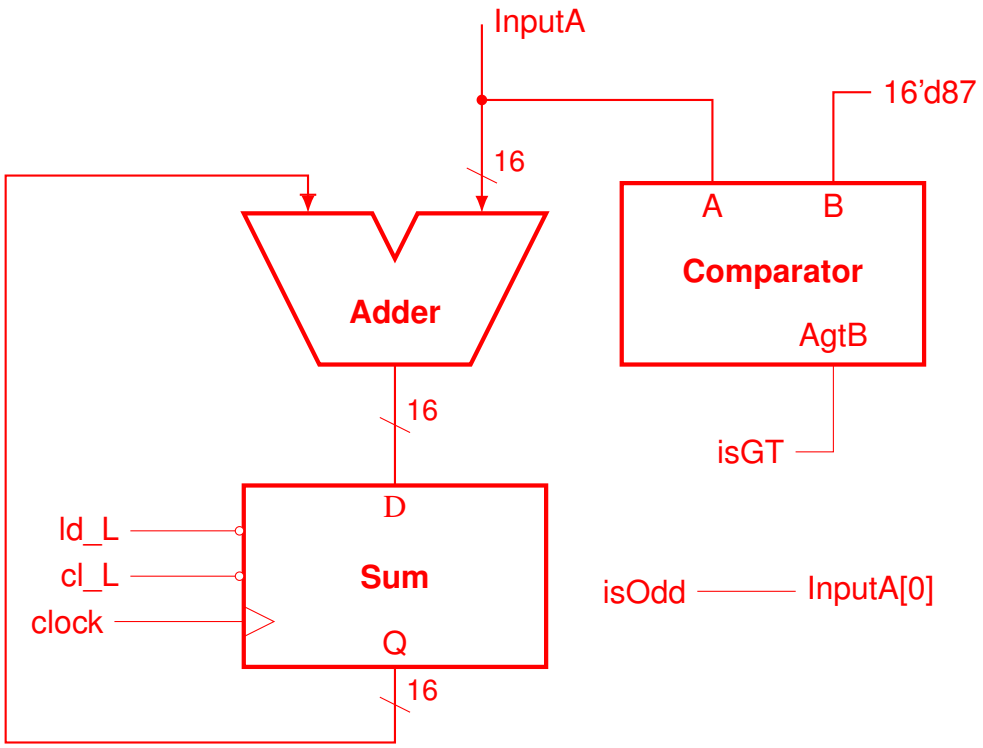
   The NOR gate is used to calculate **`AeqB`** and assert it if neither of the other outputs are a one.

   NOTE: This problem can also be solved as a FSM with a three-state STD. Reset into a "EQUAL" state and then move to a "GREATER THAN" state if $A \cdot \overline{B}$ and a "LESS THAN" state if $\overline{A} \cdot B$. Once in those states, stay there.

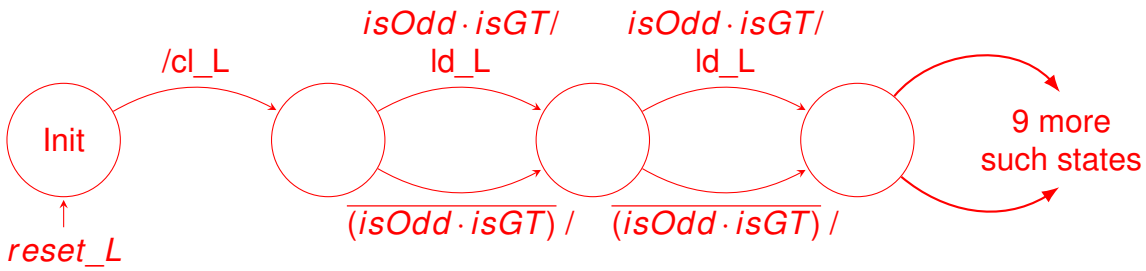

A
B
FF1
D
Q
AgtB
clock
Q
reset_L
AeqB
FF0
A
D
Q
AltB
B
clock
Q
reset_L

3. [8 points, Lecture 12] Modify the datapath from L12, slide 24 so that a series of 12 **`InputA`** values will be summed, but only if they are odd and greater than 87. In other words, it will do something like this code snippet:

The circuitry is pretty easy. We look at the LSB of **`InputA`** to figure out if it is odd or not. Use a comparator to figure out if it is > 87. Both become status points. Like so:




The FSM change is also simple: Just add enough states to sum 12 times, checking if **`isOdd`** and **`isGreaterThan`**. Changing to Mealy machine keeps the state count low. In L13, you learned/will learn how to use a counter, which could make this easier. But, for now:



## **Non-Drill Problems** [36 points]

4. [8 points, Lecture 12] Write SystemVerilog code for the Serial Magnitude Comparator of problem 2. Your library file has a Flip-flop module. You should probably write test benches to ensure it works, though they won't be graded.

   You can find **`hw6prob4.sv`** on Canvas.

5. [14 points, Lecture 13] In Lecture 13, I presented several alternative designs for the ones-count problem. The alternative on Slide 40 eliminated the loop counter and replaced it with a comparator ("And Another"). Implement this version of the entire HW thread in SystemVerilog.

   You can find **`hw6prob5.sv`** on Canvas.

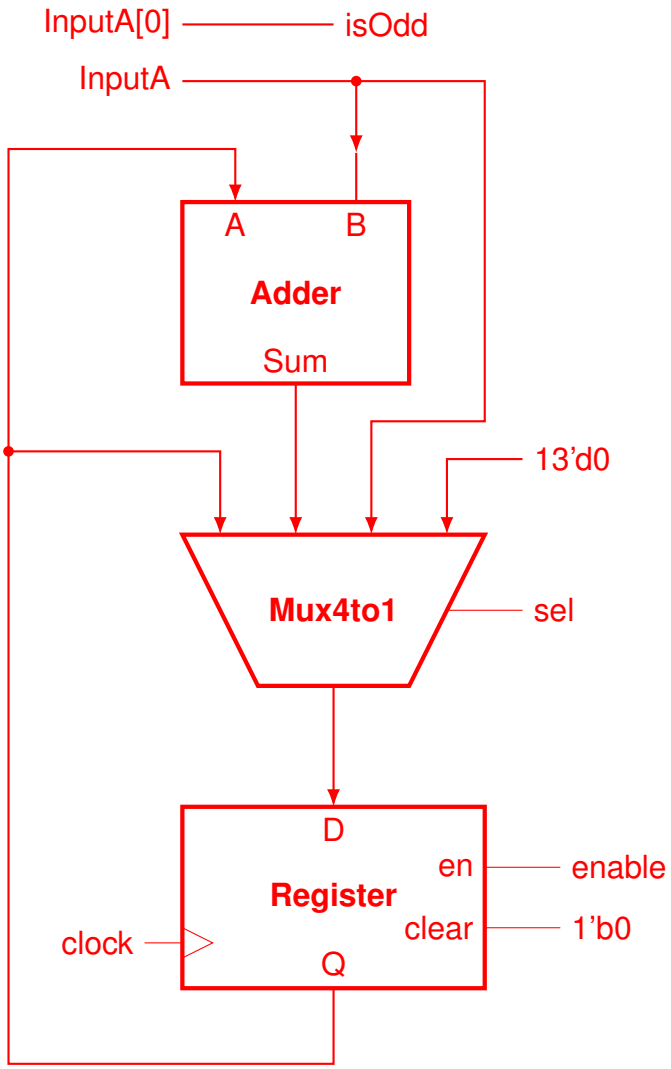
6. [14 points, Lecture 13] Design and implement a hardware thread (i.e an FSM-D) to sum the odd unsigned numbers on its input.

   (a) Four register transformations total. They all are visible on the output as **`sumOdd`**.

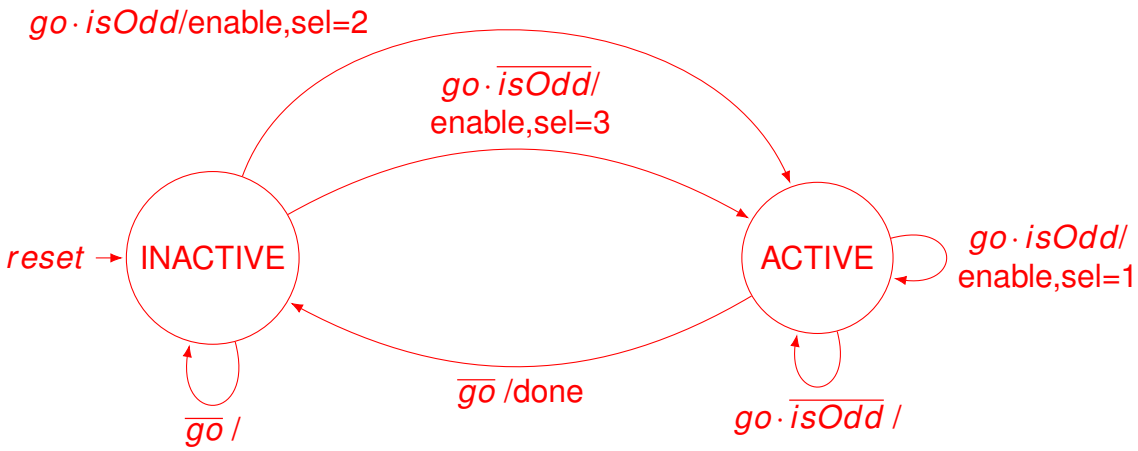
   - $sumOdd \Leftarrow sumOdd + InputA$ happens if the input is odd.
   - $sumOdd \Leftarrow sumOdd$ happens if the input is even (and **`go`** is asserted).
   - $sumOdd \Leftarrow InputA$ will happen when **`go`** is first asserted and the input is odd.
   - $sumOdd \Leftarrow 0$ also has to happen when **`go`** is first asserted, just in those cases where the input is even.

   (b) There needs to be a register to hold the value of **`sumOdd`**. Also, an adder to do the first transformation. And one (or more) multiplexers to choose which value goes into the register.

(c) The datapath schematic that I used is here. There are a few other ways to do it, for instance to use the clear on the register. Using two multiplexers would not be unusual.



(d) The FSM is fairly straightforward. I just need to track if **`go`** has already been asserted or not.




(e) You can find **`hw6prob6.sv`** on Canvas.