Hacettepe University Computer Science Department

BBM 233 Logic Design Laboratuvary

Verilog Project 1

Name Surname: Mert Çökelek

ID: 21727082

E-mail: b21727082@cs.hacettepe.edu.tr

Subject: Implementing an 8-bit comparator for two input numbers, which can be equal or not.

Due Date: 27.11.2018

Here are my verilog codes:

1- eightBitComparator.v

- I first defined a full adder. This is neccessary for using a 8-bit subtractor.
- The inputs of the adder are ain, bin and cin. Ain and bin are corresponding to the minuend and the subtrahend, respectively. And cin corresponds to the carry of the last addition problem.
- The outputs of the full adder module, are corresponding to carry and sum.
- By XOR'ing the 3 variables ain, bin, cin, we get sum output.
- By making (ain and bin) or (bin and cin) or (ain and cin) operation, we can get the carry.

- In binary system, to subtract one number from another, we can instead add the 2's complement of the second number to the first number easily. And this can be done by xor'ing the second number(bin, in this case) with one to get its complement.
- (Mathematically, ain – bin == ain + (bin xor 1))
- In the second modyle "eight bit subtractor", we should use the fulladder 8 times to operate an 8-bit operation. So, we need to get inputs as arrays with size 8, holding 1's and 0's.

The structure of an eight bit comparator is:

1- output cout: for the most significant bit, the last carry of the operation.

2- output s: This is holding sum variable. It should correspond to an 8-bit number, so it was declared as an array with size 8.

3- output leq: This is for comparing the 2 input numbers. If the first number is found to be less than or equal to the second number after the result of the subtracting operation, leq should be 1.

3.1 How to find “leq”:

- To determine leq is 1 or 0, we should check cout. If cout is 1, leq is 1, then the subtraction result is negative, then ain < bin.

4- output zero: This variable is equal to 1 if the result of the subtraction is 0, which means the inputs ain and bin are equal.

4.4 How to find “zero”:

- To determine zero is 1 or 0, we should look at every single bit of the sum.
    - Since sum is an array, we can take its bits by indexing. And after taking all the bits, we put them into a NOR gate to get a result. If s[1] nor s[2] nor.... nor s[7] is 1, that means s[1], s[2], …. s[6] are all 0’s. Because if at least one of them is 1, then the OR of them should be 1 and NOR should be 0.

5. The inputs of the subtractor are corresponding to a, b and s, corresponding to the 2 inputs and the sum variable, respectively.

- In this module, I declared a wire named bin, which is an 8 bit number.
- This is used for taking the XOR’ed value of the second input b.
- Every single bit of b is taken 2’s complement to make a subtraction easily with an adder.

- And there is a carry wire for holding the carry values after every single bit is added to each other. This is the output of one adder, and an input for the second adder... goes on like this until getting to the 8th bit.
- Finally, I used the compare module to initialize zero and leq variables.

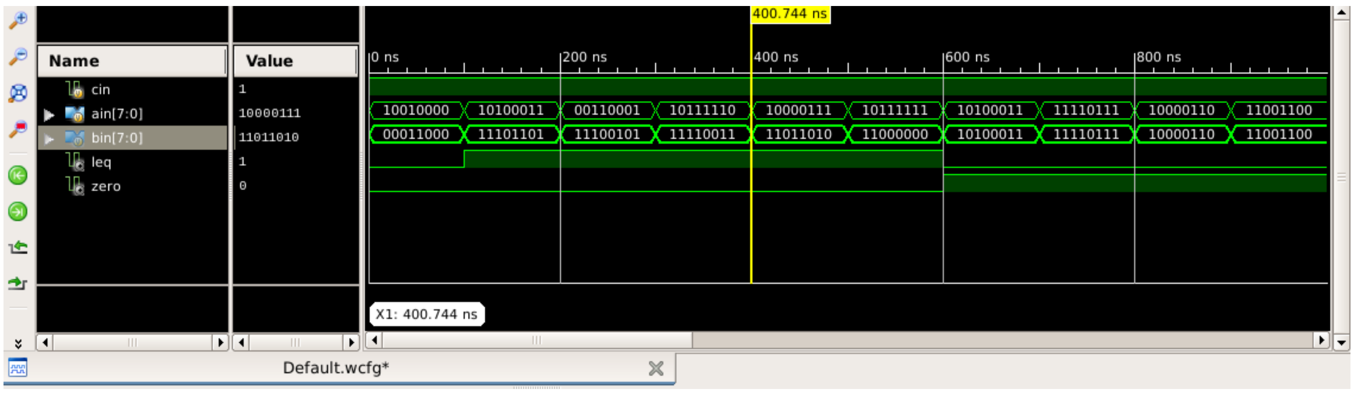
2- testbench.v

- In my testbench, I defined 2 + 1 inputs as mentioned in the pdf, corresponding to the 2 numbers which are going to be subtracted from each other, and the carryin value, respectively.
- The outputs of the testbench are zero and leq, they were mentioned in the previous part.
-
- After defining inputs and outputs, I should instantiate them for the UUT to show at the waveform.

- To give values to inputs, in the “initial begin” block, I tried 3 cases for (a > b, a < b and a = b)
- They are given random values, and cin is always taken 1 because we are making a subtraction operation. (If I was to use to adder for adding two numbers instead of subtracting them, I would give cin = 0.)
- And after each case, I waited for 100 ns to see the waveform clearly.

End.

And the output as a waveform:



```
Name        Value
cin         1
ain[7:0]    10000111
bin[7:0]    11011010
leq         1
zero        0
```

```
0 ns | 200 ns | 400 ns | 600 ns | 800 ns
ain: 10010000  10100011  00110001  10111110  10000111  10111111  10100011  11110111  10000110  11001100
bin: 00011000  11101101  11100101  11110011  11011010  11000000  10100011  11110111  10000110  11001100
```