Lab 3 Report

Nathaniel Fargo
10/2/2024
ECE 3700

Introduction

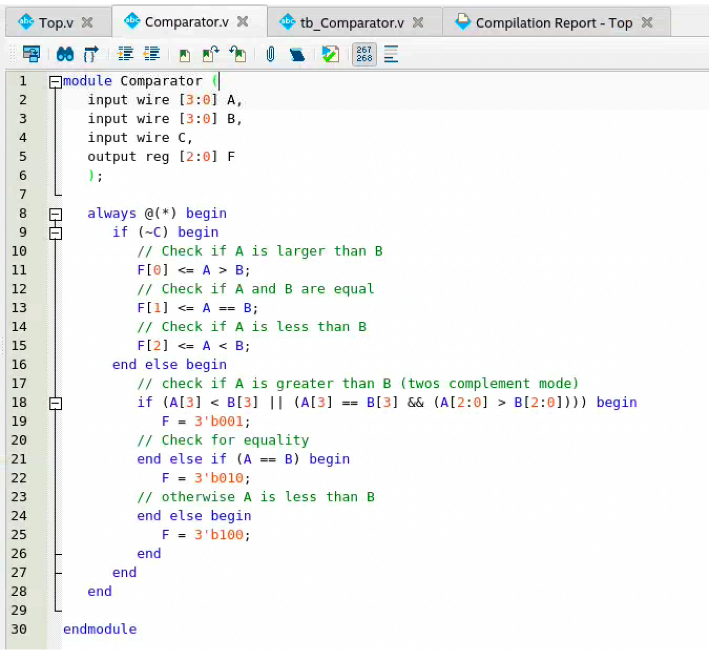
In this lab we're implementing a basic logic circuit that serves as a comparator. This module will take in two different 4-bit inputs and output a signal based on whether those two 4-bit words are equal, or whether the first one is greater than or less than the second one. Additionally, based on a third input, the circuit will switch between treating them as signed or unsigned integers. The following names are given to these signals. A[3:0] and B[3:0] are the four bit inputs. F[2:0] represent the comparison output. F[0] will be high if A > B, F[1] will be high if A is equal to B, and F[2] will be high if A < B. Finally "C" represents the control input to determine whether A and B are treated as unsigned or signed numbers.

Implementation

For the implementation of this circuit in behavioral verilog I'm going to be using the verilog $signed operator which treats binary arrays, e.g. A[3:0] as a signed decimal using two's complement form. This greatly simplifies the logic when C is one, e.g. the signed version.

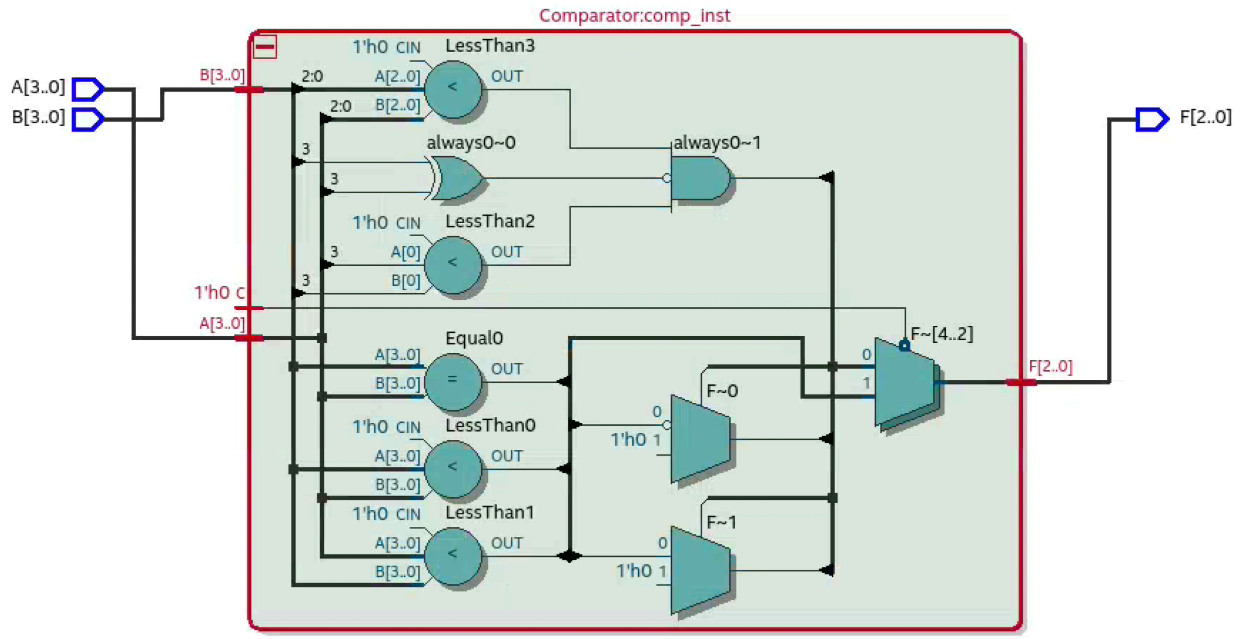
Here's the Comparator code. All of the behavioral logic gets set inside an always block, which allows if statements to be used. After checking for whether things should be treated as signed or unsigned, we then set each output of F as the output of the comparison of the vectors A and B.

This code uses combinational logic, since it bases it's outputs on the current states, as compared to sequential logic, which may set values based on the current inputs as well as the previous states of a module.



```
module Comparator (
   input wire [3:0] A,
   input wire [3:0] B,
   input wire C,
   output reg [2:0] F
   );

   always @(*) begin
      if (~C) begin
         // Check if A is larger than B
         F[0] <= A > B;
         // Check if A and B are equal
         F[1] <= A == B;
         // Check if A is less than B
         F[2] <= A < B;
      end else begin
         // check if A is greater than B (twos complement mode)
         if (A[3] < B[3] || (A[3] == B[3] && (A[2:0] > B[2:0]))) begin
            F = 3'b001;
         // Check for equality
         end else if (A == B) begin
            F = 3'b010;
         // otherwise A is less than B
         end else begin
            F = 3'b100;
         end
      end
   end

endmodule
```

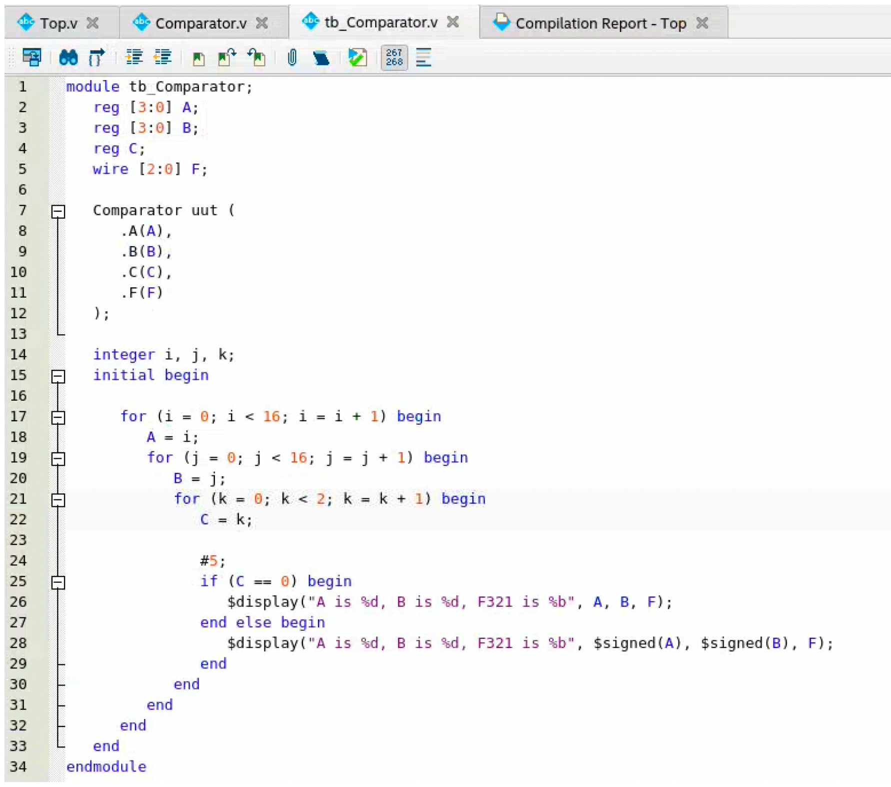
After running synthesis and implementation, we can view the RTL setlist and observe how the logic flows and check for latches. We observe no latches, and see that all the logic is implemented as expected.



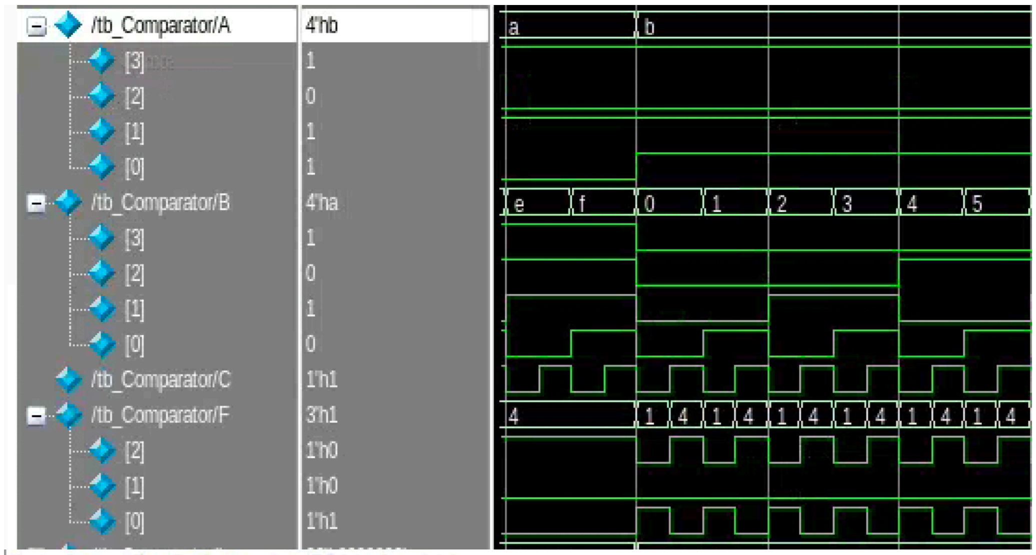
Simulation

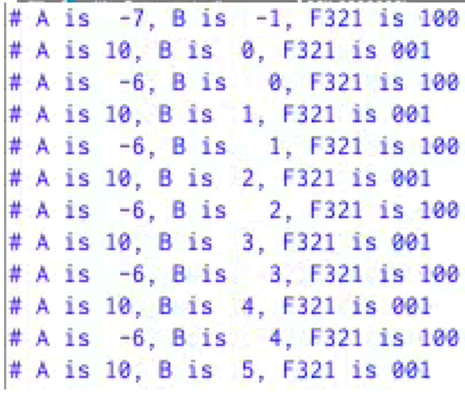
Next we run simulations of all of this code to see how it performs. Essentially this code tests every combination of A, B, and C and outputs the value of F. It also outputs A and B as signed integers when in twos complement mode.

Below are the console and waveform outputs of this code.



```
module tb_Comparator;
   reg [3:0] A;
   reg [3:0] B;
   reg C;
   wire [2:0] F;

   Comparator uut (
      .A(A),
      .B(B),
      .C(C),
      .F(F)
   );

   integer i, j, k;
   initial begin

      for (i = 0; i < 16; i = i + 1) begin
         A = i;
         for (j = 0; j < 16; j = j + 1) begin
            B = j;
            for (k = 0; k < 2; k = k + 1) begin
               C = k;

               #5;
               if (C == 0) begin
                  $display("A is %d, B is %d, F321 is %b", A, B, F);
               end else begin
                  $display("A is %d, B is %d, F321 is %b", $signed(A), $signed(B), F);
               end
            end
         end
      end
   end
endmodule
```




```
# A is  -7, B is  -1, F321 is 100
# A is 10, B is  0, F321 is 001
# A is  -6, B is   0, F321 is 100
# A is 10, B is  1, F321 is 001
# A is  -6, B is   1, F321 is 100
# A is 10, B is  2, F321 is 001
# A is  -6, B is   2, F321 is 100
# A is 10, B is  3, F321 is 001
# A is  -6, B is   3, F321 is 100
# A is 10, B is  4, F321 is 001
# A is  -6, B is   4, F321 is 100
# A is 10, B is  5, F321 is 001
```

After reviewing the console and waveform outputs, all of them seem to be what we expect for this module.

Conclusion

In this lab we've implemented a basic comparator circuit using always blocks and combinational logic in verilog. This code compares two 4-bit inputs and treats them either as unsigned words or signed (two's complements) inputs based on a third input, labeled C. We decode this into a one-hot output F which is a 3 bit output with just one high based on whether A is greater than, equal to, or less than B.