ELECTRICAL ENGINEERING

Building an 8-bit adder/subtractor using two 4-bit adder/subtractors and the result has a difference of 16 for values above 16

Asked 11 months ago Modified 20 days ago Viewed 492 times

0

I'm trying to build an 8-bit adder/subtractor using two 4-bit adder/subtractors The result is always 16 below the required value when adding two numbers above 16 and 16 above the required value when subtracting. I have connected the overflow of the first instance to the Carry in of the second instant but it is still not working. An if statement could work but I want it to be in the structural model. Can you help me figure out why this happens?

```
module full_adder(S, Cout, A, B, Cin);
   output S;
   output Cout;
   input  A;
   input  B;
   input  Cin;

   wire   w1;
   wire   w2;
   wire   w3;
   wire   w4;

   xor(w1, A, B);
   xor(S, Cin, w1);
   and(w2, A, B);
   and(w3, A, Cin);
   and(w4, B, Cin);
   or(Cout, w2, w3, w4);
endmodule

module ripple_carry_adder_subtractor(S,C,V, A, B, Op);
   output [3:0] S;
   output   C;
   output   V;
   input [3:0]  A;
   input [3:0]  B;
   input Op;

   wire     C0;
   wire     C1;
   wire     C2;
   wire     C3;

   wire     B0;
   wire     B1;
   wire     B2;
   wire     B3;



  xor(B0,Op, B[0]);
  xor(B1,Op, B[1]);
```

```
  xor(B2,Op, B[2]);
  xor(B3,Op, B[3]);
  xor(C,Op, C3);
   xor(V, C3, C2);

   full_adder fa0(S[0], C0, A[0], B0, Op);
  full_adder fa1(S[1], C1, A[1], B1, C0);
  full_adder fa2(S[2], C2, A[2], B2, C1);
   full_adder fa3(S[3], C3, A[3], B3, C2);
endmodule

module adder8(S, A, B, Op);
   output [7:0] S;
   input [7:0]  A;
   input [7:0]  B;
   input Op;
  wire Cout, Cout2, Vout,Vout2;
   ripple_carry_adder_subtractor r1(S[3:0],Cout, Vout,A[3:0],B[3:0],Op);
   ripple_carry_adder_subtractor r2(S[7:4],Vout, Vout2,A[7:4],B[7:4],Op);

  endmodule



module test;
  reg[7:0] x,y;
  reg operation;
  wire [7:0] o;
  adder8 r1(o,x,y,operation);


initial
begin
  #1 $display ("                time ", "      A   ","     B   ","result" , "
M");
 x=17; y=2; operation =1;
 #1 $display ($time,"       ",x,"       ", y, "       ",o,"       ",operation);
 x=150  ; y=120; operation =1;
#1 $display ($time,"       ",x,"       ", y, "       ",o,"       ",operation);
x=15; y=2; operation =0;
  #1 $display ($time,"       ",x,"       ", y, "       ",o,"       ",operation);
 x=0; y=1; operation =1;
  #1 $display ($time,"       ",x,"       ", y, "       ",o,"       ",operation);

end

endmodule
```

digital-logic verilog adder

Share Cite Follow

asked Jul 11, 2021 at 13:58

M.I.S.A.
1 1

1 Answer

Sorted by:

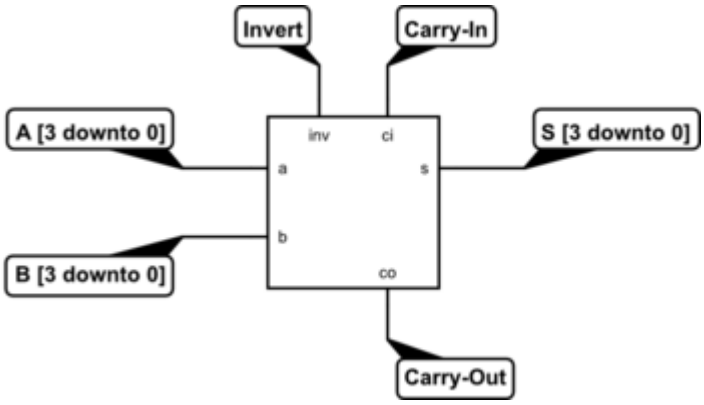
Highest score (default)

The *operation* should be fed in as the carry-in for one 4-bit module and not the other. The

0

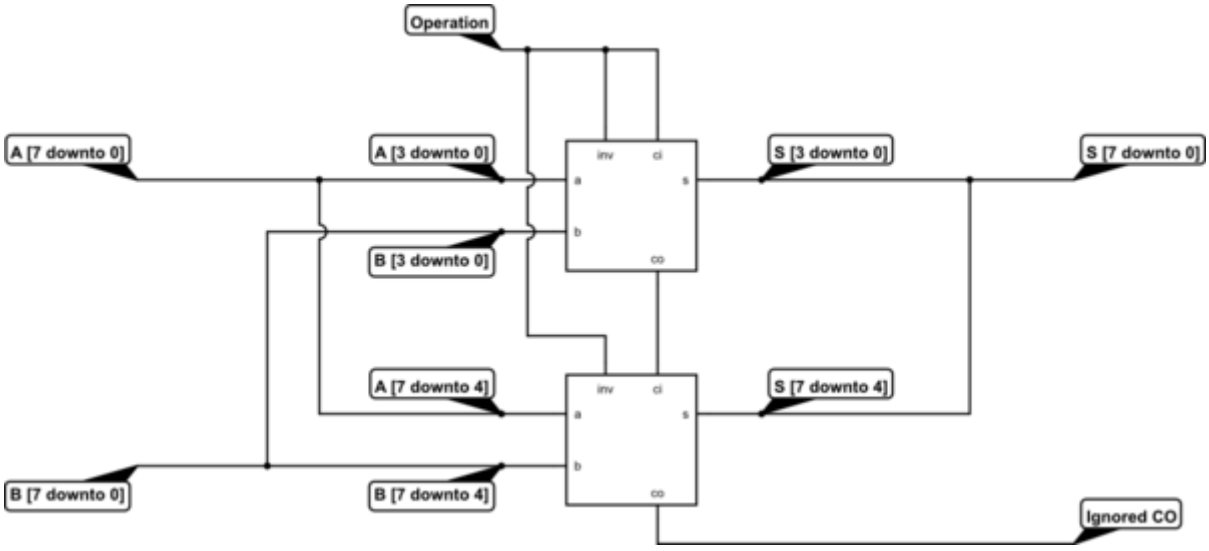
other one should get its carry/borrow from the prior module. The *operation*, though, should be fed to both as the invert line to the XOR group. It doesn't appear to me that you have a way to ***separate*** these ideas. You appear to imagine that the operation serves both purposes in both cases. It doesn't.

For example, you should have a 4-bit module with a one-bit *carry-in* input, a one-bit *invert* input, a 4-bit *a* input, a 4-bit *b* input, a 4-bit *s* output, and a one-bit *carry-out* output:



simulate this circuit – Schematic created using CircuitLab

This module should then be combined like this:



simulate this circuit

In your code:

```
xor(B0,Op, B[0]);
xor(B1,Op, B[1]);
xor(B2,Op, B[2]);
xor(B3,Op, B[3]);
xor(C,Op, C3);
```

In your incorrect arrangement, you ***always*** XOR the *Op* with *C*. But in the higher order 4-bit add/sub module, it needs to get the carry/borrow from the prior module and *Op* should have no impact on it.

Does that make sense?

Share Cite Follow

answered Jul 11, 2021 at 18:07

jonk
69.6k 4 64 160