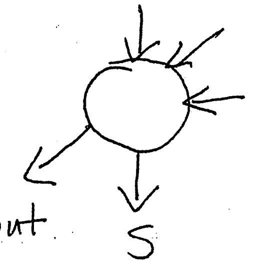
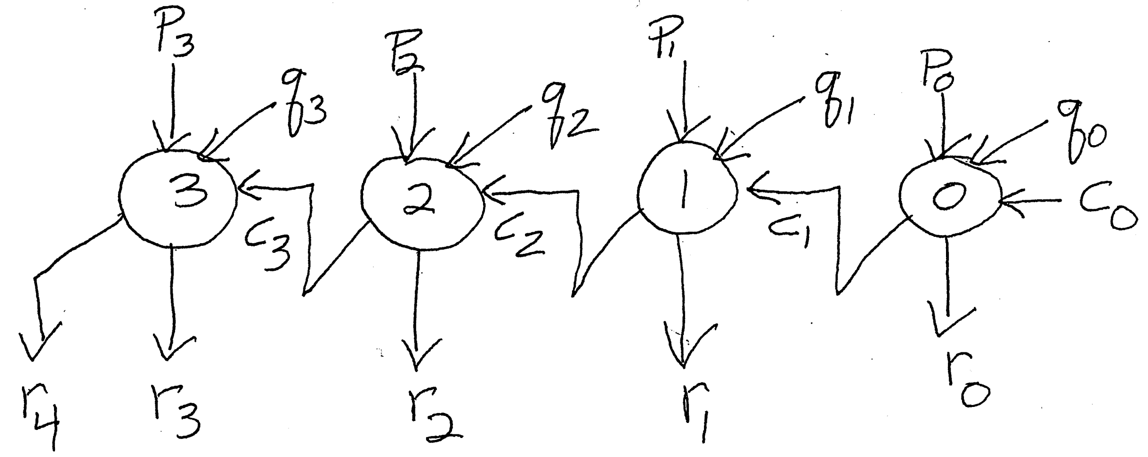
full adder:



RCA:



```
module full_adder(a, b, cin, s, cout);

input  a, b, cin;
output s, cout;

assign s = a^b^cin;
assign cout = (a&b) | (b&cin) | (a&cin);

endmodule

module RCA(p, q, c0, r);

input  [3:0] p, q;
input  c0;
output [4:0] r;

wire   c1, c2, c3;

full_adder fa0(p[0], q[0], c0, r[0], c1);
full_adder fa1(p[1], q[1], c1, r[1], c2);
full_adder fa2(p[2], q[2], c2, r[2], c3);
full_adder fa3(p[3], q[3], c3, r[3], r[4]);

endmodule
```

$\boxed{\text{file ripple.v}}$

```
module tb1; // specific input sequence, binary output

reg  [3:0] a, b;
reg  c;
wire [4:0] s;

// instantiate the 4-bit ripple-carry adder
RCA rca1(a, b, c, s);

// apply a specific set of input vectors every 10 time units
initial begin
    a = 4'b0010; b = 4'b1010; c = 1'b0;
#10 a = 4'b1111; b = 4'b0000; c = 1'b0;
#10 a = 4'b0001; b = 4'b0001; c = 1'b1;
end

// print the input and output values after they change
initial
$monitor($time, "  a = %b", a, " b = %b", b, " c = %b", c, " s = %b", s);

endmodule
```

```
verilog ripple.v tb1.v


 0  a = 0010 b = 1010 c = 0 s = 01100
10  a = 1111 b = 0000 c = 0 s = 01111
20  a = 0001 b = 0001 c = 1 s = 00011
```

file tb1.v

```
module tb3; // random inputs, decimal values including a check

reg  [3:0] a, b;  // 4-bit inputs (to be chosen randomly)
reg  c;           // carry input  (to be chosen randomly)
wire [4:0] s;     // 5-bit output of the RCA circuit
reg  [4:0] check; // 5-bit sum value used to check correctess

// instantiate the 4-bit ripple-carry adder
RCA rca1(a, b, c, s);

// simulation of 20 random addition operations
initial repeat (20) begin
    // get new operand values and compute a check value
    a = $random; b = $random; c = $random;
    check = a + b + c;

    // compute and display the sum every 10 time units
    #10 $display($time, "  %d + %d + %d = %d (%d)", a, b, c, s, check);
end

endmodule
```

file tb3.v

```
                verilog ripple.v tb3.v

                      10    4 +  1 + 1 =  6 ( 6)
                      20    3 + 13 + 1 = 17 (17)
                      30    5 +  2 + 1 =  8 ( 8)
                      40   13 +  6 + 1 = 20 (20)
                      50   13 + 12 + 1 = 26 (26)
                      60    6 +  5 + 0 = 11 (11)
                      70    5 +  7 + 0 = 12 (12)
                      80   15 +  2 + 0 = 17 (17)
                      90    8 +  5 + 0 = 13 (13)
                     100   13 + 13 + 1 = 27 (27)
                     110    3 + 10 + 0 = 13 (13)
                     120    0 + 10 + 1 = 11 (11)
                     130    6 +  3 + 1 = 10 (10)
                     140    3 + 11 + 1 = 15 (15)
                     150    2 + 14 + 1 = 17 (17)
                     160   15 +  3 + 0 = 18 (18)
                     170   10 + 12 + 0 = 22 (22)
                     180   10 +  1 + 0 = 11 (11)
                     190    8 +  9 + 1 = 18 (18)
                     200    6 +  6 + 0 = 12 (12)
```

```
module tb5; // testbench for signed addition and subtraction

reg  [3:0] a, b;         // 4-bit inputs (to be chosen randomly)
integer    aval, bval;   // numerical values of inputs a and b
reg  c;                  // carry input  (to be used for subtraction)
wire [4:0] s;            // 5-bit output of the RCA circuit
integer    sval, dval;   // numerical values of the sum and difference
integer    sum_check;    // value used to check correctness of an addition
integer    dif_check;    // value used to check correctness of a subtraction

// instantiate the 4-bit ripple-carry adder
RCA rca1(a, b, c, s);

// simulation of 10 additions and 10 subtractions using random operands
initial repeat (10) begin
    // get new operand values and compute the two check values
    a = $random; b = $random; c = 0;
    aval = -a[3]*8 + a[2:0];
    bval = -b[3]*8 + b[2:0];
    sum_check = aval + bval;
    dif_check = aval - bval;

    // compute and display the sum with its check value
    #10 sval = -s[3]*8 + s[2:0];
    $display($time, "  %d + %d = %d (%d)", aval, bval, sval, sum_check);

    // compute and display the difference with its check value
    b = ~b; c = 1; // one's complement of b plus 1 into the LSB
    #10 dval = -s[3]*8 + s[2:0];
    $display($time, "  %d - %d = %d (%d)", aval, bval, dval, dif_check);
end

endmodule
```

$\boxed{\text{file tb5.v}}$

$$\left(aval = -a_3 2^3 + \sum_{i=0}^{2} a_i 2^i \right)$$

```
verilog ripple.v tb5.v

                  10          4 +          1 =          5 (          5)
                  20          4 -          1 =          3 (          3)
                  30         -7 +          3 =         -4 (         -4)
                  40         -7 -          3 =          6 (        -10) ←
                  50         -3 +         -3 =         -6 (         -6)
                  60         -3 -         -3 =          0 (          0)
                  70          5 +          2 =          7 (          7)
                  80          5 -          2 =          3 (          3)
                  90          1 +         -3 =         -2 (         -2)
                 100          1 -         -3 =          4 (          4)
                 110          6 +         -3 =          3 (          3)
                 120          6 -         -3 =         -7 (          9) ←
                 130         -3 +         -4 =         -7 (         -7)
                 140         -3 -         -4 =          1 (          1)
                 150         -7 +          6 =         -1 (         -1)
                 160         -7 -          6 =          3 (        -13) ←
                 170          5 +         -6 =         -1 (         -1)
                 180          5 -         -6 =         -5 (         11) ←
                 190          5 +          7 =         -4 (         12) ←
                 200          5 -          7 =         -2 (         -2)
```

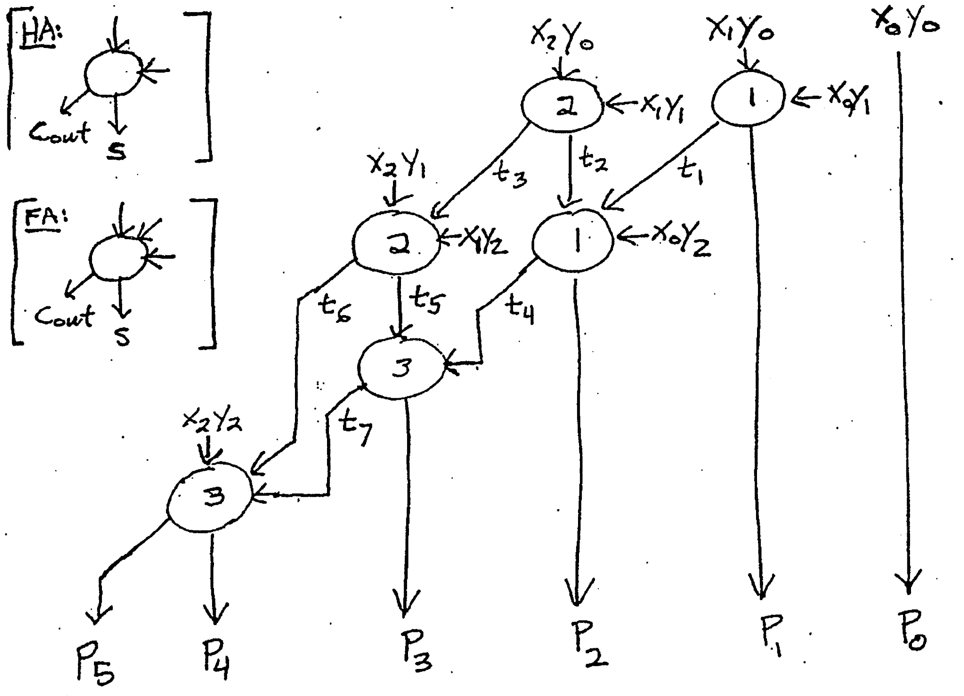
(arrows show instances of signed overflow)

```
// half adder component used in the multiplier
module half_adder(a, b, s, cout);

input  a, b;
output s, cout;

assign s = a^b;
assign cout = a&b;

endmodule

// full adder component used in the multiplier
module full_adder(a, b, cin, s, cout);

input  a, b, cin;
output s, cout;

assign s = a^b^cin;
assign cout = (a&b) | (b&cin) | (a&cin);

endmodule

// 3-bit by 3-bit unsigned multiplier
module mult3(x, y, p);

input  [2:0] x, y;
output [5:0] p;

// internal nodes within the multiplier circuit
wire   t1, t2, t3, t4, t5, t6, t7;

// structural description of the multiplier circuit
assign p[0] = x[0]&y[0];
half_adder ha1(x[1]&y[0], x[0]&y[1], p[1], t1);
half_adder ha2(x[2]&y[0], x[1]&y[1], t2, t3);
full_adder fa1(t2, t1, x[0]&y[2], p[2], t4);
full_adder fa2(x[2]&y[1], t3, x[1]&y[2], t5, t6);
half_adder ha3(t5, t4, p[3], t7);
full_adder fa3(x[2]&y[2], t6, t7, p[4], p[5]);

endmodule
```

$\boxed{\text{file mult3.v}}$

$$\begin{array}{cccccc}
 & & & x_2 & x_1 & x_0 \\
 & & & y_2 & y_1 & y_0 \\
\hline
 & & & x_2y_0 & x_1y_0 & x_0y_0 \\
 & & x_2y_1 & x_1y_1 & x_0y_1 & \\
 & x_2y_2 & x_1y_2 & x_0y_2 & & \\
\hline
P_5 & P_4 & P_3 & P_2 & P_1 & P_0
\end{array}$$

3-bit by 3-bit unsigned multiplier



file tb7.v

```
module tb7; // testbench for the 3-bit by 3-bit multiplier
            // random inputs, decimal values including a check

reg  [2:0] x, y;  // 3-bit inputs (to be chosen randomly)
wire [5:0] p;     // 6-bit output of the multiplier circuit
reg  [5:0] check; // 6-bit product value used to check correctess

// instantiate the 3-bit by 3-bit multiplier
mult3 mult_instance(x, y, p);

// simulation of 20 random multiplication operations
initial repeat (20) begin
    // get new operand values and compute a check value
    x = $random; y = $random;
    check = x * y;

    // compute and display the product every 10 time units
    #10 $display($time, "  %d * %d = %d (%d)", x, y, p, check);
end

endmodule
```

```
verilog mult3.v tb7.v

         10  4 * 1 =  4 ( 4)
         20  1 * 3 =  3 ( 3)
         30  5 * 5 = 25 (25)
         40  5 * 2 = 10 (10)
         50  1 * 5 =  5 ( 5)
         60  6 * 5 = 30 (30)
         70  5 * 4 = 20 (20)
         80  1 * 6 =  6 ( 6)
         90  5 * 2 = 10 (10)
        100  5 * 7 = 35 (35)
        110  2 * 7 = 14 (14)
        120  2 * 6 = 12 (12)
        130  0 * 5 =  0 ( 0)
        140  4 * 5 = 20 (20)
        150  5 * 5 = 25 (25)
        160  3 * 2 =  6 ( 6)
        170  0 * 0 =  0 ( 0)
        180  2 * 5 = 10 (10)
        190  6 * 3 = 18 (18)
        200  5 * 3 = 15 (15)
```

```
module tbe; // testbench for the 3-bit by 3-bit unsigned multiplier
            // exhaustive checking of all 64 possible cases

reg  [2:0] x, y;  // 3-bit inputs
wire [5:0] p;     // 6-bit output of the multiplier circuit
reg  [5:0] check; // 6-bit product value used to check correctess
integer    i, j;        // loop variables
integer    num_correct; // counter to keep track of the number correct
integer    num_wrong;   // counter to keep track of the number wrong

// instantiate the 3-bit by 3-bit multiplier
mult3 mult_instance(x, y, p);

// exhaustive checking of all 64 possible cases
initial begin
    // initialize the counter variables
    num_correct = 0; num_wrong = 0;

    // loop through all possible cases and record the results
    for (i = 0; i < 8; i = i + 1) begin
        x = i;
        for (j = 0; j < 8; j = j + 1) begin
            y = j;
            check = x * y;

            // compute and check the product
            #10 if (p == check)
                    num_correct = num_correct + 1;
                else
                    num_wrong = num_wrong + 1;

            // following line is commented out, but is useful for debugging
            // $display($time, "  %d * %d = %d (%d)", x, y, p, check);
        end
    end

    // print the final counter values
    $display("num_correct = %d, num_wrong = %d", num_correct, num_wrong);

end

endmodule
```

file tbe.v

```
verilog mult3.v tbe.v

                num_correct =          64, num_wrong =           0
```