# Hadd

Create a half adder. A half adder adds two bits (with no carry-in) and produces a sum and carry-out.

```
module top_module(
    input a, b,
    output cout, sum );
    assign {cout,sum} = a+b;

endmodule
```

# Fadd

Create a full adder. A full adder adds three bits (including carry-in) and produces a sum and carry-out.

```
module top_module(
    input a, b, cin,
    output cout, sum );

    assign{cout,sum}=a+b+cin;

endmodule
```

# Adder3

Now that you know how to build a full adder, make 3 instances of it to create a 3-bit binary ripple-carry adder. The adder adds two 3-bit numbers and a carry-in to produce a 3-bit sum and carry out. To encourage you to actually instantiate full adders, also output the carry-out from *each* full adder in the ripple-carry adder. cout[2] is the final carry-out from the last full adder, and is the carry-out you usually see.

```
module top_module(
    input [2:0] a, b,
    input cin,
    output [2:0] cout,
    output [2:0] sum );

    wire carry0, carry1, carry2;
    wire sum0, sum1, sum2;
    assign {carry0,sum0}=a[0]+b[0]+cin;
    assign {carry1,sum1}=a[1]+b[1]+carry0;
```

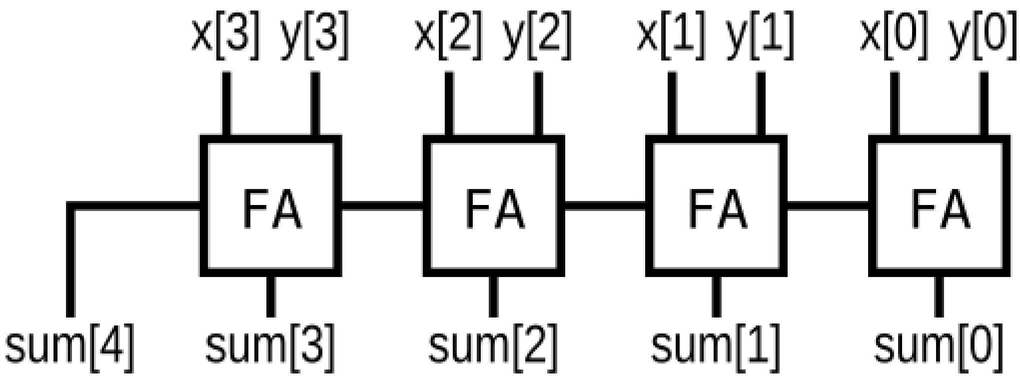
```
    assign {carry2,sum2}=a[2]+b[2]+carry1;

    assign sum = {sum2,sum1,sum0};
    assign cout = {carry2,carry1,carry0};


endmodule
```

# Exams/m2014 q4j

Implement the following circuit:




("FA" is a full adder)

```
module top_module (
    input [3:0] x,
    input [3:0] y,
    output [4:0] sum);

    assign sum = {1'b0,x}+{1'b0,y};
endmodule
```

# Exams/ece241 2014 q1c

Assume that you have two 8-bit 2's complement numbers, a[7:0] and b[7:0]. These numbers are added to produce s[7:0]. Also compute whether a (signed) overflow has occurred.

A *signed* overflow occurs when adding two positive numbers produces a negative result, or adding two negative numbers produces a positive result. There are several methods to detect overflow: It could be computed by comparing the signs of the input and output numbers, or derived from the carry-out of bit n and n-1.

```
module top_module (
    input [7:0] a,
    input [7:0] b,
    output [7:0] s,
    output overflow
); //

    assign s = a+b;
                //pos 0  neg1  : adding two positive numbers but  produces a negative outcome
                //               adding two negative numbers but  produces a positive outcome
                //          0     0    1        1       1      0
    assign overflow = ~a[7]&~b[7]&s[7] | a[7]&b[7]&~s[7];

endmodule
```

# Adder100

Create a 100-bit binary adder. The adder adds two 100-bit numbers and a carry-in to produce a 100-bit sum and carry out.

```
module top_module(
    input [99:0] a, b,
    input cin,
    output cout,
    output [99:0] sum );
    assign {cout,sum} = a+b+cin;

endmodule
```

# Bcdadd4

You are provided with a BCD (binary-coded decimal) one-digit adder named `bcd_fadd` that adds two BCD digits and carry-in, and produces a sum and carry-out.

```
module bcd_fadd {
    input [3:0] a,
    input [3:0] b,
    input     cin,
    output   cout,
    output [3:0] sum );
```

Instantiate 4 copies of `bcd_fadd` to create a 4-digit BCD ripple-carry adder. Your adder should add two 4-digit BCD numbers (packed into 16-bit vectors) and a carry-in to produce a 4-digit sum and carry out.

- The BCD representation for the 5-digit decimal number `12345` is `20'h12345`. This is not the same as `14'd12345` (which is `14'h3039`).
- The circuit is structured just like a binary ripple-carry adder, except the adders are base-10 rather than base-2.

```
module top_module(
   input [15:0] a, b,
   input cin,
   output cout,
   output [15:0] sum );

   wire c0,c1,c2;

   bcd_fadd add0(a[ 3: 0],b[ 3: 0], cin,   c0,sum[ 3: 0]);
   bcd_fadd add1(a[ 7: 4],b[ 7: 4],  c0,   c1,sum[ 7: 4]);
   bcd_fadd add2(a[11: 8],b[11: 8],  c1,   c2,sum[11:08]);
   bcd_fadd add3(a[15:12],b[15:12],  c2, cout,sum[15:12]);
endmodule
```