Combinational Circuits:

Half Adder

```
module HalfAdder(
    input A,
    input B,
    output S,
    output Cout
    );
    assign S = A^B;
    assign Cout = A&B;
endmodule
```

```
module test1;
reg A;
reg B;
wire S;
wire Cout;
HalfAdder uut (
.A(A),
.B(B),
.S(S),
.Cout(Cout));
initial begin
A = 0; B = 0; #100;
A = 1; B = 0; #100;
A = 1; B = 1; #100;
A = 0; B = 1; #100;
end
endmodule
```

Full Adder

```
module FullAdder(
    input A,
    input B,
    input Cin,
    output S,
    output Cout
    );
    HalfAdder inst1(.A(A),.B(B),.S(b),.Cout(p));
    HalfAdder inst2(.A(Cin),.B(b),.S(S),.Cout(q));
    assign Cout = p|q;
endmodule
```

```
module test1;
reg A;
reg B;
reg Cin;
wire S;
wire Cout;
FullAdder uut (
.A(A),
.B(B),
.Cin(Cin),
.S(S),
.Cout(Cout)
);
initial begin
A = 0; B = 0; Cin = 0;
#100;
A = 1; B = 0; Cin = 0;
#100;
A = 1; B = 1; Cin = 1;
#100;
A = 0; B = 1; Cin = 1;
#100;
end
endmodule
```

Decoder 3x8

```
module Decoder(a,b,c,d0,d1,d2,d3,d4,d5,d6,d7);
input a,b,c;
output d0,d1,d2,d3,d4,d5,d6,d7;
assign d0=(~a&~b&~c),
d1=(~a&~b&c),
d2=(~a&b&~c),
d3=(~a&b&c),
d4=(a&~b&~c),
d5=(a&~b&c),
d6=(a&b&~c),
d7=(a&b&c);
endmodule
```

```
module TestModule;
reg a;reg b;reg c;
wire d0;wire d1;wire d2;wire d3;
wire d4;wire d5;wire d6;wire d7;
Decoder uut (
.a(a),.b(b),.c(c),.d0(d0),.d1(d1),.d2(d2),
.d3(d3),.d4(d4),.d5(d5),.d6(d6),.d7(d7)
);
initial begin
a = 0; b = 0; c = 0; #100;
a = 1; b = 0; c = 1; #100;
end
endmodule
```

Magnitude Comparator

```
module comparator(
    input [7:0] numA,
    input [7:0] numB,
    output isEqual,
    output isGreater
    );
    assign isEqual=(numA==numB);
    assign isGreater=(numA>numB);
endmodule
```

```
module test;
// Inputs
reg [7:0] numA;
reg [7:0] numB;
// Outputs
wire isEqual;
wire isGreater;
// Instantiate the Unit Under Test (UUT)
comparator uut (
.numA(numA),
.numB(numB),
.isGreater(isGreater),
.isEqual(isEqual)
);
initial begin
#100 numA=8'b10000000; numB=8'b10000000;
#100 numA=8'b00000001; numB=8'b00000000;
#100 numA=8'b00000000; numB=8'b10000000;
end
endmodule
```

Demux 8x1

```
module Demultiplexer(in,s0,s1,s2,d0,d1,d2,d3,d4,d5,d6,d7);
input in,s0,s1,s2;
output d0,d1,d2,d3,d4,d5,d6,d7;
assign d0=(in & ~s2 & ~s1 &~s0),
d1=(in & ~s2 & ~s1 &s0),
d2=(in & ~s2 & s1 &~s0),
d3=(in & ~s2 & s1 &s0),
d4=(in & s2 & ~s1 &~s0),
d5=(in & s2 & ~s1 &s0),
d6=(in & s2 & s1 &~s0),
d7=(in & s2 & s1 &s0);
endmodule
```

```
module TestModule;
// Inputs
reg in;
reg s0;
reg s1;
reg s2;
// Outputs
wire d0;wire d1;wire d2;wire d3;
wire d4;wire d5;wire d6;wire d7;
// Instantiate the Unit Under Test (UUT)
Demultiplexer uut (
.in(in),.s0(s0),.s1(s1),.s2(s2),
.d0(d0),.d1(d1),.d2(d2),.d3(d3),
.d4(d4),.d5(d5),.d6(d6),.d7(d7)
);
initial begin
in = 0;s0 = 0;s1 = 0;s2 = 0;#100;
in = 1;s0 = 0;s1 = 1;s2 = 0;#100;
end
endmodule
```

Encoder 8x3

```
module Encoder(d0,d1,d2,d3,d4,d5,d6,d7,a,b,c);
input d0,d1,d2,d3,d4,d5,d6,d7;
output a,b,c;
or(a,d4,d5,d6,d7);
or(b,d2,d3,d6,d7);
or(c,d1,d3,d5,d7);
endmodule
```

```
module TestModule;
// Inputs
reg d0;reg d1;reg d2;reg d3;
reg d4;reg d5;reg d6;reg d7;
// Outputs
wire a;wire b;wire c;
// Instantiate the Unit Under
Encoder uut (
.d0(d0),.d1(d1),
.d2(d2),.d3(d3),
.d4(d4),.d5(d5),
.d6(d6),.d7(d7),
.a(a),.b(b),.c(c)
);
initial begin
d0 = 0;d1 = 0;d2 = 0;d3 = 0;
d4 = 0;d5 = 0;d6 = 0;d7 = 0;
#100;
d0 = 0;d1 = 0;d2 = 0;d3 = 1;
d4 = 0;d5 = 0;d6 = 0;d7 = 0;
#100;
end
endmodule
```

Carry Look Ahead Adder

```
module CLA_Adder(a,b,cin,sum,cout);
input[3:0] a,b;
input cin;
output [3:0] sum;
output cout;
wire p0,p1,p2,p3,g0,g1,g2,g3,c1,c2,c3,c4;
assign p0=(a[0]^b[0]),
p1=(a[1]^b[1]),
p2=(a[2]^b[2]),
p3=(a[3]^b[3]);
assign g0=(a[0]&b[0]),
g1=(a[1]&b[1]),
g2=(a[2]&b[2]),
g3=(a[3]&b[3]);
assign c0=cin,
c1=g0|(p0&cin),
c2=g1|(p1&g0)|(p1&p0&cin),
c3=g2|(p2&g1)|(p2&p1&g0)|(p1&p1&p0&cin),
c4=g3|(p3&g2)|(p3&p2&g1)|(p3&p2&p1&g0)|(p3&p2&p1&p0&cin);
assign sum[0]=p0^c0,
sum[1]=p1^c1,
sum[2]=p2^c2,
sum[3]=p3^c3;
assign cout=c4;
endmodule
```

```
module TestModule;
// Inputs
reg [3:0] a;
reg [3:0] b;
reg cin;
// Outputs
wire [3:0] sum;
wire cout;
// Instantiate the Unit U
CLA_Adder uut (
.a(a),.b(b),
.cin(cin),
.sum(sum),
.cout(cout)
);
initial begin
a = 0;b = 0;cin = 0;#100;
a = 5;b = 6;cin = 1;#100;
end
endmodule
```

4-bit CLA

```
module CLA(
    input [3:0] numA,
    input [3:0] numB,
    input ci,
    output reg co,
    output reg [3:0] sum
    );
    reg [3:0] G, P;
    always@(numA or numB or ci)
        begin
        sum = numA+numB+ci;
        G = numA&numB;
        P = numA^numB;
        co = G[3]|(P[3]&G[2])|(P[3]&P[2]&G[1])|
             (P[3]&P[2]&P[1]&G[0])|
             (P[3]&P[2]&P[1]&P[0]&ci);
        end
endmodule
```

```
module test;
// Inputs
reg [3:0] numA;
reg [3:0] numB;
reg ci;
// Outputs
wire co;
wire [3:0] sum;
// Instantiate the Unit Under Test (U
CLA uut (
.numA(numA),
.numB(numB),
.ci(ci),.co(co),
.sum(sum)
);
initial begin
#100 numA=4'b1000;numB=4'b1000;ci=0;
#100 numA=4'b1000;numB=4'b1000;ci=1;
#100 numA=4'b0001;numB=4'b0000;ci=1;
#100 numA=4'b0000;numB=4'b1000;ci=1;
#100 numA=4'b0011;numB=4'b1100;ci=0;
#100 numA=4'b0011;numB=4'b1100;ci=1;
end
endmodule
```

16-bit CLA

```
module CascadeCLA(
    input [15:0] numA,
    input [15:0] numB,
    input ci,
    output co,
    output [15:0] sum
    );
    CLA inst1(numA[3:0], numB[3:0], ci, co1, sum[3:0]);
    CLA inst2(numA[7:4], numB[7:4], co1, co2, sum[7:4]);
    CLA inst3(numA[11:8], numB[11:8], co2, co3, sum[11:8]);
    CLA inst4(numA[15:12], numB[15:12], co3, co, sum[15:12]);
endmodule
```

```
module test;
reg [15:0] numA;
reg [15:0] numB;
reg ci;
wire co;
wire [15:0] sum;
CascadeCLA uut (
.numA(numA),
.numB(numB),
.ci(ci),.co(co),
.sum(sum));
initial begin
#100 numA=16'b0000000000000000;
ci=0;numB=16'b1000000000000000;
#100 numA=16'b0000000000000000;
ci=1;numB=16'b1000000000000001;
#100 numA=16'b0000000011110000;
ci=1;numB=16'b0001100000010000;
#100 numA=16'b0000000000000100;
ci=1;numB=16'b0001100000000000;
#100 numA=16'b1000000000000000;
ci=0;numB=16'b1000000000000000;
#100 numA=16'b1000000000000000;
ci=1;numB=16'b1000000000000000;
end
endmodule
```

Sequential Circuits

Linear feedback shift register

```
module lfsr(
    input shiftLoadBar,
    input [7:0] seedValue,
    input Clk,
    input Rst,
    output reg lfsrDone,
    output reg [7:0] oSequence);
    initial
    begin
    oSequence<=seedValue;
    end
    always@(posedge Clk or posedge Rst)
        if(Rst) oSequence=seedValue;
        else
            begin
            if (shiftLoadBar) oSequence=seedValue;
            oSequence<=oSequence>>1;
            oSequence[7]<=oSequence[7]^oSequence[2]^oSequence[1]^oSequence[0];
            if(oSequence==seedValue) lfsrDone=1'b1;
            else lfsrDone=1'b0;
            end
endmodule
```

```
module test;
reg [7:0] seedValue;
reg shiftLoadBar;
reg Rst; reg Clk;
wire lfsrDone;
wire [7:0] oSequence;
// Instantiate the Unit Unde
lfsr uut (
.seedValue(seedValue),
.shiftLoadBar(shiftLoadBar),
.Rst(Rst),.Clk(Clk),
.lfsrDone(lfsrDone),
.oSequence(oSequence)
);
always #10Clk=!Clk;
initial begin
#10
Clk=0; Rst=0;
shiftLoadBar=0;
seedValue=6;
#10 Rst=1;
#10 Rst=0;
end
endmodule
```

Counter

```
module counter(
    input clk,
    input reset,
    output reg [3:0] count,
    output reg max_tick,
    reg [3:0] next_count
    );
always@(posedge clk, negedge reset)
begin
    if(reset==0)
        count<=0;
    else
        count<=next_count;
end
always@*
    next_count<=count+1;
always@*
begin
    if(count==4'b1111)
        max_tick<=1;
    else
        max_tick<=0;
end
endmodule
```

```
module test1();
// Inputs
reg clk;
reg reset;
// Outputs
wire [3:0] count;
wire max_tick;
// Instantiate the Un
counter uut (
.clk(clk),
.reset(reset),
.count(count),
.max_tick(max_tick)
);
always #50 clk=~clk;
initial begin
reset=1; clk=0; #100;
reset=0; #100;
reset=1; #100;
reset=0; #100;
reset=1; #100;
end
endmodule
```

D Flipflop

```
module D_flipflop(Din,clk,reset,q);
input Din,clk,reset;
output reg q;
always@(posedge clk)
begin
if(reset)
q=1'b0;
else
q=Din;
end
endmodule
```

```
module test;
// Inputs
reg Din;
reg clk;
reg reset;
// Outputs
wire q;
// Instantiate the Un
D_flipflop uut (
.Din(Din),
.clk(clk),
.reset(reset),
.q(q)
);
always #50 clk=~clk;
initial begin
clk=0;Din=0;
reset=0;#100
Din=1;reset=0;#100;
Din=0;#100
Din=1;#100;
end
endmodule
```

T Flipflop

```
module T_flipflop(T,clk,reset,q);
input T,clk,reset;
output q;
wire w;
assign w=T^q;
D_Flipflop dff(w,clk,reset,q);
endmodule

module D_Flipflop(Din,clk,reset,q);
input Din,clk,reset;
output reg q;
always@(posedge clk)
begin
if(reset)
q=1'b0;
else
q=Din;
end
endmodule
```

```
module test;
// Inputs
reg T;
reg clk;
reg reset;
// Outputs
wire q;
// Instantiate the Ui
T_flipflop uut (
.T(T),
.clk(clk),
.reset(reset),
.q(q)
);
always #50 clk=~clk;
initial begin
clk=0;T=0;
reset=0;#100
T=1;reset=0;#100;
T=0;#100
T=1;#100;
end
endmodule
```