4:1 Mux using 2:1 Mux

```
module mx4(A,S,F);
input [3:0]A;
input [1:0]S;
output  F;
wire [1:0]FF;
mx01 f1(A[1:0],S[0],FF[0]);
mx01 f2(A[3:2],S[0],FF[1]);
mx01 f3(FF[1:0],S[1],F);
endmodule

module mx01(a,s,f);
input [1:0]a,s;
output reg f;
always @(a,s)
begin
if (s==0)
f=a[0];
else
f=a[1];
end
endmodule
```

4:1 Mux

```
module mx4(d,e,f);
input [3:0]d;
input [1:0]e;
output reg f;
always @(d,e)
begin
if (e==0)
f=d[0];
else if(e==1)
f=d[1];
else if(e==2)
f=d[2];
else
f=d[3];
end
endmodule
```

16:1 Mux using 4:1 Mux

```
module mx16(A,B,C);
input [15:0]A;
input [3:0]B;
output C;
wire [3:0]CC;
mx4 f1(A[3:0],B[1:0],CC[0]);
mx4 f2(A[7:4],B[1:0],CC[1]);
mx4 f3(A[11:8],B[1:0],CC[2]);
mx4 f4(A[15:12],B[1:0],CC[3]);
mx4 f5(CC[3:0],B[3:2],C);
endmodule

module mx4(d,e,f);
input [3:0]d;
input [1:0]e;
output reg f;
always @(d,e)
begin
if (e==0)
f=d[0];
else if(e==1)
f=d[1];
else if(e==2)
f=d[2];
else
f=d[3];
end
endmodule
```

1 Digit BCD Adder

```
module bcd(a,b,c);
input [3:0]a;
input [3:0]b;
output reg c;
reg d;
always @(a,b)
begin
d=a+b;
if (d>10)
c<=d+6;
else
c<=d;
end
endmodule
```

4 bit Comparator

```
module cmpr(a,b,c,d,e,f);
input [3:0]a;
input [3:0]b;
output reg [1:0]c;
output reg [3:0]d;
output reg [3:0]e;
output reg [3:0]f;
integer g;
always @(a,b)
begin
for (g=0; g<=3; g=g+1)
begin
if (a[g]==b[g])
d[g]=1;
else if (a[g]>b[g])
e[g]=1;
else
f[g]=1;
end
if (a>b)
c<=1;
else
c<=0;
end
endmodule
```

4 bit Adder/4 Bit Full Adder using 1 bit Full Adders

```
module fadd4(A,B,C,S,C0);
input [3:0]A;
input [3:0]B;
input C;
output C0;
wire [2:0]Ca;
output [3:0]S;
fadd f1(A[0],B[0],C,S[0],Ca[0]);
fadd
f2(A[1],B[1],Ca[0],S[1],Ca[1]);
fadd
f3(A[2],B[2],Ca[1],S[2],Ca[2]);
fadd f4(A[3],B[3],Ca[2],S[3],C0);
endmodule

module fadd(a,b,c,s,ca);
input a,b,c;
output s,ca;
assign s=a^b^c;
assign ca=a*b+(a^b)*c;
endmodule
```

4 bit up-Down Counter

```
module udcounter(c,r,e,q);
input c,r,e;
output reg [3:0]q;
always @(c,r,e)
begin
if (r==0 && e==0)
q<=0;
else if (r==1 && e==0)
q<=q+1;
if (r==0 && e==1)
q<=15;
else if (r==1 && e==1)
q<=q-1;
end
endmodule
```

7 Segment Decoder

```
module seg(a,b);
input [3:0]a;
output reg [6:0]b;
always @(a)
begin
casex (a)
0: b=63;
1: b=6;
2: b=91;
3: b=79;
4: b=102;
5: b=109;
6: b=125;
7: b=7;
8: b=127;
0: b=111;
default: b=0;
endcase
end
endmodule
```

Universal Shift Register

```
module uni(a,c,s,b,y);
input a,c;
input [1:0]s;
input [3:0]b;
output reg [3:0]y;
always @(posedge c)
if (s==0)
begin
y[0]<=y[0];
y[1]<=y[1];
y[2]<=y[2];
y[3]<=y[3];
end
else if (s==1)
begin
y[0]<=b[0];
y[1]<=b[1];
y[2]<=b[2];
y[3]<=b[3];
end
else if (s==2)
begin
y[3]<=y[2];
y[2]<=y[1];
y[1]<=y[0];
y[0]<=a;
end
else
begin
y[3]<=a;
y[2]<=y[3];
y[1]<=y[2];
y[0]<=y[1];
end
endmodule
```

JK Flip-Flop using D

```
module djt(j,k,c,q);
input j,k,c;
output q;
wire a,b,d,q;
d(d,c,q);
assign a=(~q&j);
assign b=(~k&q);
assign d=a|b;
endmodule
```

```
module d(d,c,q);
input d,c;
output reg q;
always @(posedge c)
q<=d;
endmodule
```

T Flip-Flop using JK

```
module jkt(t,C,F);
input t,C;
output F;
jj(t,t,c,F);
endmodule


module jj(j,k,c,f);
input j,k,c;
output reg f;
always @(posedge c)
if ((j==1 & k==0) | (j==0 &
k==1))
f<=j;
else if (j==1 & k==1)
f<=~f;
endmodule
```