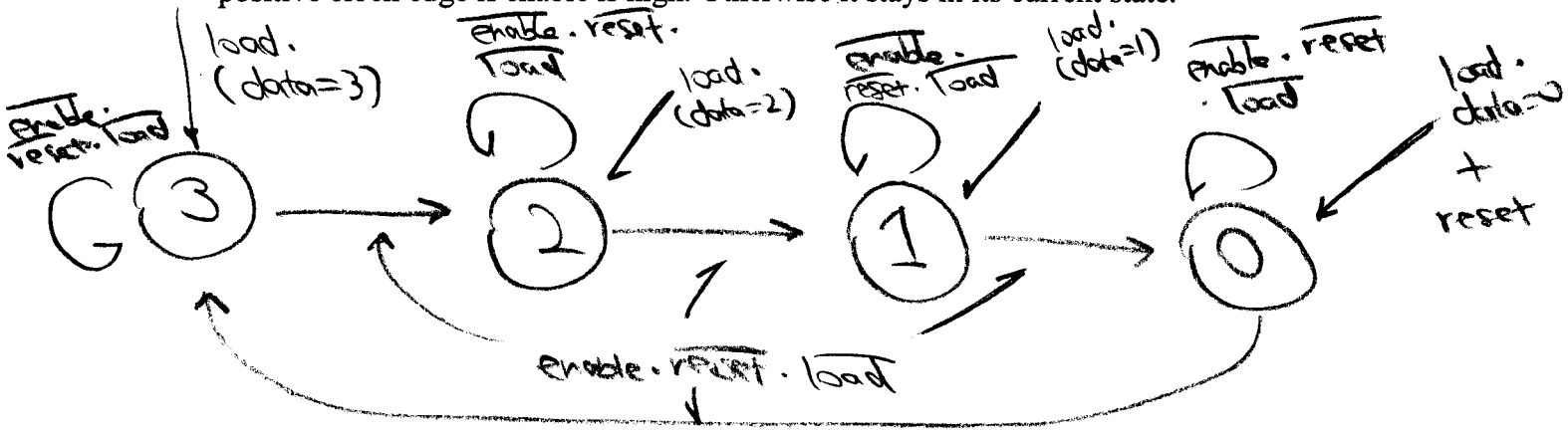
Synchronous Down Counter

Make a synchronous down counter that counts down from 3 to 0 and takes in synchronous reset, enable, load, and data. Reset takes precedence over all other inputs, and upon reset the counter should go to 0. Upon load, the FSM goes to whatever state is on the input data. If neither load or reset is high, the FSM goes to the next state on positive clock edge if enable is high. Otherwise it stays in its current state.

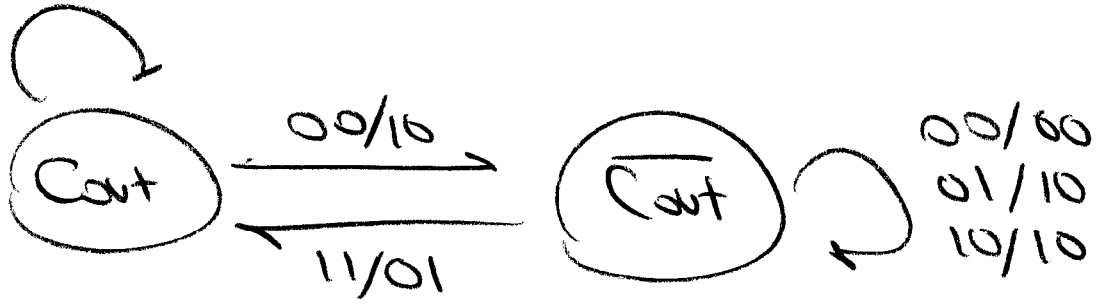



```
module DownCounter(Reset, Load, Enable, Data, Count, clk);

input Reset, Load, Enable, clk;
input [1:0]Data;
output [1:0]Count;
reg [1:0]Count;

always @(posedge clk)
begin

        if(Reset)
        Count <= 0;

        else if(Load)
        Count <= Data;

        else if(Enable && Count==0 )
        Count <= 3;

        else if(Enable)
        Count <= Count - 1;

end

endmodule
```

Adder Mealy FSM

Implement the adder from lab 5 in mealy form. Inputs are 2 bits, A and B. The outputs should be the sum and carry bits.

01/10
10/10
11/11



Input:				Output:			
A	B	Sum	Carry	A	B	Sum	Carry
0	0	x	0	x	x	0	0
0	0	x	1	x	x	1	0
0	1	x	0	0	0	1	0
0	1	x	1	0	0	0	1
1	0	x	0	x	x	1	0
1	0	x	1	x	x	0	1
1	1	x	0	x	x	0	1
1	1	x	1	x	x	1	1

```
module DownCounter(A, B, Sum, Cout, clk);

input A, B, clk;
output Sum;
output Cout;
reg Cout;

wire Sum;
assign Sum = (A ^ B) ^ Cout;

wire CoutNext;
assign CountNext = (Cout && (A || B)) && ( ~Cout && A && B );

always @ (posedge clk)
        Cout <= CoutNext;

endmodule
```

Synchronous and Asynchronous Flip Flops
Implement 2 D Flip Flops, one synchronous and the other asynchronous

Asynchronous:

```
module DFF(D, clk, R, Q);

input D, clk, R;
output Q;
reg Q;

always@(posedge clk or posedge R)
begin
        if(R)
        Q<=0;
        else
        Q<=D;
end

endmodule
```

Synchronous:

```
module DFF(D, clk, R, Q);

input D, clk, R;
output Q;
reg Q;

always@(posedge clk)
begin
        if(R)
        Q<=0;
        else
        Q<=D;
end

endmodule
```