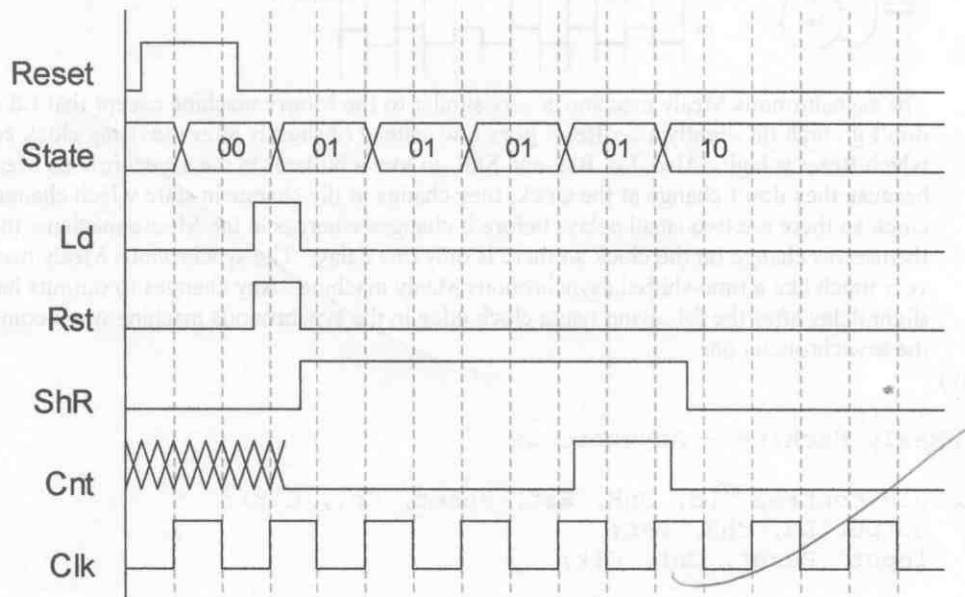


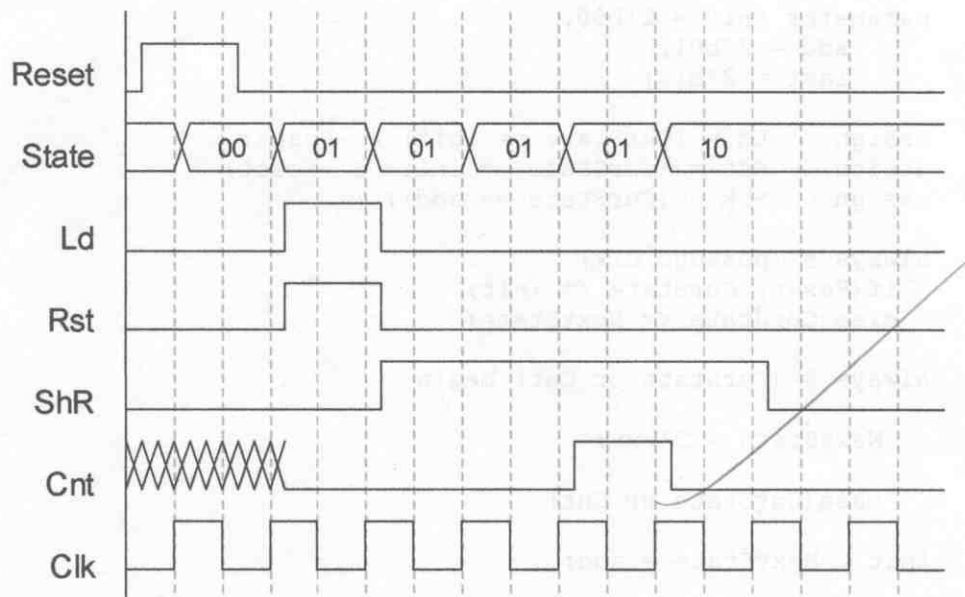
# CS150 Spring 2004

## Problem Set #5 Solutions

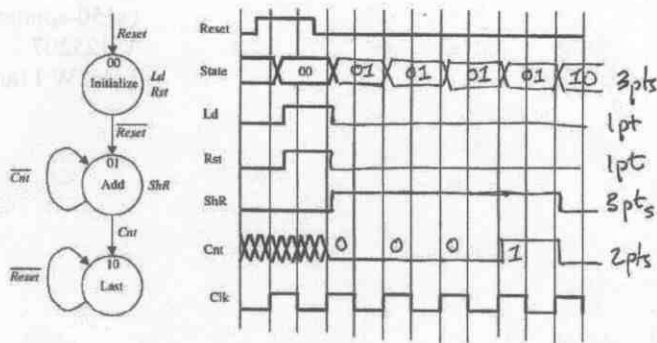
### 1. Mealy Machine (i)



### (ii)



(iii)



The asynchronous Mealy machine is very similar to the Moore machine except that Ld and Rst don't go high till slightly after Reset goes low instead of slightly after the rising clock edge during which Reset is high. Also, Ld, Rst, and ShR go low a bit later in the asynchronous Mealy machine because they don't change at the clock, they change at the change in state which changes at the clock so there are two small delays before it changes whereas in the Moore machine, they themselves change on the clock so there is only one delay. The synchronous Mealy machine is very much like a time-shifted asynchronous Mealy machine. Any changes to outputs happen a slight delay after the following rising clock edge in the synchronous machine when compared to the asynchronous one.

(iv)

//Mealy Machine - Asynchronous

```
module control (Ld, ShR, Rst, Reset, Cnt, Clk);
    output Ld, ShR, Rst;
    input Reset, Cnt, Clk;
```

```
    reg [1:0] CurState;
    reg [1:0] NextState;
```

```
    parameter init = 2'b00,
        add = 2'b01,
        last = 2'b10;
```

```
    assign Ld = (CurState == init) & ~Reset;
    assign Rst = (CurState == init) & ~Reset;
    assign ShR = (CurState == add);
```

```
    always @ (posedge Clk)
        if(Reset) CurState <= init;
        else CurState <= NextState;
```

```
    always @ (CurState or Cnt) begin
```

```
        NextState = 2'bxx;
```

```
        case(CurState or Cnt)
```

```
        init : NextState = add;
```

```
        add :
            if(~Cnt) NextState = add;
            else NextState = last;
```

```

    last : NextState = last;

    endcase // case(CurState)

end // always @ (CurState or Cnt)

endmodule // control

//Mealy Machine - Synchronous

module control (Ld, ShR, Rst, Reset, Cnt, Clk);
    output Ld, ShR, Rst;
    input  Reset, Cnt, Clk;

    reg [1:0] CurState;
    reg [1:0] NextState;
    reg      Ld, ShR, Rst;

    parameter init = 2'b00,
               add = 2'b01,
               last = 2'b10;

    always @ (posedge Clk) begin

        if(Reset) begin
            CurState <= init;
            Ld <= 0;
            ShR <= 0;
            Rst <= 0;
            end

        else begin
            CurState <= NextState;
            Ld <= nextLd;
            ShR <= nextShR;
            Rst <= nextRst;
            end // else: !if(Reset)

    end // always @ (posedge Clk)

    always @ (CurState or Cnt) begin

        NextState = 2'bxx;
        nextLd = 0;
        nextRst = 0;
        nextShR = 0;

        case(CurState)

        init : begin
            NextState = add;
            if(~Reset) begin
                nextLd = 1;

```

```

        nextRst = 1;
    end
end

add : begin
    nextShR = 1;
    if (~Cnt) NextState = add;
    else NextState = last;
end

last : NextState = last;

default : begin
    NextState = 2'bxx;
end

    endcase // case(CurState)

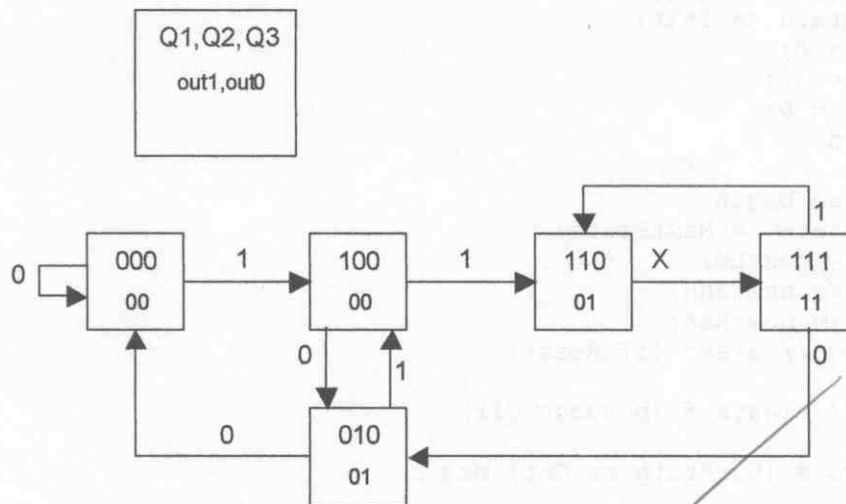
end // always @ (CurState or Cnt)

endmodule // control

```

## 2. State Machine Reverse Engineering

- (i) Moore Machine, because the outputs are solely determined by state elements (the flipflops), so it's a Moore Machine (ie: there is no path that you can take from input to output that is purely combinational).
- (ii) Inputs: in, Outputs: out0, out1



(iii)

```

module reversedFSM(out1, out0, in, clk);

    output out1, out0;
    input in, clk;

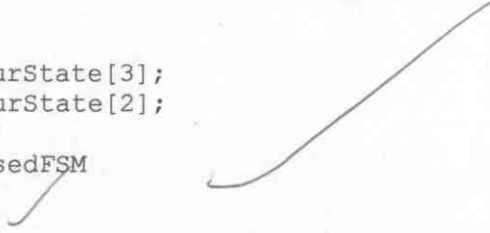
    reg [1:3] curState;

```

```
always @ (posedge clk) begin
    curState[1] <= in;
    curState[2] <= curState[1];
    curState[3] <= (curState[1] & curState[2]) ^| curState[3];
end

assign out1 = curState[3];
assign out0 = curState[2];

endmodule // reversedFSM
```



- (iv) It is self-starting
- (v) I can't think of any sort of common function that it performs ☹.