


EE461 Verilog-HDL

Week 13 FSM & Digial Design

Alex Yang, Engineering School, NPU

Week 13 Outlines

- Finite State Machine
  - Design by pen & paper
  - Mealy machine & Moore machine
  - Design in Verilog
  - State coding in FSM
  - Questions
- Digital Hardware Designs
  - Basic hardware module designs
    - Encoder/Decoder;
    - Mux/Demux;
    - Counter;
    - Edge Detector;
    - Memory;
    - Parity & CRC

Week 13 Outlines

- Digital Hardware Designs
  - Synchronization designs
    - Single bit synchronizer
    - More bits handshaking protocol
    - FIFO designs
      - Synchronous FIFO
      - Asynchronous FIFO for different clock domain

Week 13 Outlines

- Digital Hardware Designs
  - Data Communication
    - UART design
    - I2C design
    - Round robin arbiter design
  - DSP design examples
    - FIR filter design
    - IIR filter design
  - Computer Architecture design
    - ALU / Processor

Week 13 Outlines

- Race Condition Issues
  - Guideline to Avoid Race Conditions

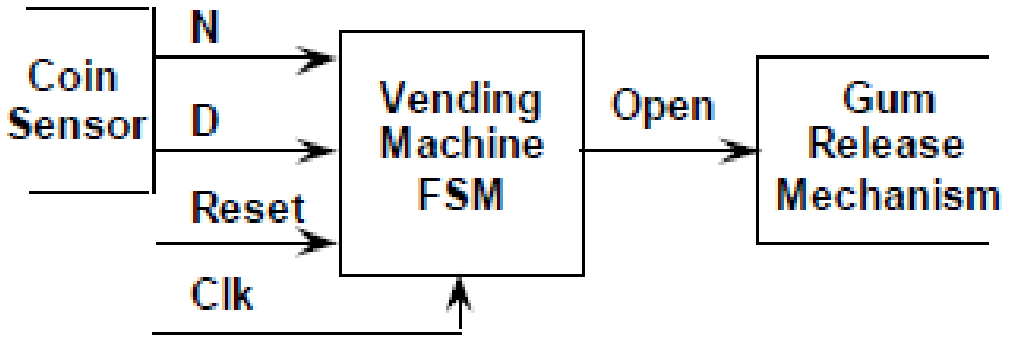
Finite State Machine

- Design by pen & paper
  - Step 1. State block diagram
  - Step 2. State diagram simplification
  - Step 3. Truth table
  - Step 4. Logic function
  - Step 5. Logic schematics

Finite State Machine

- Design by pen & paper
  - Design example: vending machine

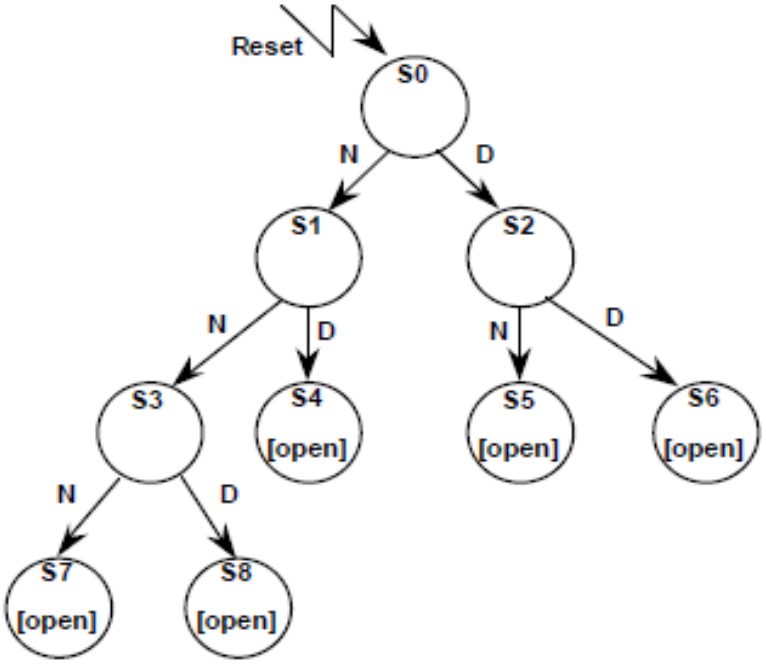
Input signals: dimes, nickels, clk, rst from coin sensor;
Output signals: open only (no change)



Finite State Machine

- Design by pen & paper
  - Step 1. State block diagram

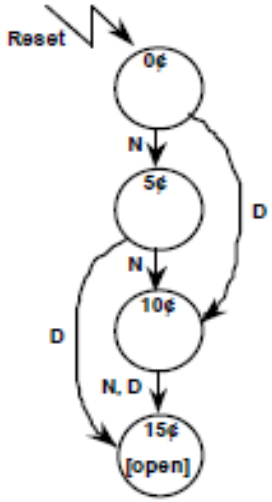
D : Dime; N : Nickel



Finite State Machine

- Design by pen & paper
  - Step 2. State diagram simplification

D : Dime; N : Nickel



Present State	Inputs D	Inputs N	Next State	Output Open
0¢	0	0	0¢	0
	0	1	5¢	0
	1	0	10¢	0
	1	1	X	X
5¢	0	0	5¢	0
	0	1	10¢	0
	1	0	15¢	0
	1	1	X	X
10¢	0	0	10¢	0
	0	1	15¢	0
	1	0	15¢	0
	1	1	X	X
15¢	X	X	15¢	1

Finite State Machine

- Design by pen & paper
  - Step 3. Truth table

Present State Q_1	Present State Q_0	Inputs D	Inputs N	Next State D_1	Next State D_0	Output Open
0	0	0	0	0	0	0
		0	1	0	1	0
		1	0	1	0	0
		1	1	X	X	X
0	1	0	0	0	1	0
		0	1	1	0	0
		1	0	1	1	0
		1	1	X	X	X
1	0	0	0	1	0	0
		0	1	1	1	0
		1	0	1	1	0
		1	1	X	X	X
1	1	0	0	1	1	1
		0	1	1	1	1
		1	0	1	1	1
		1	1	X	X	X

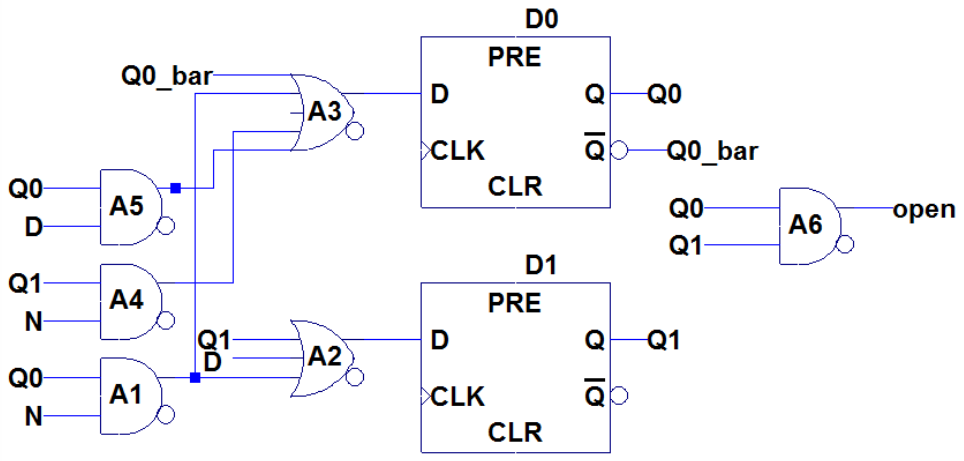
Finite State Machine

- Design by pen & paper
  - Step 4. Logic function

    D1 = Q1 + D + Q0*N;

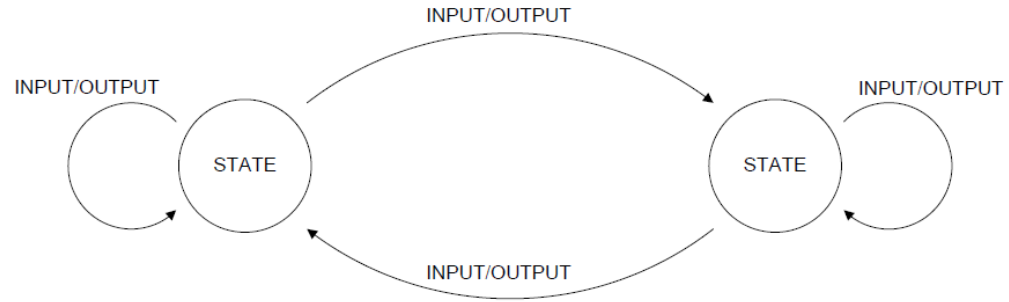
    D0 = Q0_bar + Q0*N + Q1*N + Q0*D;

    Open = Q1*Q0;
  - Step 5. Logic schematics

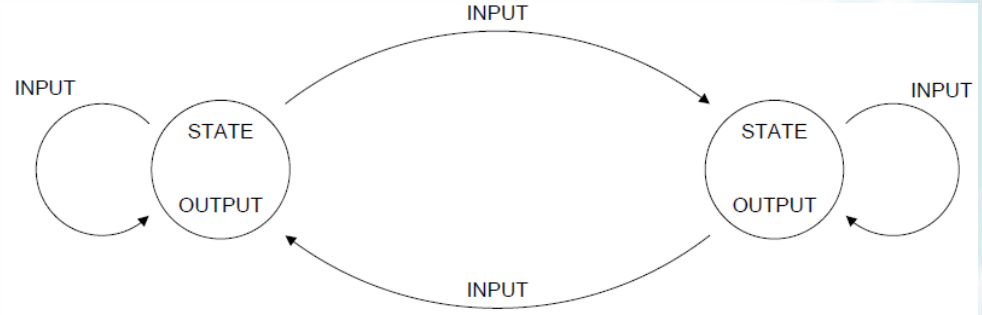


Finite State Machine

- Mealy Machine & Moore Machine
  - Mealy Machine: output <= current state + input signals



  - Moore Machine: output <= current state



Finite State Machine

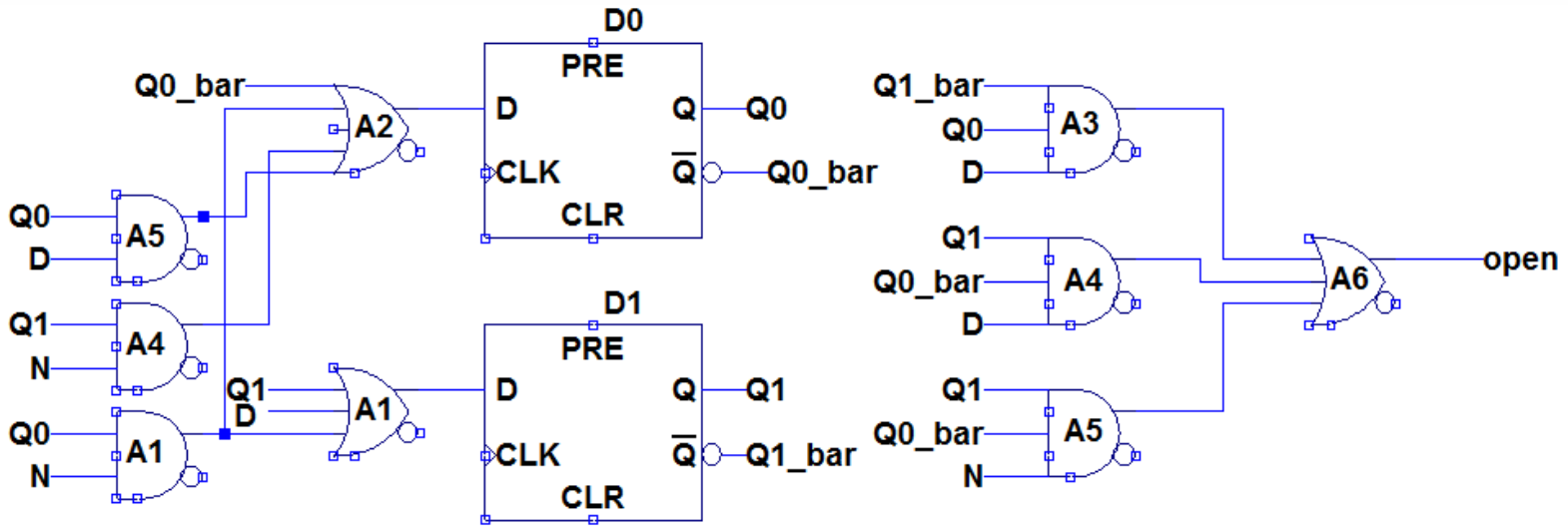
- Mealy Machine & Moore Machine
  - Mealy Machine: from truth table

State	Q1	Q0	D	N	Q1'	Q0'	Open
0¢	0	0	0	0	0¢: 0	0	0
	0	0	0	1	5¢: 0	1	0
	0	0	1	0	10¢: 1	0	0
	0	0	1	1	x		x
5¢	0	1	0	0	5¢: 0	1	0
	0	1	0	1	10¢: 1	0	0
	0	1	1	0	15¢: 1	1	1
	0	1	1	1	x		x
10¢	1	0	0	0	10¢: 1	0	0
	1	0	0	1	15¢: 1	1	1
	1	0	1	0	15¢: 1	1	1
	1	0	1	1	x		x
15¢	1	1	x	x	0¢: 0	0	0

Open = Q1_bar*Q0*D + Q1*Q1_bar*D + Q1*Q1_bar*N

Finite State Machine

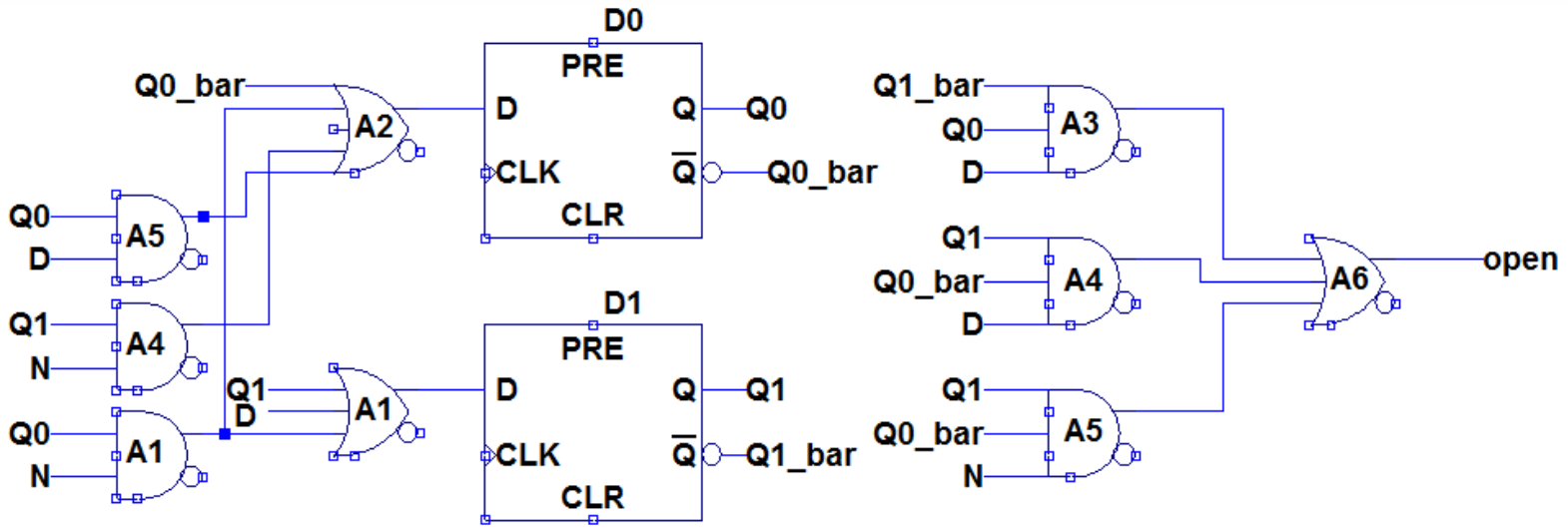
- Mealy Machine & Moore Machine
  - Mealy Machine:



  - Moore Machine: as previous example

Finite State Machine

- Mealy Machine & Moore Machine
  - Mealy Machine:



  - Moore Machine: as previous example

Finite State Machine

- Design in Verilog: (Mealy Machine)

```
module Vending(
                        input wire D_i,
                        input wire N_i,
                        input wire rst_i,
                        input wire clk_i,
                        output reg open_o
);
  parameter    pZeroCent = 2'b00;
  parameter    pFiveCent = 2'b01;
  parameter    pTenCent  = 2'b10;
  parameter    pFifteenCent = 2'b11;

  reg [1:0]     curSt_r;
  reg [1:0]     nxtSt_r;
  always@(posedge clk_i)begin
    if (rst_i) curSt_r <= pZeroCent;
    else       curSt_r <= nxtSt_r;
  end
  always@(*)begin
    open_o = 1'b0;
    case(curSt_r)
      pZeroCent: begin
        case({D_i, N_1})
          2'b00: begin
                nxtSt_r= pZeroCent;
          end
          2'b01: begin
                nxtSt_r= pFiveCent;
          end
          2'b10: begin
                nxtSt_r= pTenCent;
          end
        default: nxtSt_r = pZeroCent;
        endcase
      end
      pFiveCent: begin
        case({D_i, N_1})
          2'b00: begin
                nxtSt_r= pFiveCent;
          end
```

Finite State Machine

- Design in Verilog: (Mealy Machine)

```
        2'b01: begin
                nxtSt_r= pTenCent;
        end
        2'b10: begin
                open_o = 1'b1;
                nxtSt_r= pFifteenCent;
        end
                default: nxtSt_r = pZeroCent ;
    endcase
end
pTenCent:begin
    case({D_i, N_1})
        2'b00: begin
                nxtSt_r= pTenCent;
        end
        2'b01: begin
                open_o = 1'b1;
                nxtSt_r= pFifteenCent;
        end
        2'b10: begin
                open_o = 1'b1;
                nxtSt_r= pFifteenCent;
        end
        2'b10: begin
                open_o = 1'b1;
                nxtSt_r= pFifteenCent;
        end
                default: nxtSt_r = pZeroCent;
    endcase
end
pFifteenCent:begin
                nxtSt_r= pZeroCent;
end
default: begin
                nxtSt_r = pZeroCent;
end
endcase
end
endmodule
```

Finite State Machine

- Design in Verilog: (Mealy Machine)

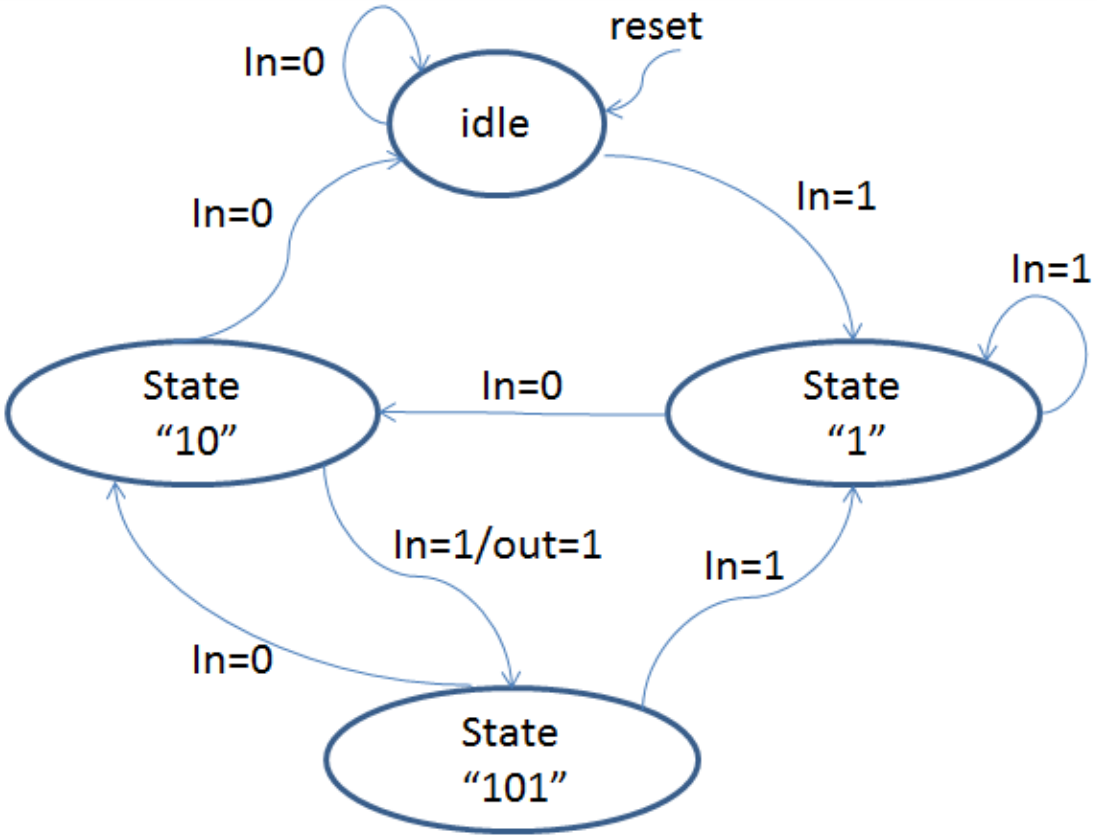
  In Verilog code, there are two “always” blocks. One is for sequential logic using nonblocking assignment and the other is for combinational logic using blocking assignment.

Finite State Machine

- State Coding in FSM:
  - Binary code
    - Less number of DFFs
    - More Combo Logic Devices
    - Lower Speed (Lower fclk)
    - State Transition Errors
  - One-hot code
    - More number of DFFs
    - Less Combo logic Devices
    - Faster (Hi fclk)
    - State Transition Errors
  - Gray code
    - Less number of DFFs
    - More Combo Logic Devices
    - Lower Speed (Lower fclk)
    - Less State Transition Errors

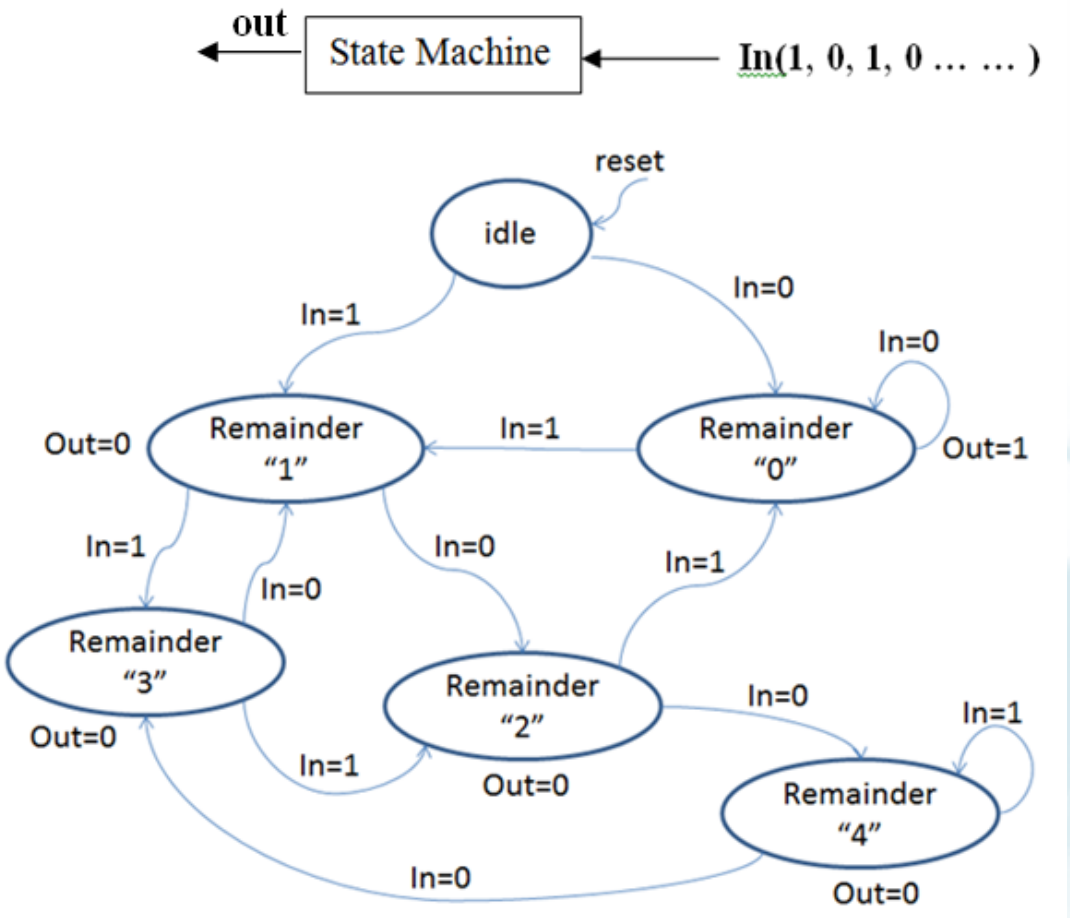
Finite State Machine

- Questions: detect 101 by FSM



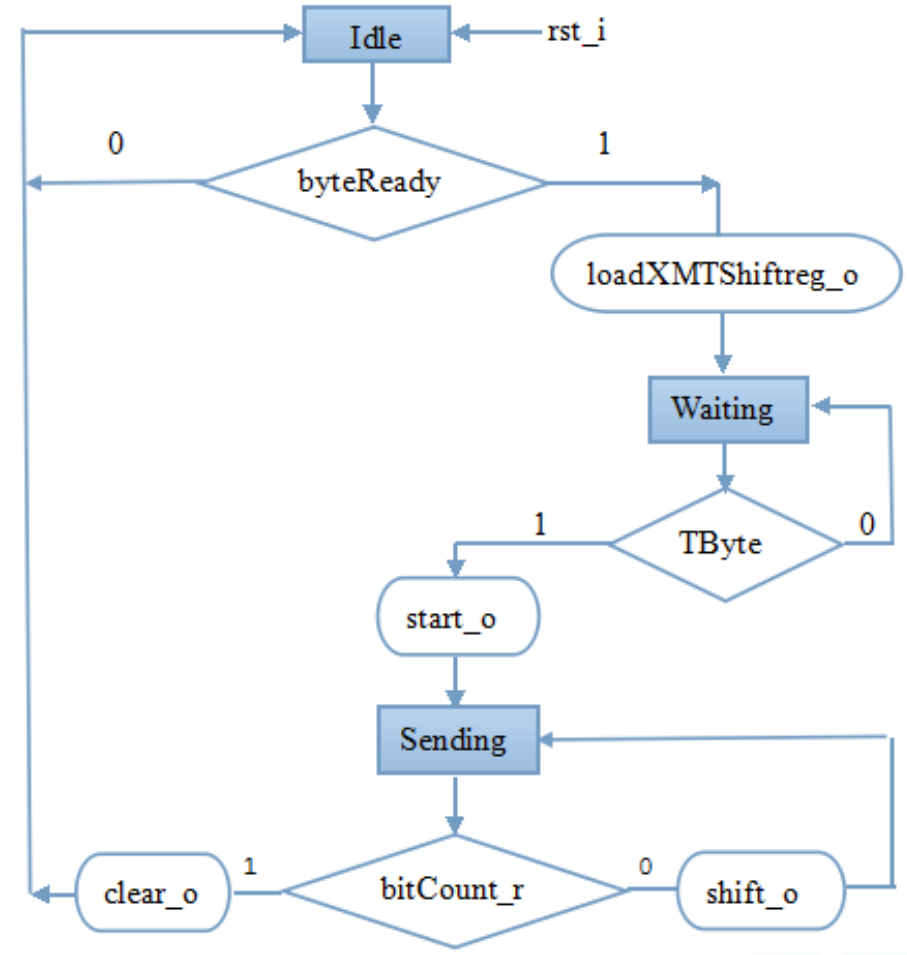
Finite State Machine

- Questions: detect bit stream divisible by 5



Finite State Machine

- Example: UART Algorithmic State Machine Chart (ASMC)

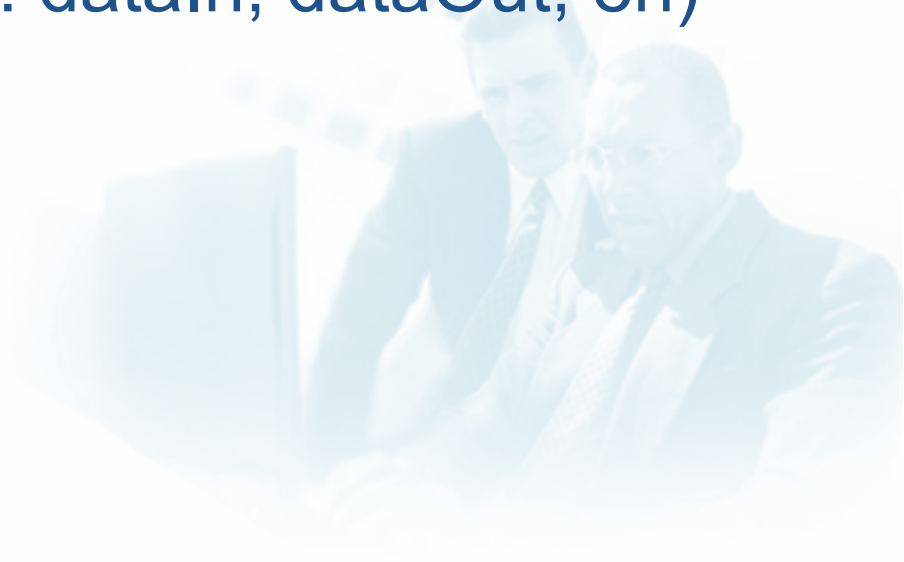


Digital Hardware Designs

- Basic hardware module designs
  - Encoder; (input/output: dataIn, dataOut, en)

    0000_0000_0000_0000 => 0000
    0000_0000_0000_0001 => 0001
    0000_0000_0000_0010 => 0010
    0000_0000_0000_0100 => 0011
    ... ...
    1000_0000_0000_0000 => 1111
  - Decoder; (input/output: dataIn, dataOut, en)

    0000 => 0000_0000_0000_0000
    0001 => 0000_0000_0000_0001
    0010 => 0000_0000_0000_0010
    0011 => 0000_0000_0000_0100
    ... ...
    1111 => 1000_0000_0000_0000



Digital Hardware Designs

- Basic hardware module designs
  - Mux/Demux; (4-to-1)
  - Counter;
    - load, upDown counter (8 bits)

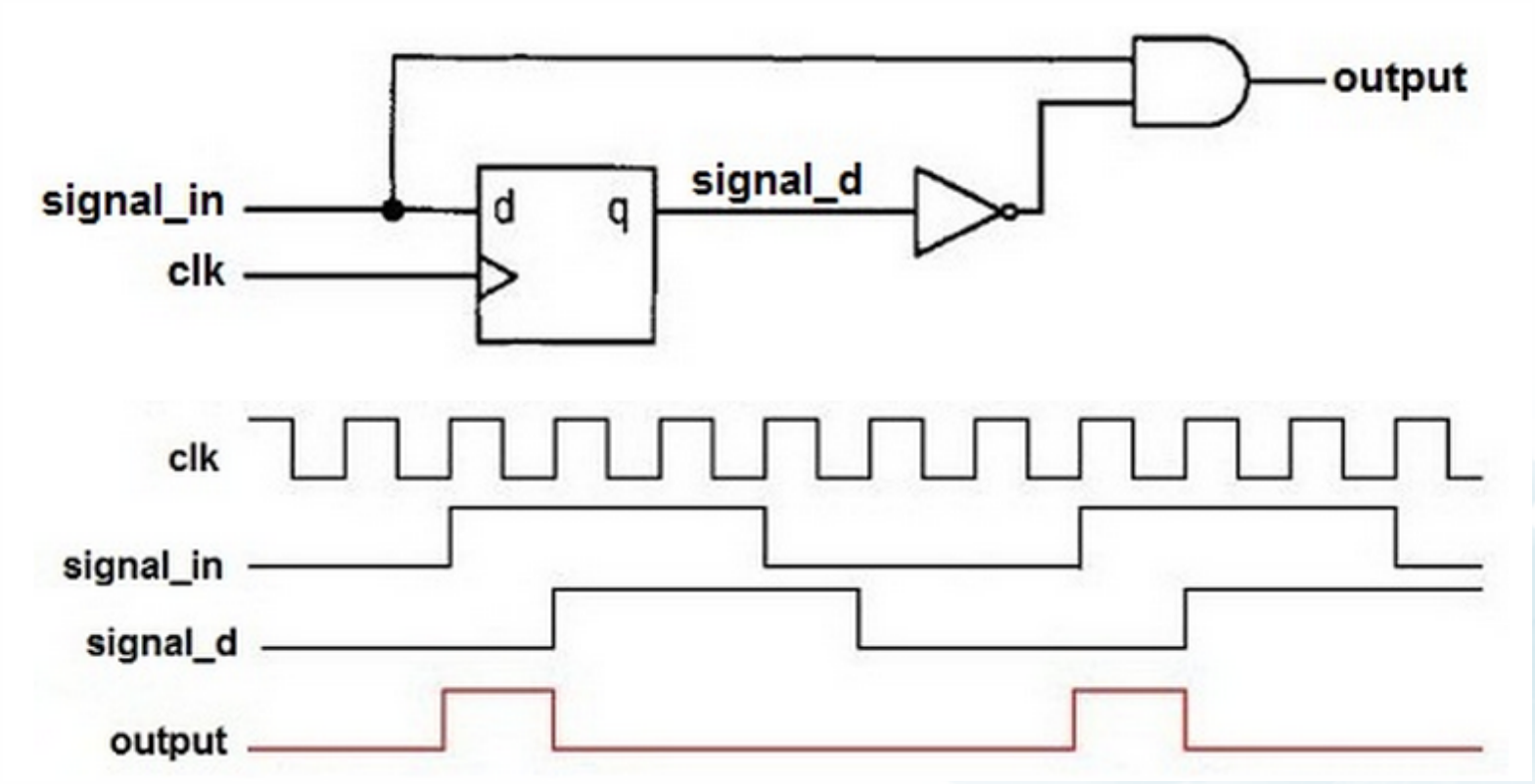
      out, data, load, enable(counting), upDown, clk, reset
    - load, upDown base counter (8 bits)

      out, data, load, enable(counting), upDown, clk, reset
    - LFSR counter (linear feedback shift register)

      x^4 + x3 + 1
    - counter divided by 3 or 3.5 or 5 or 7

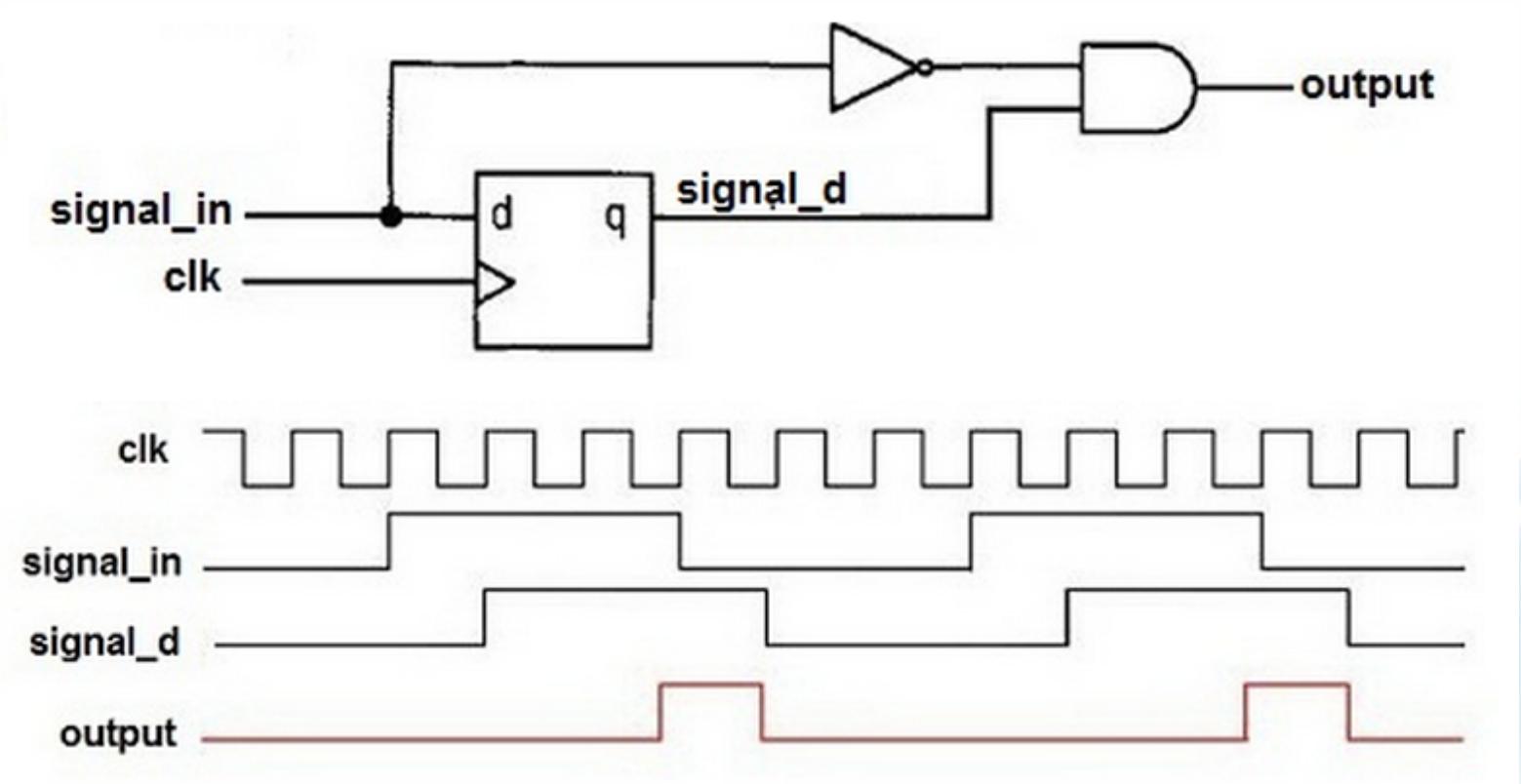
Digital Hardware Designs

- Basic hardware module designs
  - Edge Detector: (rising edge, synchronized signal_in)



Digital Hardware Designs

- Basic hardware module designs
  - Edge Detector: (falling edge, synchronized signal_in)



Digital Hardware Designs

- Basic hardware module designs
  - Memory: (dual synch. port, non-synthesizable)

```
module synDualram ( clk, address_0,
data_0,cs_0,we_0,oe_0,address_1 ,
data_1, cs_1, we_1,oe_1
);

parameter data_0_WIDTH = 8 ;
parameter ADDR_WIDTH = 8 ;
parameter RAM_DEPTH = 1 << ADDR_WIDTH;

//-------------Input Ports----------------------
input [ADDR_WIDTH-1:0] address_0 ;
input cs_0 ;
input we_0 ;
input oe_0 ;
input [ADDR_WIDTH-1:0] address_1 ;
input cs_1 ;
input we_1 ;
input oe_1 ;
```

```
//-------------Internal variables----------------
reg [data_0_WIDTH-1:0] data_0_out ;
reg [data_0_WIDTH-1:0] data_1_out ;
reg [data_0_WIDTH-1:0] mem [0:RAM_DEPTH-1];

always @ (posedge clk)
begin : MEM_WRITE
  if ( cs_0 && we_0 ) begin
     mem[address_0] <= data_0;
  end else if (cs_1 && we_1) begin
     mem[address_1] <= data_1;
  end
end

// Tri-State Buffer control
// output : When we_0 = 0, oe_0 = 1, cs_0 = 1
assign data_0 = (cs_0 && oe_0 && !we_0) ?
                        data_0_out : 8'bz;
```

Digital Hardware Designs

- Basic hardware module designs
  - Memory: (dual synch. port, non-synthesizable)

```
// Memory Read Block
// Read Operation : When we_0 = 0, oe_0 = 1,
cs_0 = 1
always @ (posedge clk)
begin : MEM_READ_0
  if (cs_0 && !we_0 && oe_0) begin
    data_0_out <= mem[address_0];
  end else begin
    data_0_out <= 0;
  end
end

//Second Port of RAM
// Tri-State Buffer control
// output : When we_0 = 0, oe_0 = 1, cs_0 = 1
assign data_1 = (cs_1 && oe_1 && !we_1) ?
                    data_1_out : 8'bz;
```

```
// Memory Read Block 1
// Read Operation : When we_1 = 0,
oe_1 = 1, cs_1 = 1
always @ (posedge clk)
begin : MEM_READ_1
  if (cs_1 && !we_1 && oe_1) begin
    data_1_out <= mem[address_1];
  end else begin
    data_1_out <= 0;
  end
end

endmodule
```

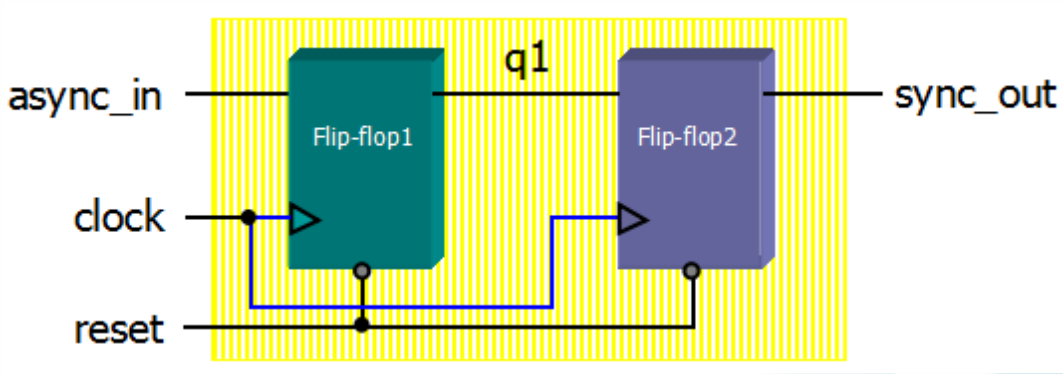
Digital Hardware Designs

- Basic hardware module designs
  - Parity & CRC

    See the special file about this

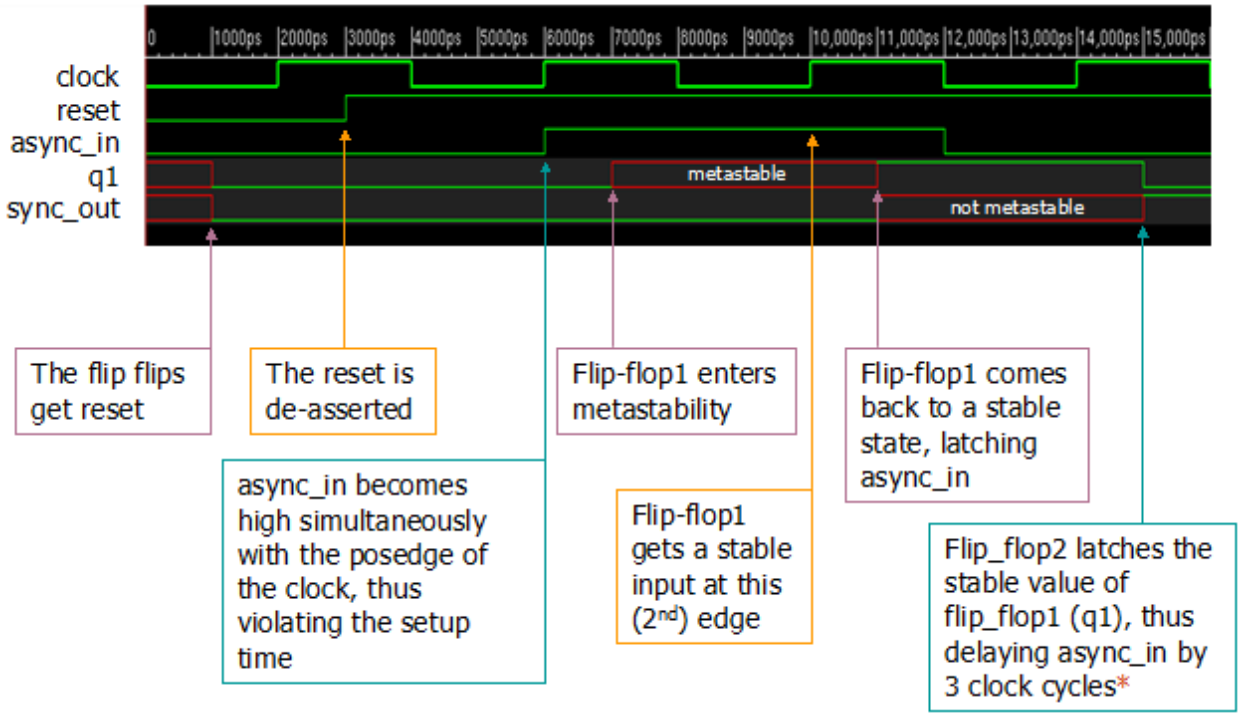
Digital Hardware Designs

- Synchronization designs
  - Single bit synchronizer
    - Case 1: Tasync_in > Tclock



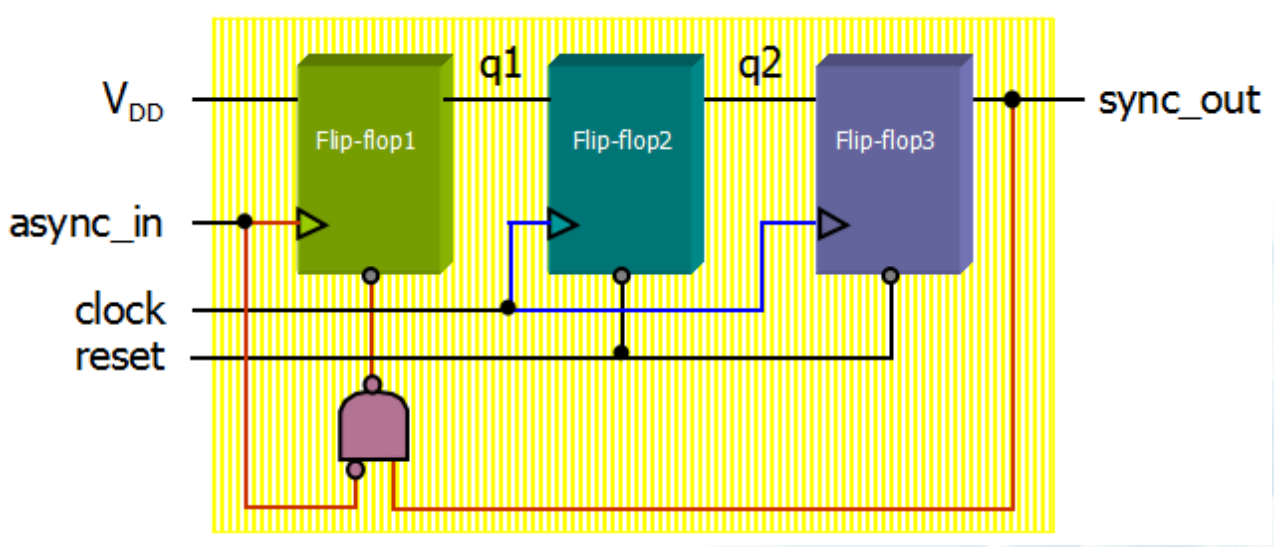
Digital Hardware Designs

- Synchronization designs
  - Single bit synchronizer
    - Case 1: Tasync_in > Tclock



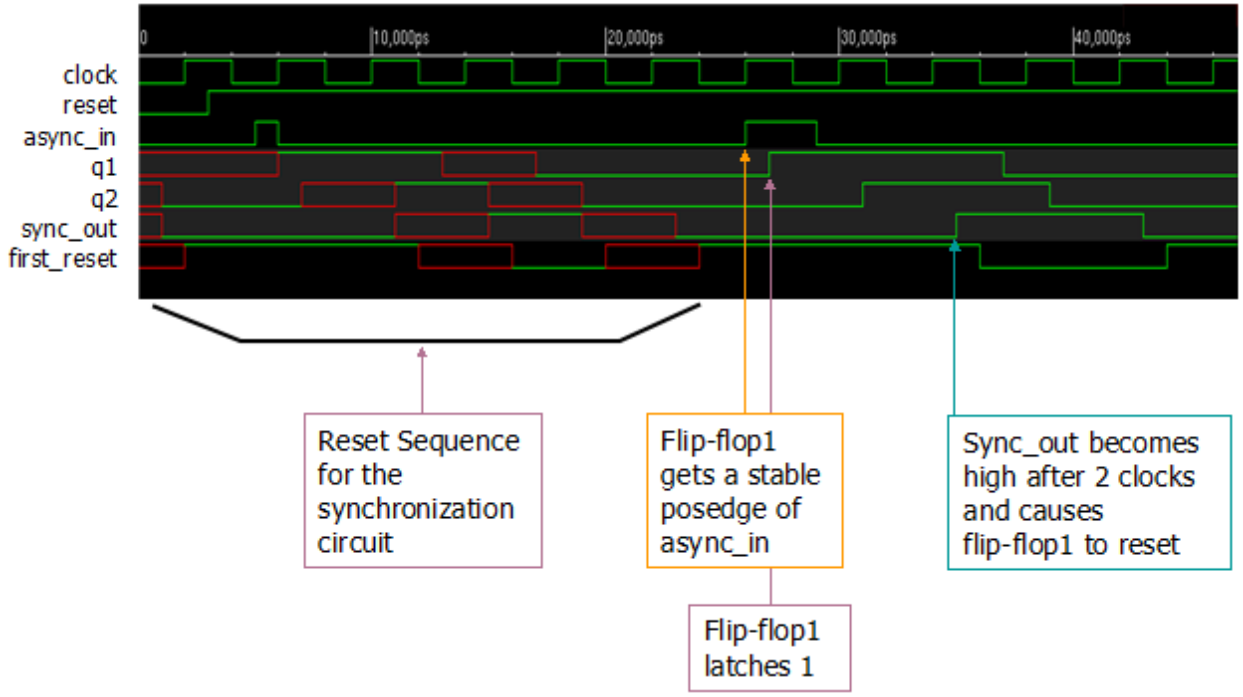
Digital Hardware Designs

- Synchronization designs
  - Single bit synchronizer
    - Case 2: Tasync_in < Tclock



Digital Hardware Designs

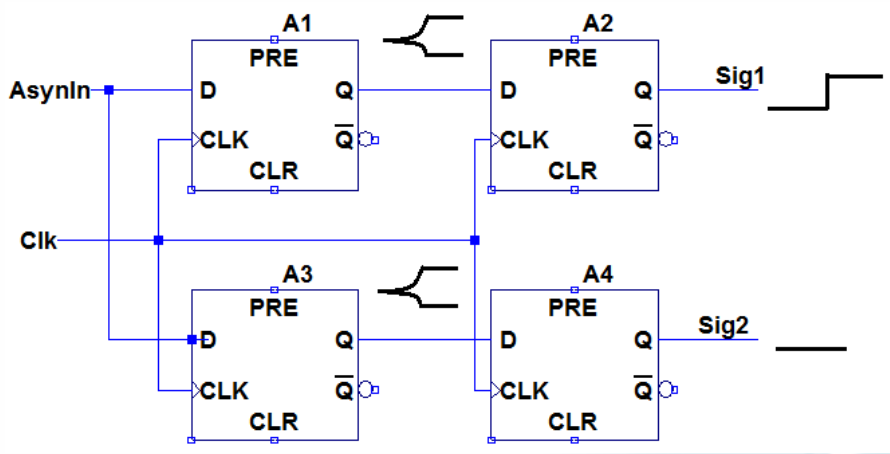
- Synchronization designs
  - Single bit synchronizer
    - Case 2: Tasync_in < Tclock



Digital Hardware Designs

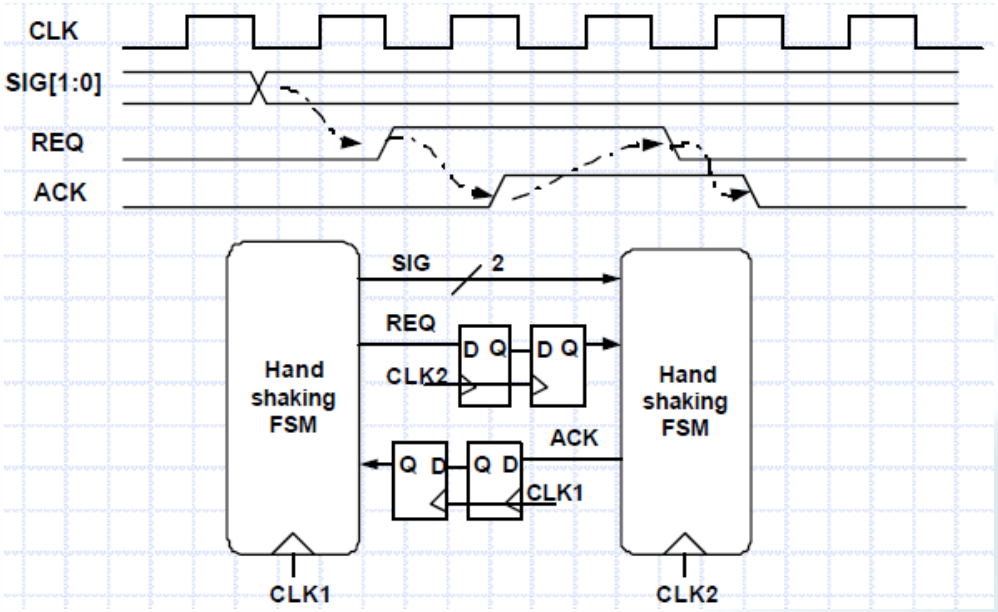
- Synchronization designs
  - Single bit synchronizer pitfall

    Never synchronize the same signal in multiple places!
    Inconsistency will result!



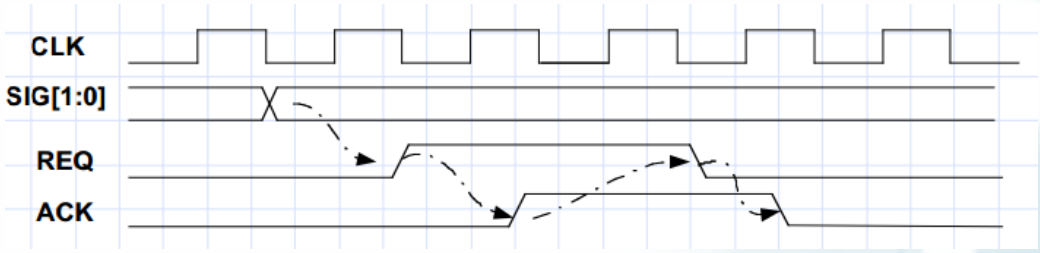
Digital Hardware Designs

- Synchronization designs
  - More bits handshaking protocol
    - Bus Synchronizer: more possibilities in metastable state
    - Handshaking is an answer



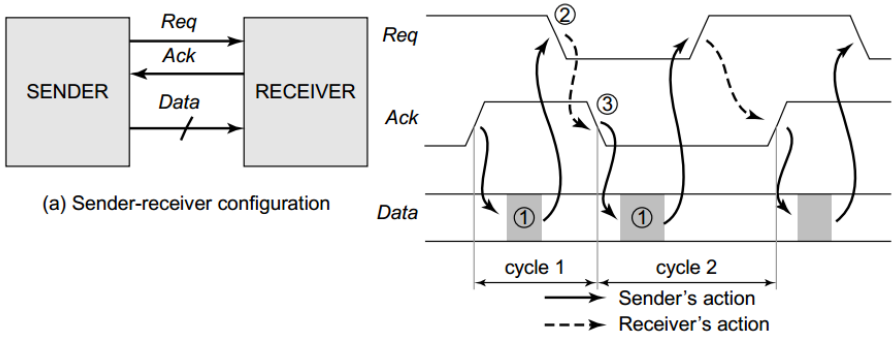
Digital Hardware Designs

- Synchronization designs
  - More bits handshaking protocol
    - 4-phase rule
      - req (sender: please get it-> data is there)
      - ack (receiver: get it -> receive data)
      - dereq (sender: sending is done)
      - deack (receiver: receiving is done, and next)



Digital Hardware Designs

- Synchronization designs
  - More bits handshaking protocol
    - 2-phase rule:
      - req (sender: please get it-> data is there)
      - ack (receiver: get it -> next)



Digital Hardware Designs

- Synchronization designs
  - Sync. FIFO design: (single clock)
    - Write/Read pointer moving
    - Pointer Counter
    - Full/Empty signal generator
    - Write-in/Read-out from memory

    See the special file about this
  - Async. FIFO design:
    - two different frequency clocks

      See the special file about this

Digital Hardware Designs

- Data Communication
  - UART design
    - Universal Asynchronous Receiver/Transmitter
    - It includes RSRS232、RS499、RS423、RS422 and RS485

      See the special file about this
  - I2C design
    - Inter-Integrated Circuit protocol
    - Used for commnumcation among chips and blocks in a chip

    See the special file about this

Digital Hardware Designs

- Data Communication
  - Round robin arbiter design

Race Condition Issues

- Guideline to Avoid Race Conditions
  - Guideline #1: Sequential logic - use nonblocking assignments
  - Guideline #2: Latches - use nonblocking assignments
  - Guideline #3: Combinational logic in an always block - use blocking assignments
  - Guideline #4: Mixed sequential and combinational logic in the same always block - use nonblocking assignments
  - Guideline #5: Do not mix blocking and nonblocking assignments in the same always block
  - Guideline #6: Do not make assignments to the same variable from more than one always block
  - Guideline #7: Use $strobe to display values that have been assigned using nonblocking assignments
  - Guideline #8: Do not make #0 procedural assignments