

Verilog

Module #6

Powerful inference capabilities of the synthesis tools

RTL Coding style

Use a case statement to describe both, the CU and the DPU

Take caution in coding the RESET section

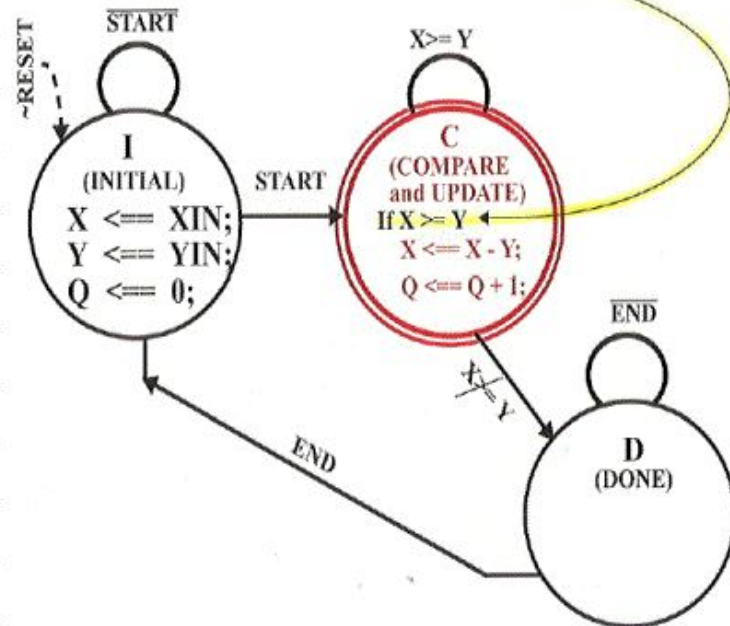
**By
Gandhi Puvvada**



Engineer

Engineer is responsible for design and design improvement.

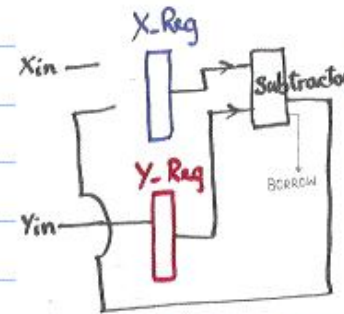
Divider -- Mealy machine design



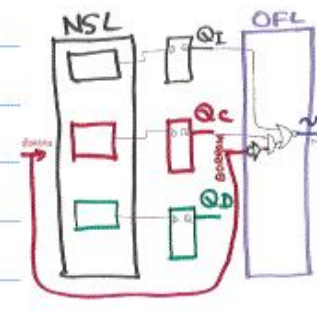
State diagram
with RTL

Technician

Technician is responsible for design detailing (DPU, CU, Timing design).



DPU



CU

Timing design

Setup Time check
Hold Time check

CAD Tools to synthesize

Now, CAD tools do what the skillful experienced technician was doing 30 years back. CAD tools can do much more tirelessly!

Schematic

Schematic-based design of 1970's

1.

2.

3.

4.

HDL RTL-level

HDL-based design currently used

1.

2.

3.

4.

Schematic

Schematic-based design of 1970's

1. Explicit

2. Manual
Inference

3. Low-level design

4. Error-prone

HDL RTL-level

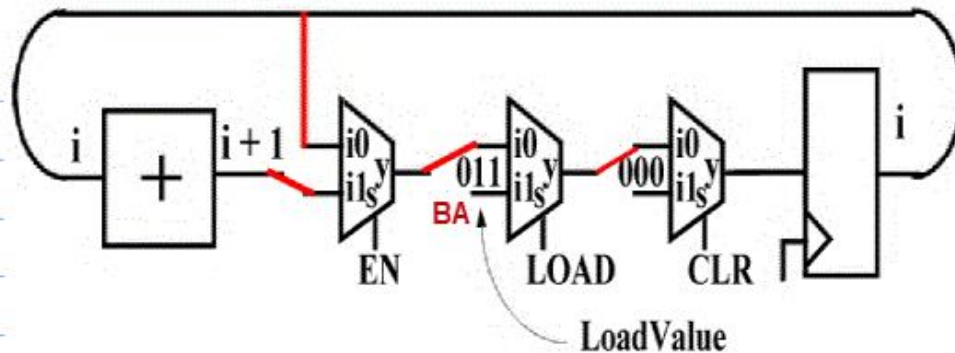
HDL-based design currently used

1. Implicit

2. Tool Infers

3. High-level
design

4. Reliable and
maintainable

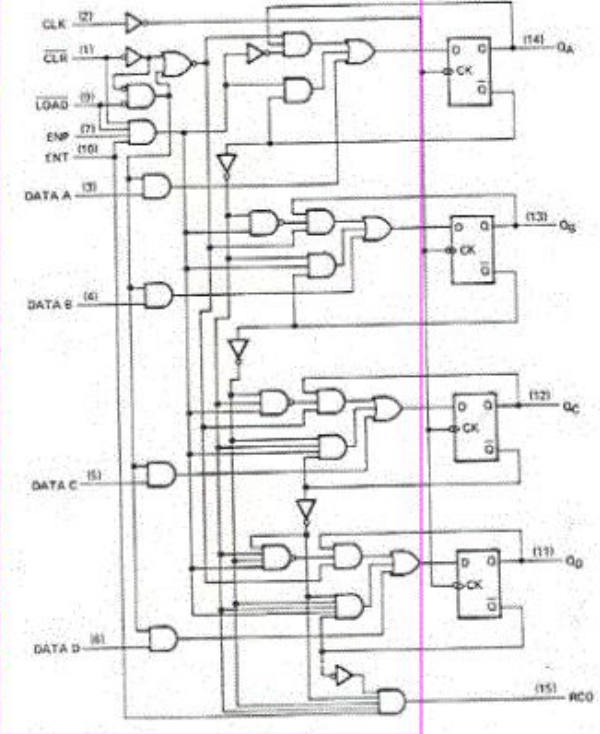


```
always @ (posedge clk)
```

```
begin : COUNT_BLOCK
    if (clr)
        Q <= 3'b000;
    else if (load)
        Q <= BA;
    else if (en)
        Q <= Q + 1;
end
```

upstream of the
4-bit Register
Combinational Logic

74LS163A 4-bit counter



We code a counter with clear, load and enable controls

-- not at gate level

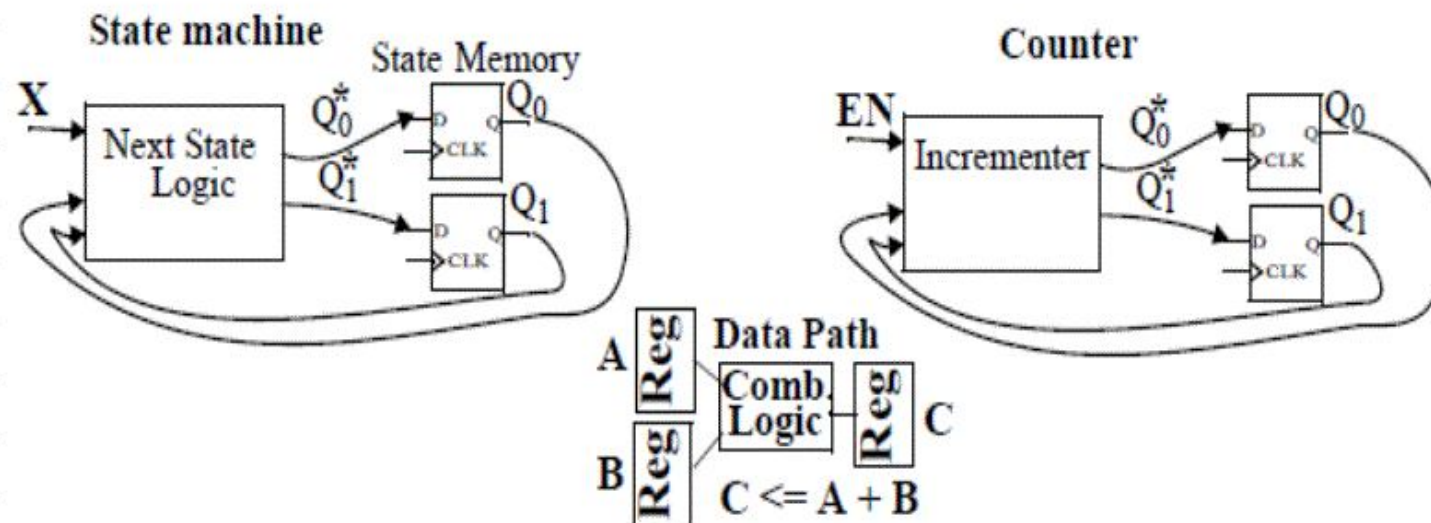
-- not at MSI level

but **behaviorally** using if .. else statement in a procedural "always" block.

Pure Synchronous Design

Single Clock, all registers use the same edge of that single clock. Design can be viewed as a bunch of registers with combinational logic in between them.

HDL coding using RTL design is easy for pure synchronous designs as you can use `always @ (posedge Clk, negedge Reset_b)`

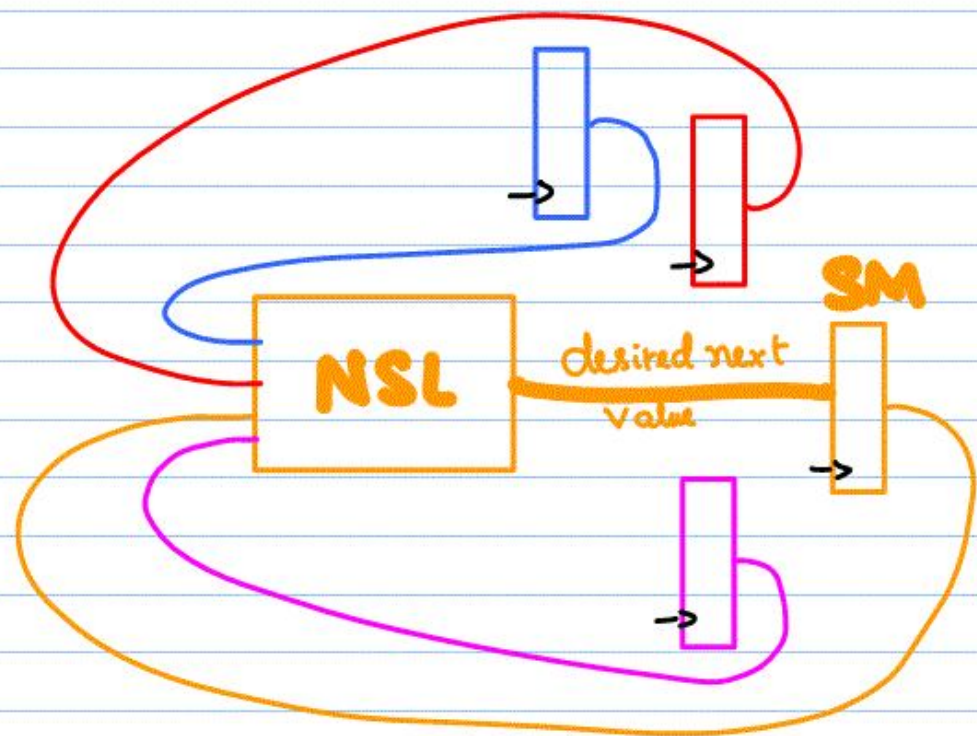
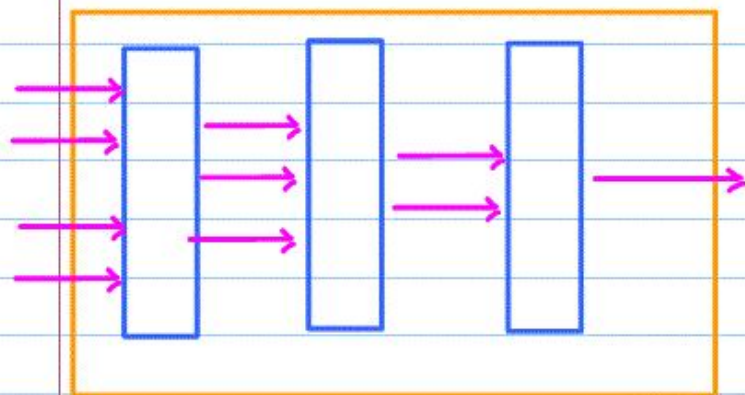


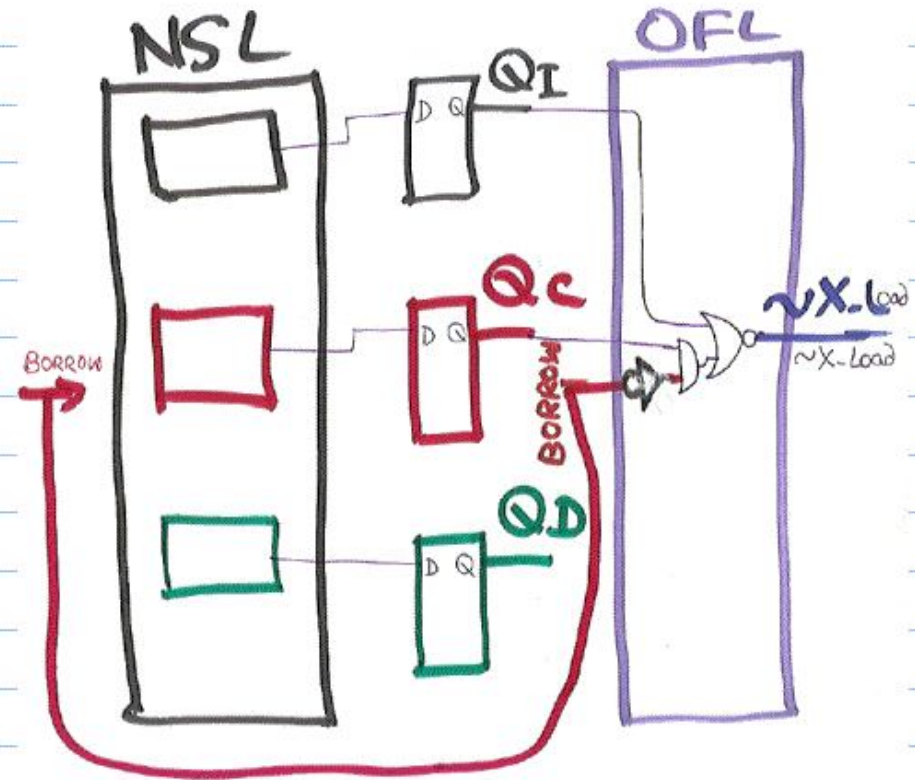
```
always @ (posedge Clk, negedge Reset_b)
```

In one always block both the register (say the SM) and also its upstream logic (say the NSL) can be coded together.

Many engineers in industry do not know this as they are basically glorified technicians. They think that the combinational logic needs to be coded separately in combinational always blocks and the the clocked block is just to update the register with what the combinational block has produced. It is sad that they did not go to USC! (Or it is good that they are

NSL

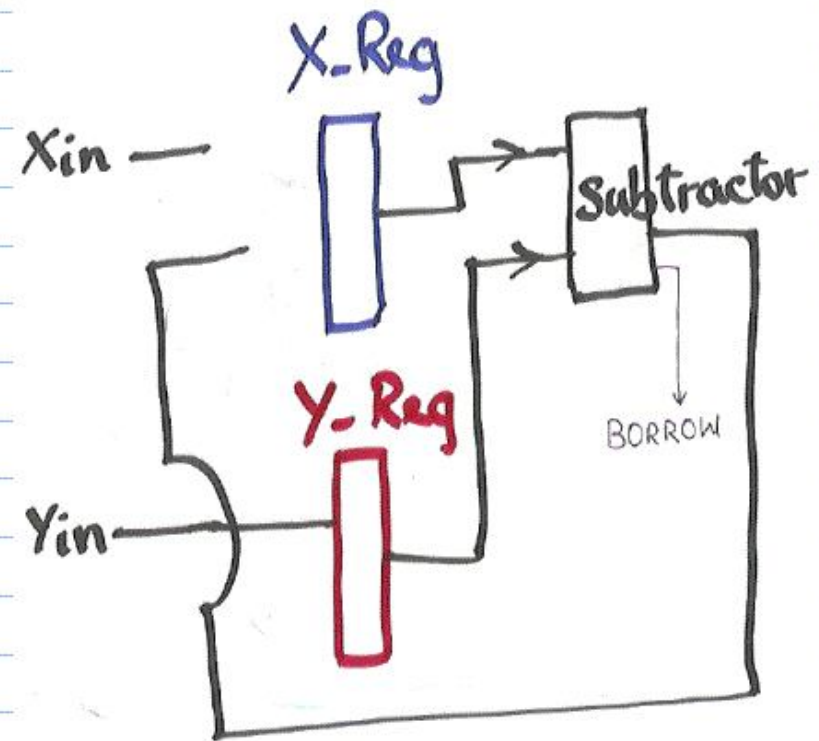
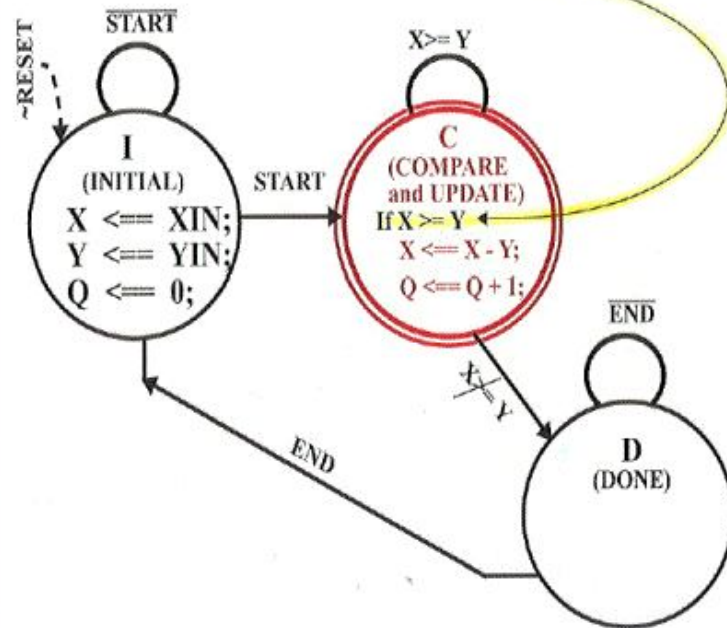




Synthesis tool can be told to use a number of state coding choices (one-hot, encoded, output-coded, gray-code, etc.) and it will build the control unit for us!

Synthesis

Divider -- Mealy machine design

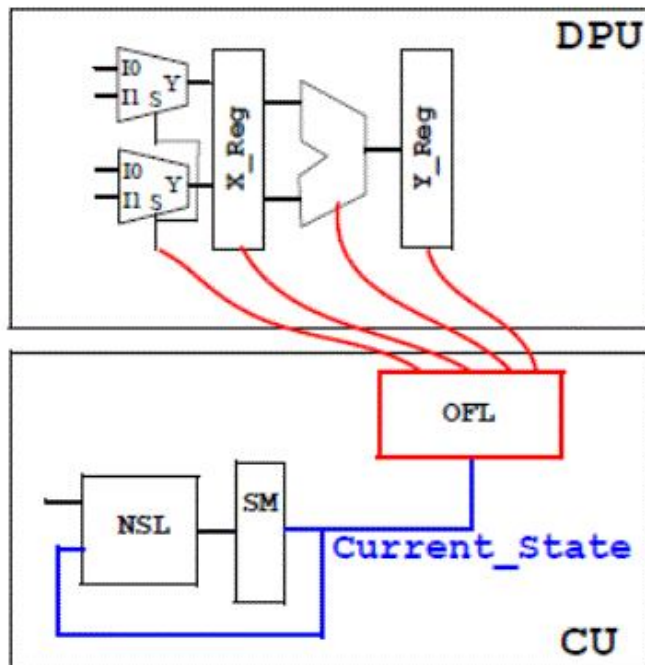


Synthesis tool uses the same "Intercept and Inject method" that we use, and builds the data path unit for us! It optimizes both the CU and DPU to meet timing and area constraints imposed by us.

Synthesis

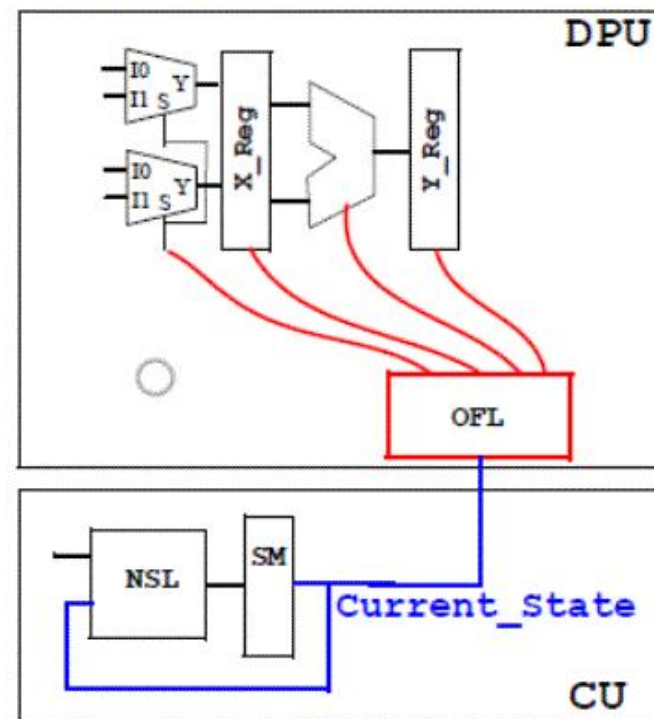
Traditional division between DPU and CU

OFL (combinational logic) is in the CU.



Division between DPU and CU for HDL coding

OFL (combinational logic) is moved to DPU. It is NOT coded explicitly. The OFL is implicit in the DPU's RTL in the CASE statement.



```
case (state)
  INITIAL :
    begin
      // state transitions in the control unit
      if (Start)
        state <= COMPUTE;
      // RTL operations in the Data Path
      x <= Xin;
      y <= Yin;
      Quotient <= 0;
    end
  COMPUTE :
    begin
```

Quick review of Flip-flop coding

With or without an asynchronous Reset_b

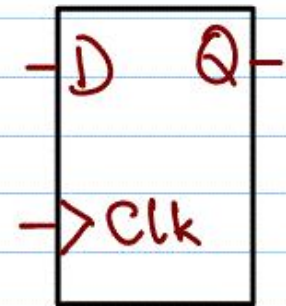
With or without a synchronous Reset_b

With or without a data-enable (de) control

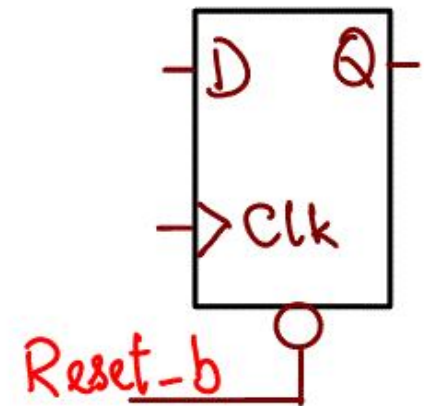
```
always @ (posedge Clk , negedge Reset_b)
```

```
Q    <=    D;
```

```
always @ (posedge Clk)
begin
  Q <= D;
end
```



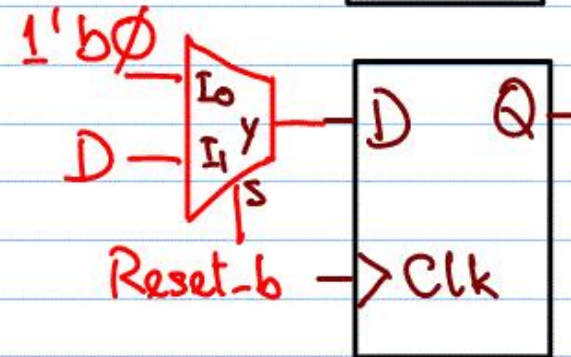
```
always @ (posedge Clk, negedge Reset_b)
begin
  if (!Reset_b)
    Q <= 1'b0;
  else
    Q <= D;
end
```



```

always @ (posedge Clk)
begin
    if (!Reset_b)
        Q <= 1'b0;
    else
        Q <= D;
end

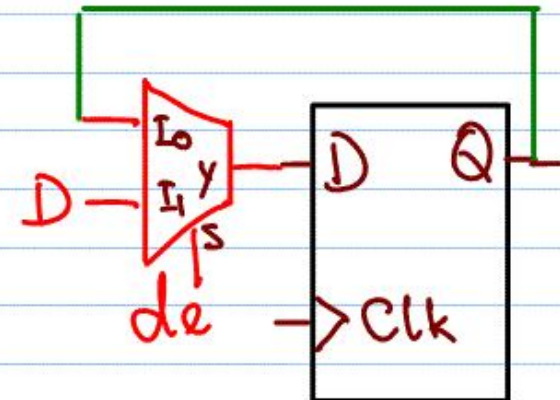
```



```

always @ (posedge Clk)
begin
    if (de)
        Q <= D;
end

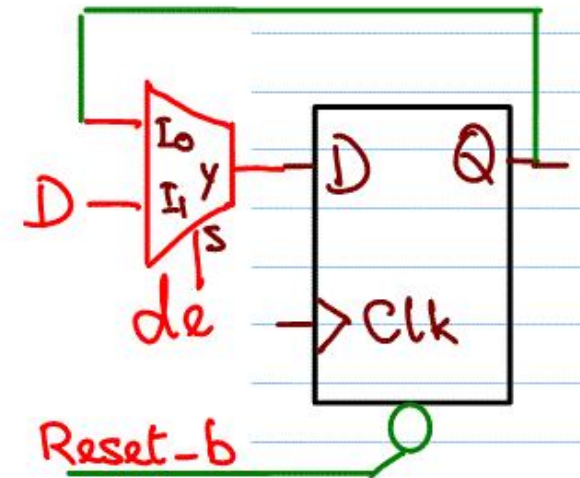
```



```

always @(posedge Clk, negedge Reset_b)
begin
    if (!Reset_b)
        Q <= 1'b0;
    else if (de)
        Q <= D;
end

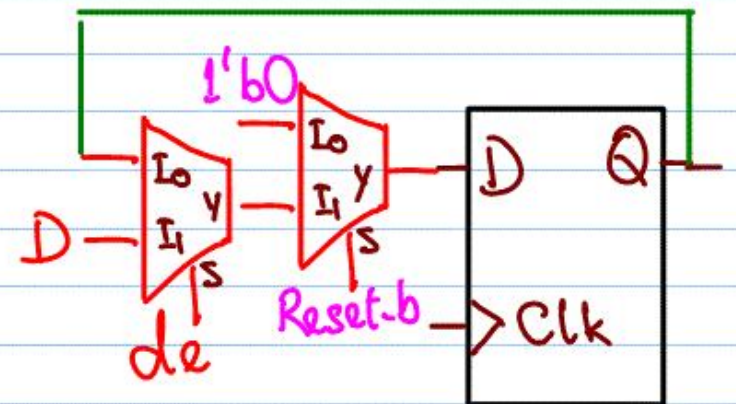
```



```

always @(posedge Clk)
begin
    if (!Reset_b)
        Q <= 1'b0;
    else if (de)
        Q <= D;
end

```



While the previous 6 codings are the most common, consider the following!

```
always @(posedge Clk, negedge Reset_b)
begin
    if (!Reset_b)
        ; // nothing to do
    else if (de)
        Q <= D;
end
```

Nothing

asynchronous
what?

```
always @(posedge Clk, negedge Reset_b)
begin
    if (!Reset_b)
        ; // nothing to do
    else if (de)
        Q <= D;
end
```

Nothing

synchronous
what?

While the previous 6 codings are the most common, consider the following!

```
always @(posedge Clk, negedge Reset_b)
begin
    if (!Reset_b)
        ; // nothing to do
    else
        Q <= D;
end
```

```
always @(posedge Clk, negedge Reset_b)
begin
    if (!Reset_b)
        ; // nothing to do
    else
        Q <= D;
end
```

Both are preventing
synchronous action

clk → ⊕
D → ⊕

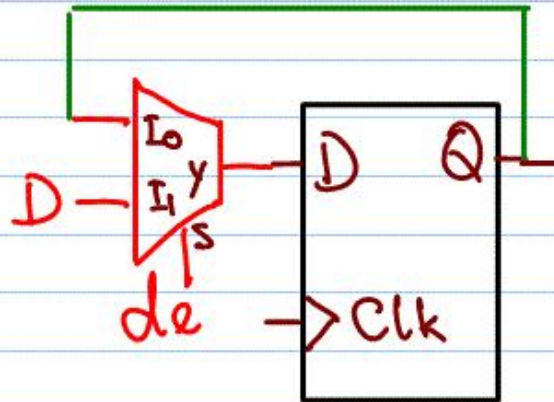
preventing Q from being updated

While the previous 6 codings are the most common, consider the following!

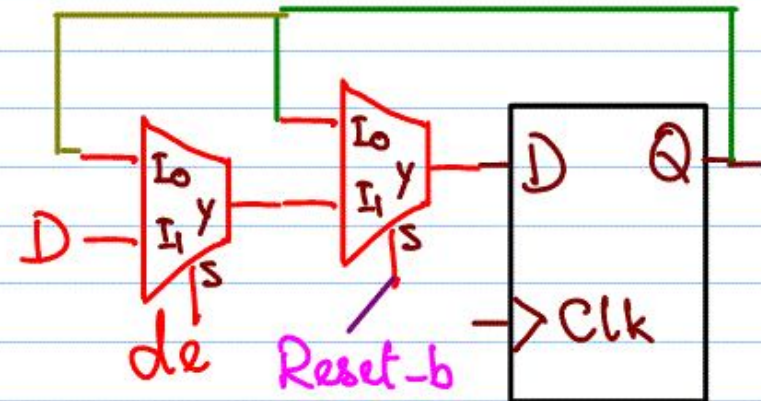
```
always @(posedge Clk, negedge Reset_b)
begin
    if (!Reset_b)
        ; // nothing to do
    else if (de)
        Q <= D;
end
```

do nothing to Q
≡ retain its value

Nothing



What you wanted



What you got

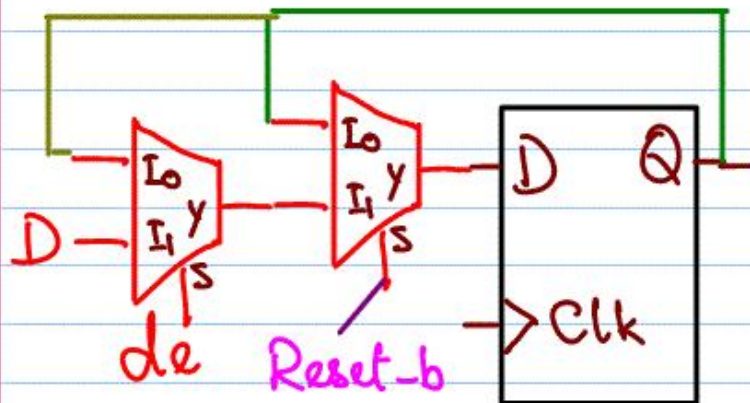
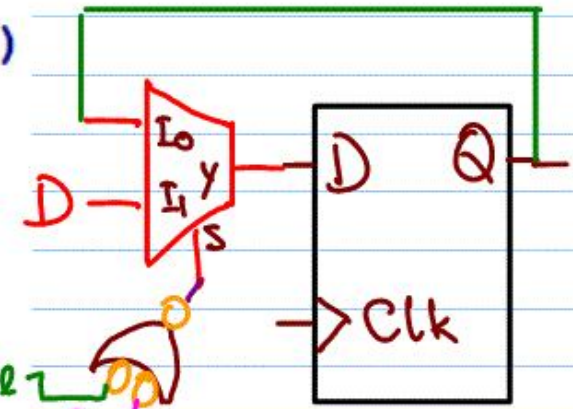
```

always @(posedge Clk, negedge Reset_b)
begin
    if (!Reset_b)
        ; // nothing to do
    else if (de)
        Q <= D;
end

```

Nothing

Reset-b

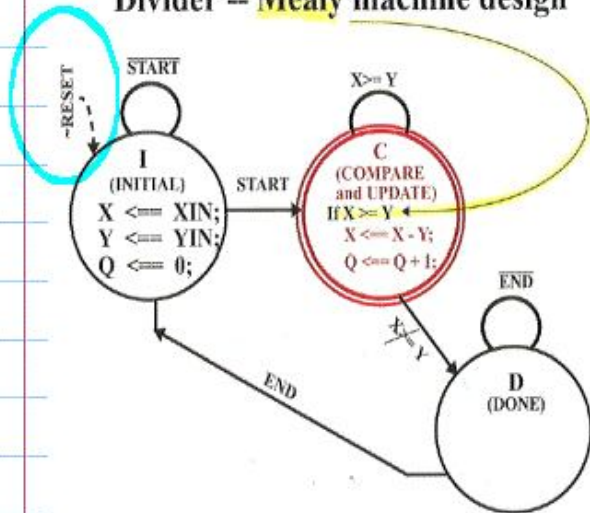


No one codes like this deliberately, but ...

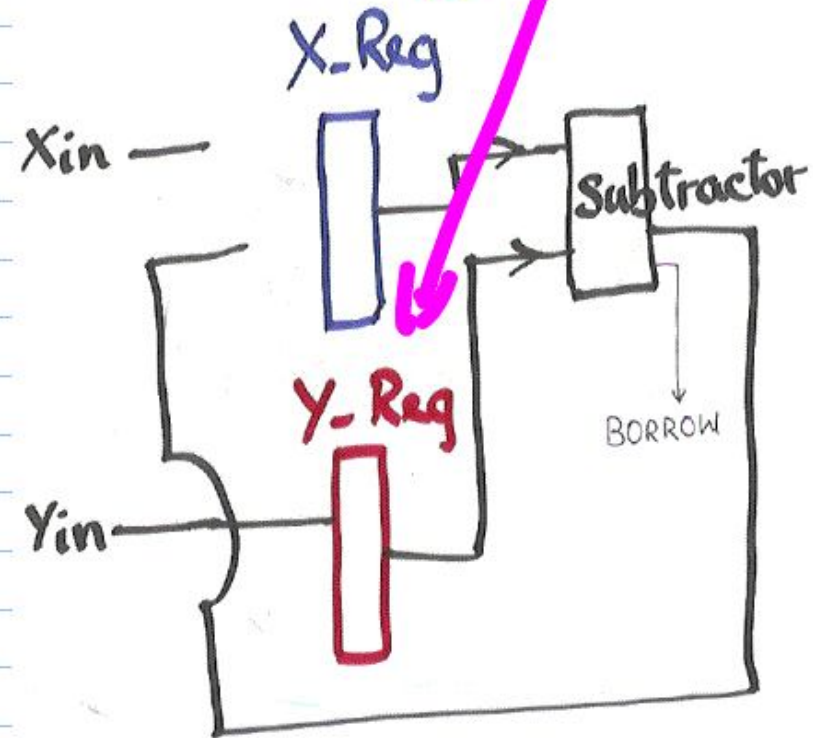
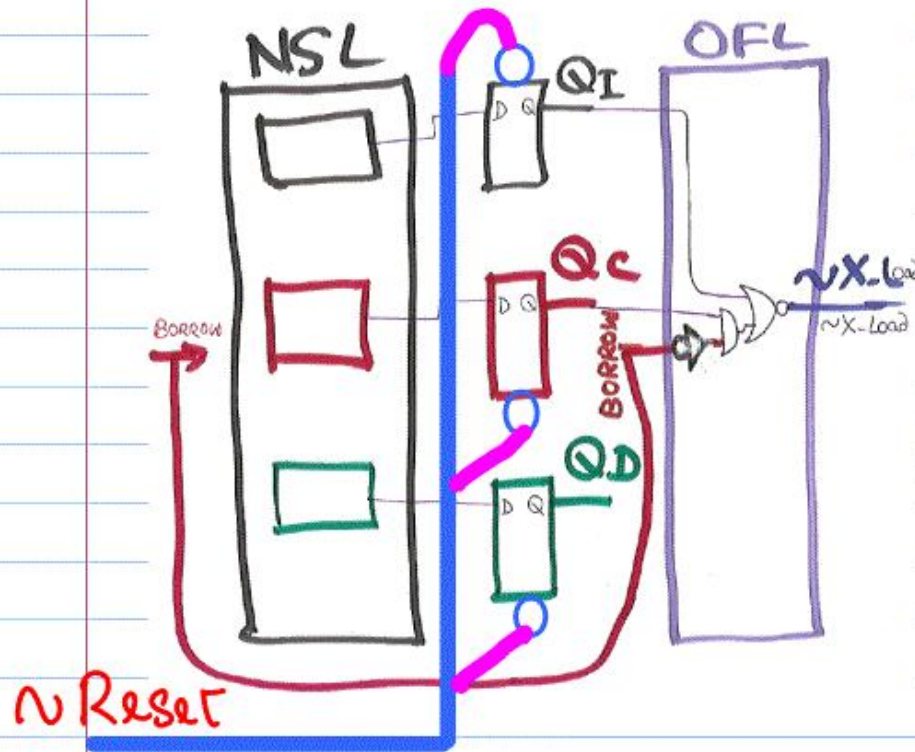
Who needs an asynchronous reset?

Control registers (State Memory Flip-flops,
Program Counter)
or data registers too?

Divider – Mealy machine design



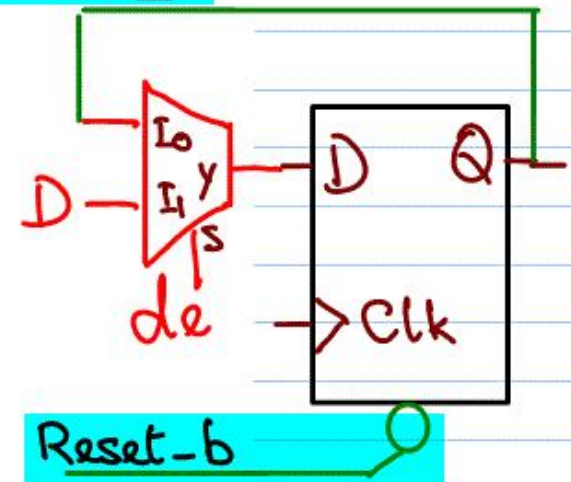
No asynchronous
reset for registers
in datapath //



Then code **CU** (control unit) separately and **DPU** (data path unit) separately!

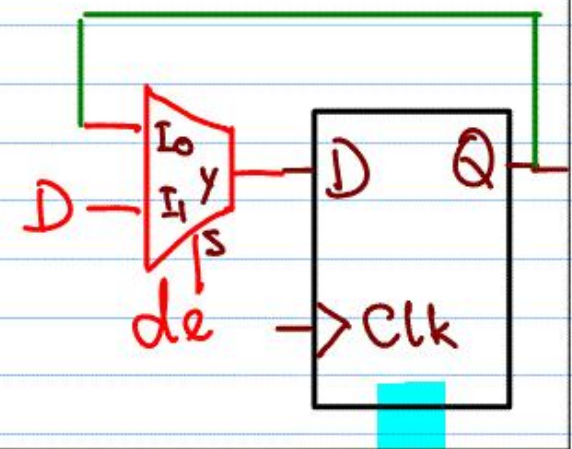
```
always @(posedge Clk, negedge Reset_b)
begin
    if (!Reset_b)
        Q <= 1'b0;
    else if (de)
        Q <= D;
end
```

CU



```
always @(posedge Clk)
begin
    if (de)
        Q <= D;
end
```

DPU



CU

CONTROL_UNIT

```

always @(posedge Clk, posedge Reset)

begin : CONTROL_UNIT
    if (Reset)
        begin
            state <= INITIAL;
        end
    else
        begin
            (* full_case, parallel_case *)
            case (state)
                INITIAL:
                    begin
                        if (Start)
                            state <= COMPUTE;
                        end
                COMPUTE:
                    begin
                        if (X < Y)
                            state <= DONE_S;
                        end
                DONE_S:
                    begin
                        if (Ack)
                            state <= INITIAL;
                        end
            endcase
        end
    end
end // CONTROL_UNIT

```

DPU

DATA_PATH_UNIT

```

always @(posedge Clk)

begin : DATA_PATH_UNIT
    (* full_case, parallel_case *)
    case (state)
        INITIAL:
            begin
                X <= Xin;
                Y <= Yin;
                Quotient <= 0;
            end
        COMPUTE:
            begin
                if (!(X < Y))
                    begin
                        X <= X - Y;
                        Quotient <= Quotient + 1;
                    end
            end
        DONE_S:
            begin // it is best to have "begin"
            end // and "end" so that code
                // can be added later
    endcase
end // DATA_PATH_UNIT

```

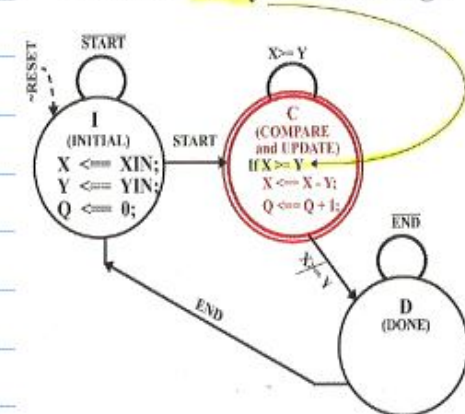
We call the above coding a
"Technician's way of coding" ! Imagine

Imagine ... Instead of this →
 you are given these two:

CU

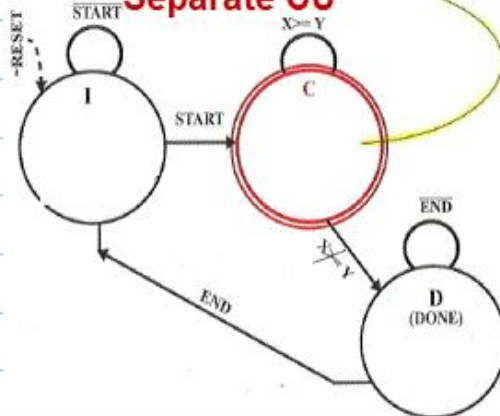
DPU

Divider -- Mealy machine design



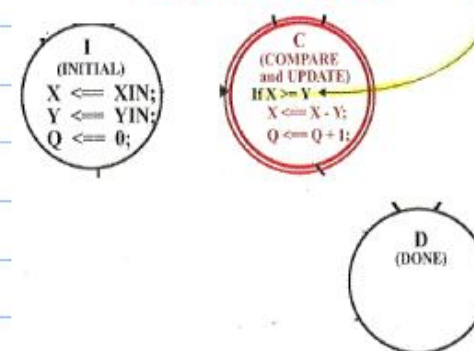
Divider -- Mealy machine design

Separate CU



Divider -- Mealy machine design

Separate DPU



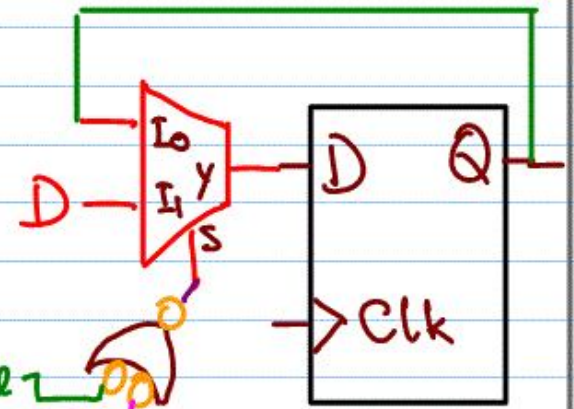
```

always @(posedge Clk, negedge Reset_b)
begin
    if (!Reset_b)
        ; // nothing to do
    else if (de)
        Q <= D;
end

```

Nothing

Reset-b



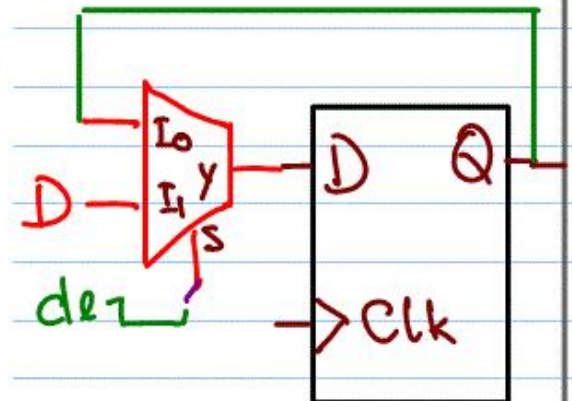
```

always @(posedge Clk, negedge Reset_b)
begin
    if (!Reset_b)
        Q <= 1'bX;
    // avoids recirculating mux
    else if (de)
        Q <= D;
end

```

FIXED

avoided the mux



Our method is to code **CU** (control unit) and **DPU** (data path unit) together!

```
always @(posedge Clk, posedge Reset)

begin : CU_n_DU // Control Unit and Data Path Unit combined
  if (Reset)
  begin
    state <= INITIAL;
    X <= 4'bXXXX; // to avoid recirculating mux controlled by Reset
    Y <= 4'bXXXX;
    Quotient <= 4'bXXXX;
  end
else
```

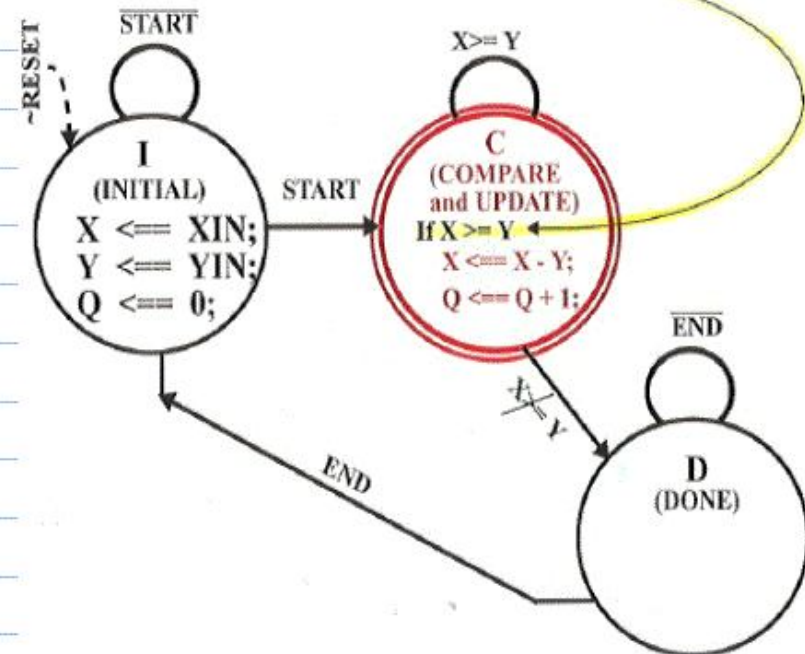


Please refer to
[EE201L_RTL_coding_style_verilog.pdf](#)
Go through the divider coding example.

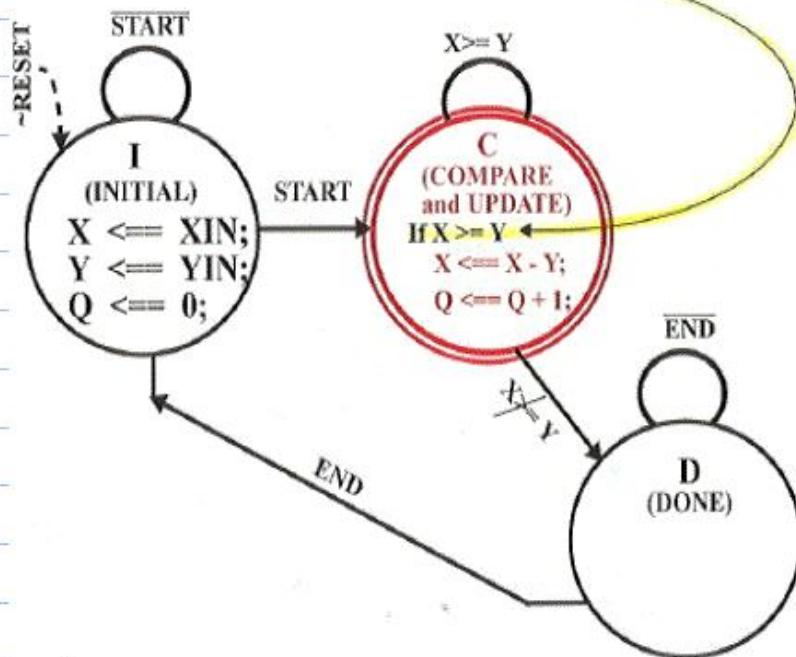
```
always @(posedge Clk, posedge Reset)
```

```
begin : CU_n_DU
  if (Reset)
    begin
      end
    else
      begin
        case (state)
          INITIAL :
            begin
              end
          COMPUTE :
            begin
              end
          DONE_S :
            begin
              end
        endcase
      end
    end
end
```

Divider -- Mealy machine design



Divider -- Mealy machine design



COMPUTE :

begin

// state transitions in the control unit

// RTL operations in the Data Path

end

COMPUTE :

begin

// state transitions in the control unit

if (X < Y)

state <= DONE_S;

// RTL operations in the Data Path

if (!(X < Y))

begin

X <= X - Y;

Quotient <= Quotient + 1;

end

end

Enjoy coding in Verilog RTL!

1. Code your design (design.v)
2. Write a testbench (design_tb.v)
3. Simulate and verify
4. Save the waveform format into design_wave.do

EE201L
EE457
EE560

5. Create a desgin.do file, which
compiles (vlog ...),
starts simulation (vsim ...)
executes the wave.do file (do ...) and
runs the simulation (run 1us)

6. Write the top file (design_top.v)
7. Write the constraints file (design.ucf)
8. Synthesize the design
9. Download the .bit file the FPGA board
10. Verify

EE201L
EE560

11 Celebrate