



IIT KHARAGPUR

NPTEL ONLINE
CERTIFICATION COURSES

NPTEL

Lecture 25: DATAPATH AND CONTROLLER DESIGN (PART 1)

PROF. INDRANIL SENGUPTA

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Introduction

- In a complex digital system, the hardware is typically partitioned into two parts:
 - a) *Data Path*, which consists of the functional units where all computations are carried out.
 - Typically consists of registers, multiplexers, bus, adders, multipliers, counters, and other functional blocks.
 - b) *Control Path*, which implements a finite-state machine and provides control signals to the data path in proper sequence.
 - In response to the control signals, various operations are carried out by the data path.
 - Also takes inputs from the data path regarding various status information.



IIT KHARAGPUR

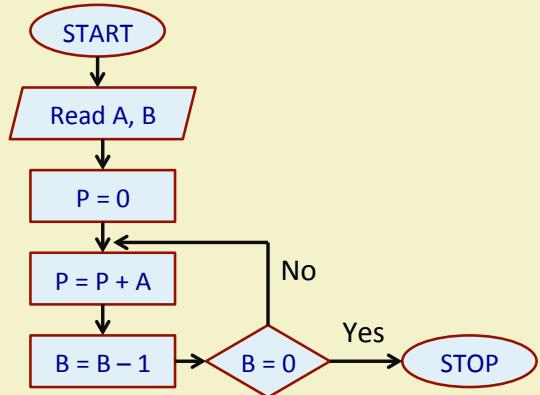
NPTEL ONLINE
CERTIFICATION COURSES

Hardware Modeling Using Verilog

2

Example 1: Multiplication by Repeated Addition

- We consider a simple algorithm using repeated addition.
 - Assume B is non-zero.
- We identify the functional blocks required in the data path, and the corresponding control signals.
- Then we design the FSM to implement the multiplication algorithm using the data path.

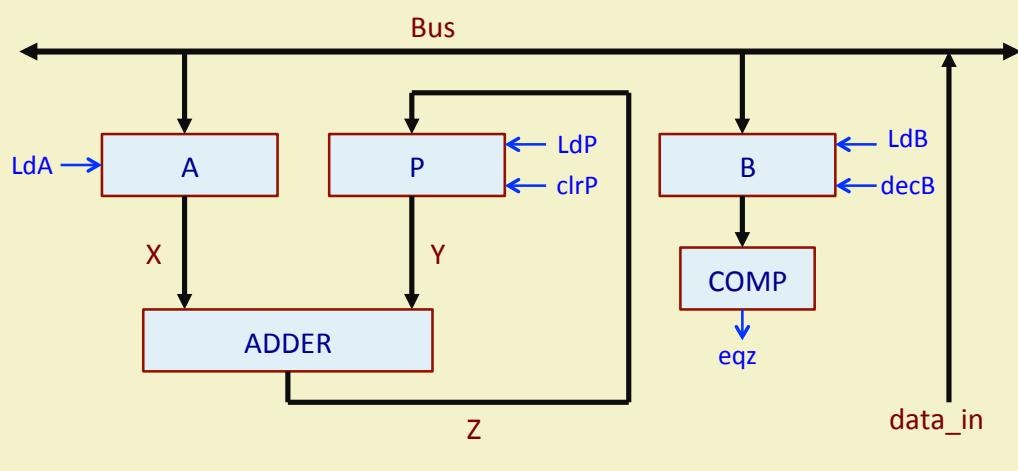


IIT KHARAGPUR

NPTEL
ONLINE
CERTIFICATION COURSES

Hardware Modeling Using Verilog

3

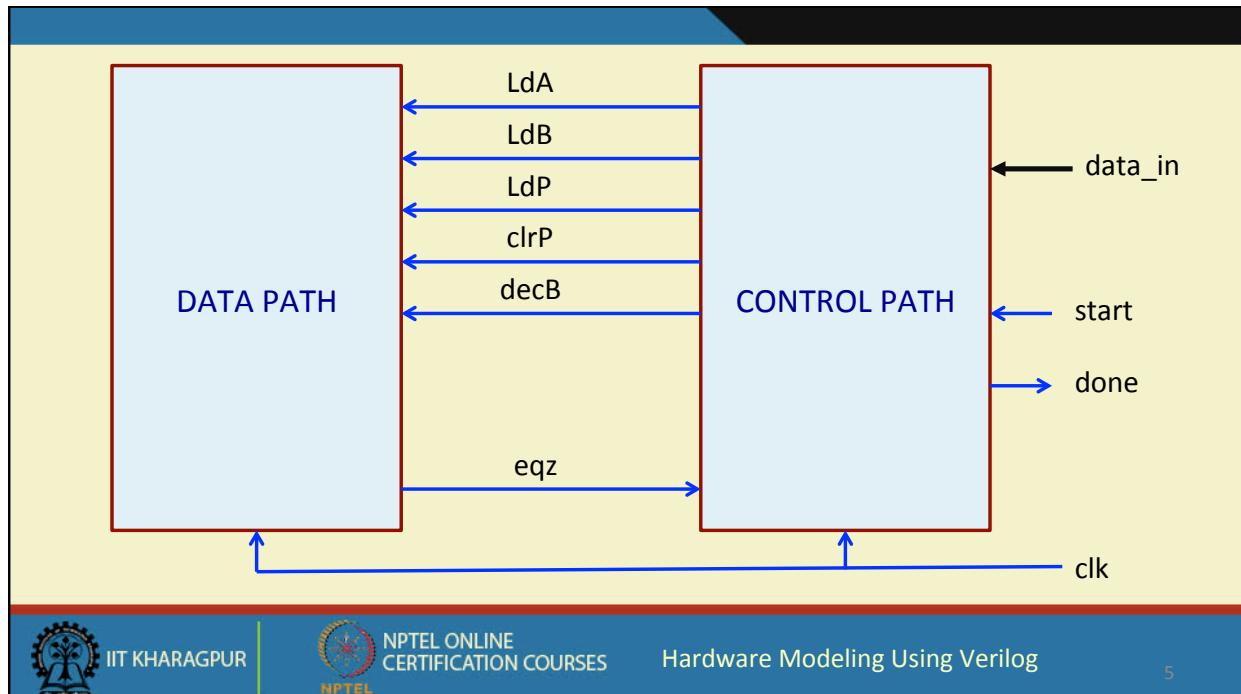


IIT KHARAGPUR

NPTEL
ONLINE
CERTIFICATION COURSES

Hardware Modeling Using Verilog

4

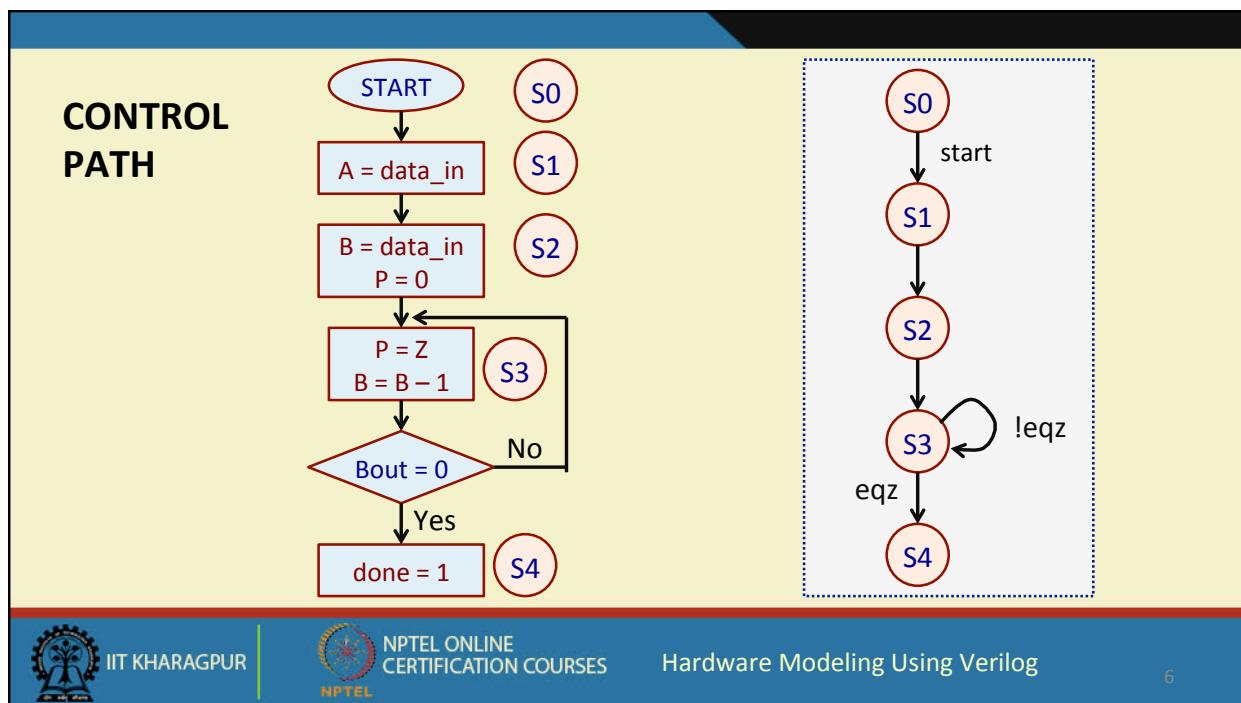


IIT KHARAGPUR

NPTEL ONLINE
CERTIFICATION COURSES

Hardware Modeling Using Verilog

5



IIT KHARAGPUR

NPTEL ONLINE
CERTIFICATION COURSES

Hardware Modeling Using Verilog

6

```

module MUL_datapath (eqz, LdA, LdB, LdP, clrP, decB, data_in, clk);
    input LdA, LdB, LdP, clrP, decB, clk;
    input [15:0] data_in;
    output eqz;
    wire [15:0] X, Y, Z, Bout, Bus;

    PIPO1 A (X, Bus, LdA, clk);
    PIPO2 P (Y, Z, LdB, clrP, clk);
    CNTR B (Bout, Bus, LdB, decB, clk);
    ADD AD (Z, X, Y);
    EQZ COMP (eqz, Bout);
endmodule

```

THE DATA PATH



IIT KHARAGPUR

NPTEL ONLINE
CERTIFICATION COURSES

Hardware Modeling Using Verilog

7

```

module PIPO1 (dout, din, ld, clk);
    input [15:0] din;
    input ld, clk;
    output reg [15:0] dout;
    always @(posedge clk)
        if (ld) dout <= din;
endmodule

module ADD (out, in1, in2);
    input [15:0] in1, in2;
    output reg [15:0] out;
    always @(*)
        out = in1 + in2;
endmodule

```

```

module PIPO2 (dout, din, ld,
              clr, clk);
    input [15:0] din;
    input ld, clr, clk;
    output reg [15:0] dout;
    always @(posedge clk)
        if (clr) dout <= 16'b0;
        else if (ld) dout <= din;
endmodule

module EQZ (eqz, data);
    input [15:0] data;
    output eqz;
    assign eqz = (data == 0);
endmodule

```



IIT KHARAGPUR

NPTEL ONLINE
CERTIFICATION COURSES

Hardware Modeling Using Verilog

8

```
module CNTR (dout, din, ld, dec, clk);
    input [15:0] din;
    input ld, dec, clk;
    output reg [15:0] dout;
    always @ (posedge clk)
        if (ld) dout <= din;
        else if (dec) dout <= dout - 1;
endmodule
```



```
module controller (LdA, LdB, LdP, clrP, decB, done, clk, eqz, start);
    input clk, eqz, start;
    output reg LdA, LdB, LdP, clrP, decB, done;
    reg [2:0] state;
    parameter S0=3'b000, S1=3'b001, S2=3'b010, S3=3'b011, S4=3'b100;
    always @ (posedge clk)
        begin
            case (state)
                S0:   if (start) state <= S1;
                S1:   state <= S2;
                S2:   state <= S3;
                S3:   #2 if (eqz) state <= S4;
                S4:   state <= S4;
                default: state <= S0;
            endcase
        end
end
```

**THE CONTROL
PATH**



```

always @(state)
begin
  case (state)
    S0: begin #1 LdA = 0; LdB = 0; LdP = 0; clrP = 0; decB = 0; end
    S1: begin #1 LdA = 1; end
    S2: begin #1 LdA = 0; LdB = 1; clrP = 1; end
    S3: begin #1 LdB = 0; LdP = 1; clrP = 0; decB = 1; end
    S4: begin #1 done = 1; LdB = 0; LdP = 0; decB = 0; end
  default: begin #1 LdA = 0; LdB = 0; LdP = 0; clrP = 0; decB = 0; end
  endcase
end
endmodule

```



```

module MUL_test;
  reg [15:0] data_in;
  reg clk, start;
  wire done;

  MUL_datapath DP (eqz, LdA, LdB, LdP, clrP, decB, data_in, clk);
  controller CON (LdA, LdB, LdP, clrP, decB, done, clk, eqz, start);

  initial
    begin
      clk = 1'b0;
      #3 start = 1'b1;
      #500 $finish;
    end

  always #5 clk = ~clk;

```

THE TEST BENCH

```

initial
begin
  #17 data_in = 17;
  #10 data_in = 5;
end

initial
begin
  $monitor ($time, " %d %b", DP.Y, done);
  $dumpfile ("mul.vcd"); $dumpvars (0, MUL_test);
end

```

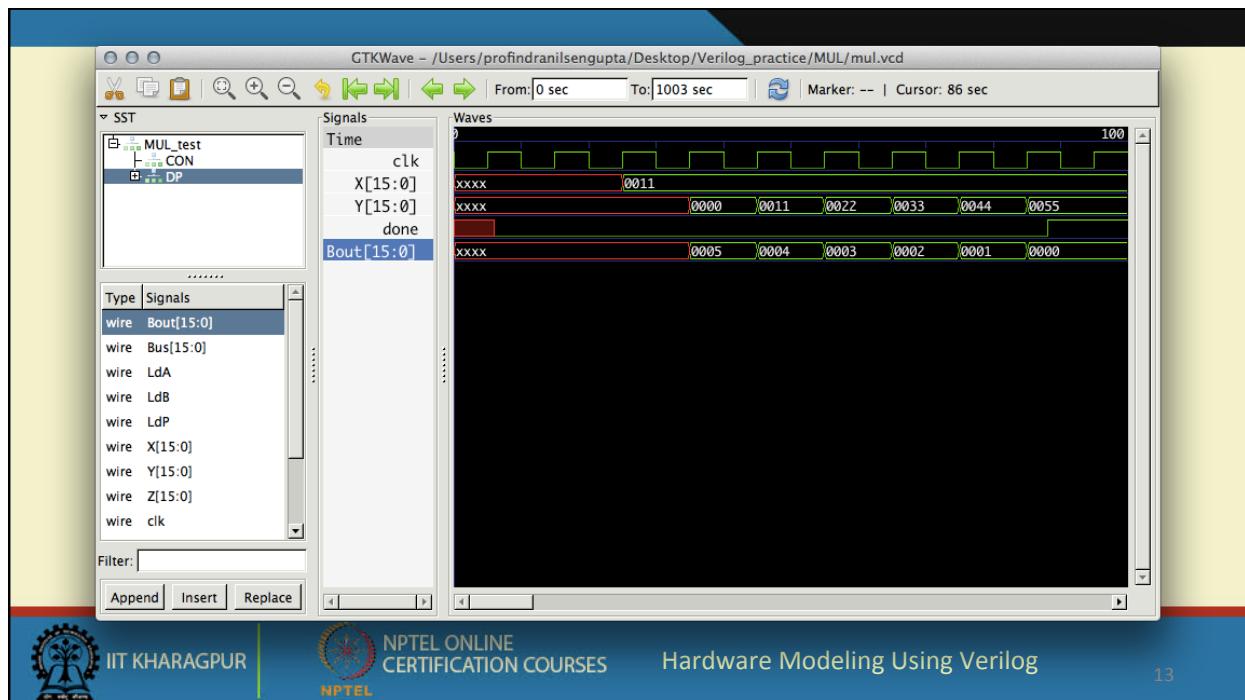
```

endmodule

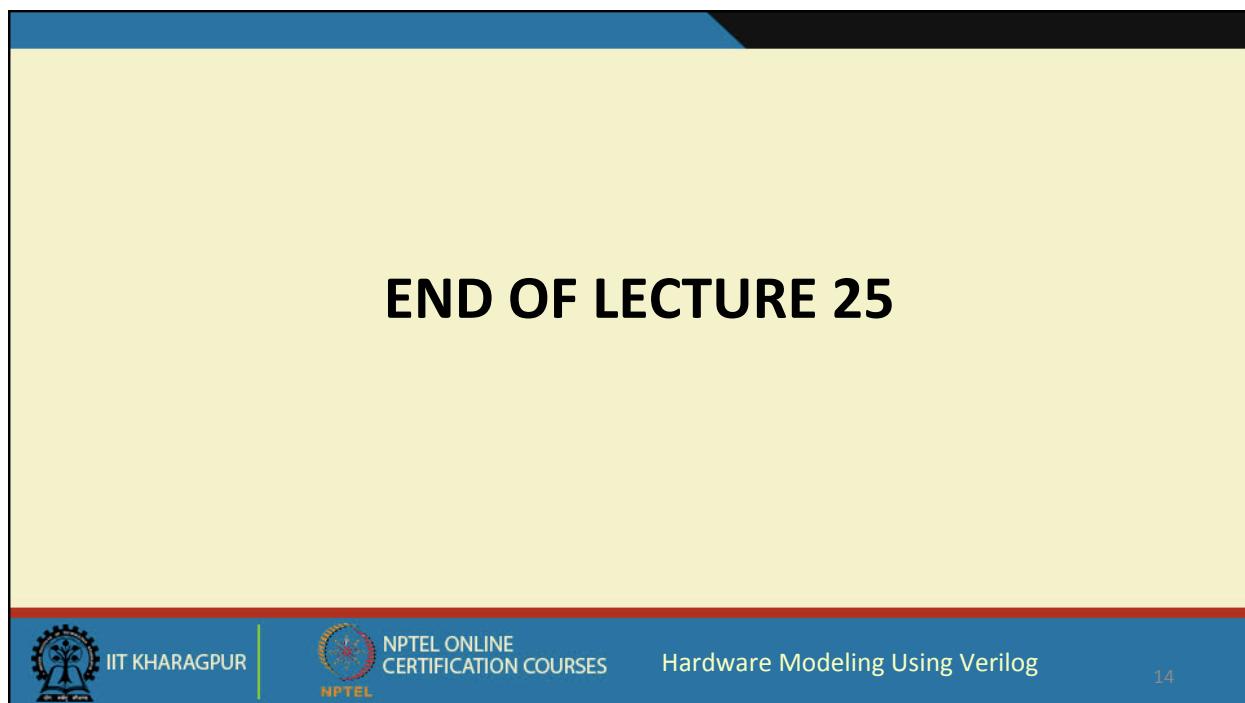
```

| | | |
|----|----|---|
| 0 | x | x |
| 6 | x | 0 |
| 35 | 0 | 0 |
| 45 | 17 | 0 |
| 55 | 34 | 0 |
| 65 | 51 | 0 |
| 75 | 68 | 0 |
| 85 | 85 | 0 |
| 88 | 85 | 1 |





END OF LECTURE 25





IIT KHARAGPUR

NPTEL ONLINE
CERTIFICATION COURSES

NPTEL

Lecture 26: DATAPATH AND CONTROLLER DESIGN (PART 2)

PROF. INDRANIL SENGUPTA

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

A Better Style of Modeling Data / Control Path

- In the previous example, in the “*always*” block activated by clock edge, both state change as well as computation of the next state is performed.
- A better and recommended approach:
 - Only trigger the state change in the clock activated “*always*” block.
 - In a separate “*always*” block using blocking assignments, compute the next state.
 - As in the previous example, in a separate “*always*” block, generate the control signals for the data path.



IIT KHARAGPUR

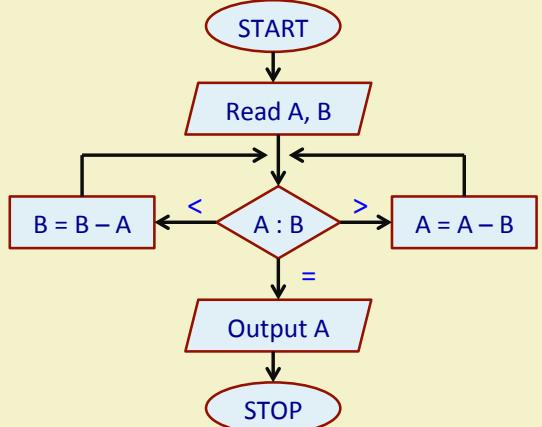
NPTEL ONLINE
CERTIFICATION COURSES

Hardware Modeling Using Verilog

16

Example 2: GCD Computation

- We consider a simple algorithm using repeated subtraction.
- We identify the functional blocks required in the data path, and the corresponding control signals.
- Then we design the FSM to implement the GCD computation algorithm using the data path.



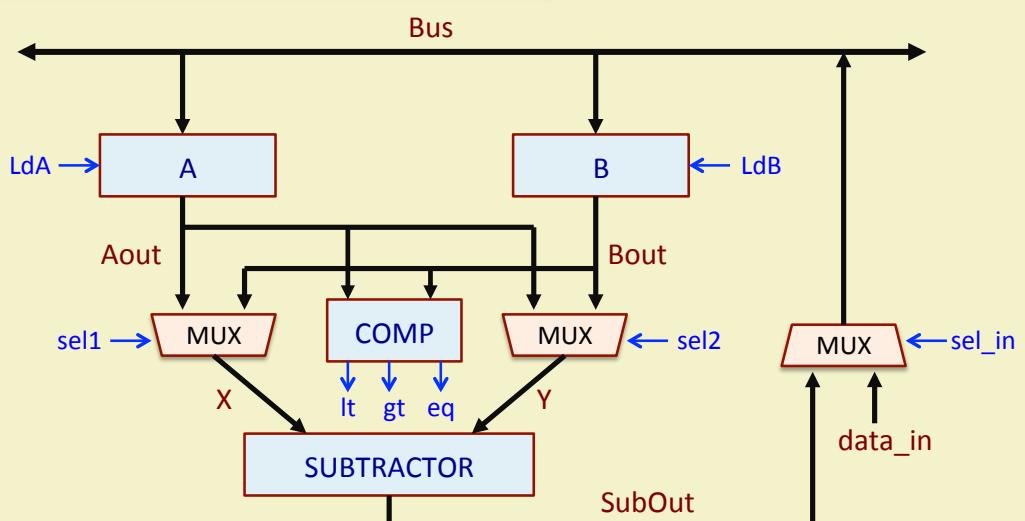
IIT KHARAGPUR

NPTEL ONLINE
CERTIFICATION COURSES

Hardware Modeling Using Verilog

17

**DATA
PATH**

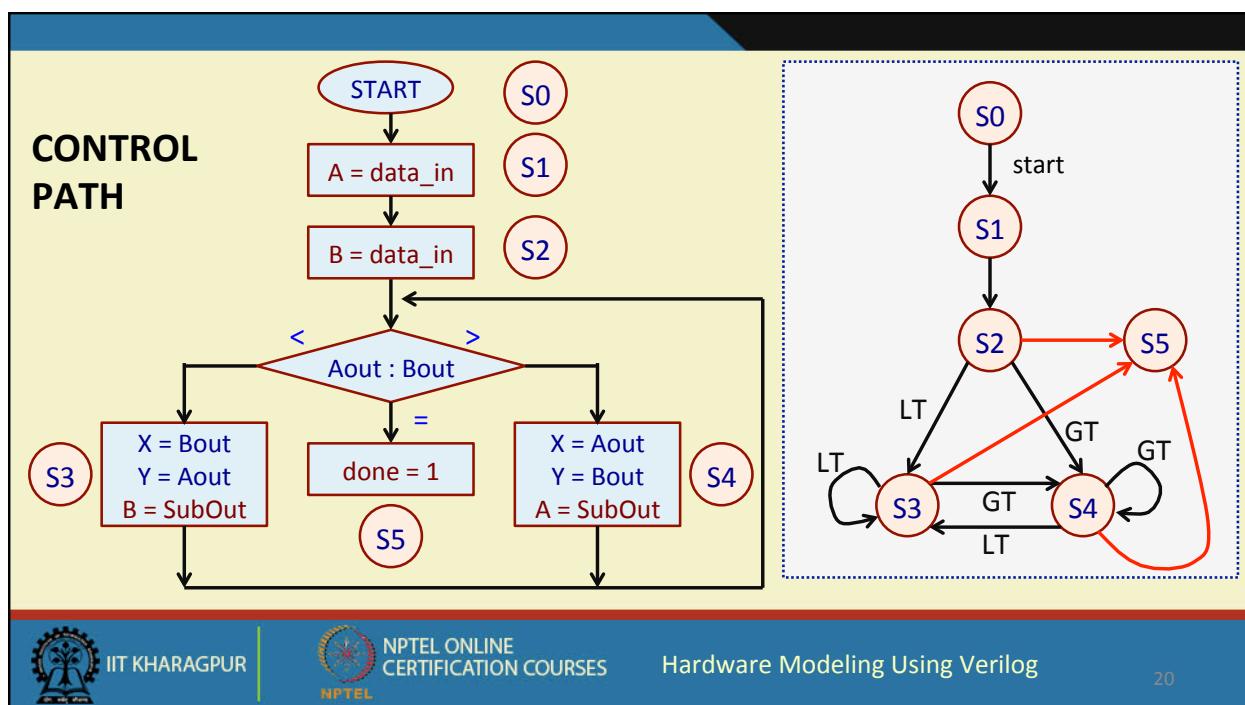
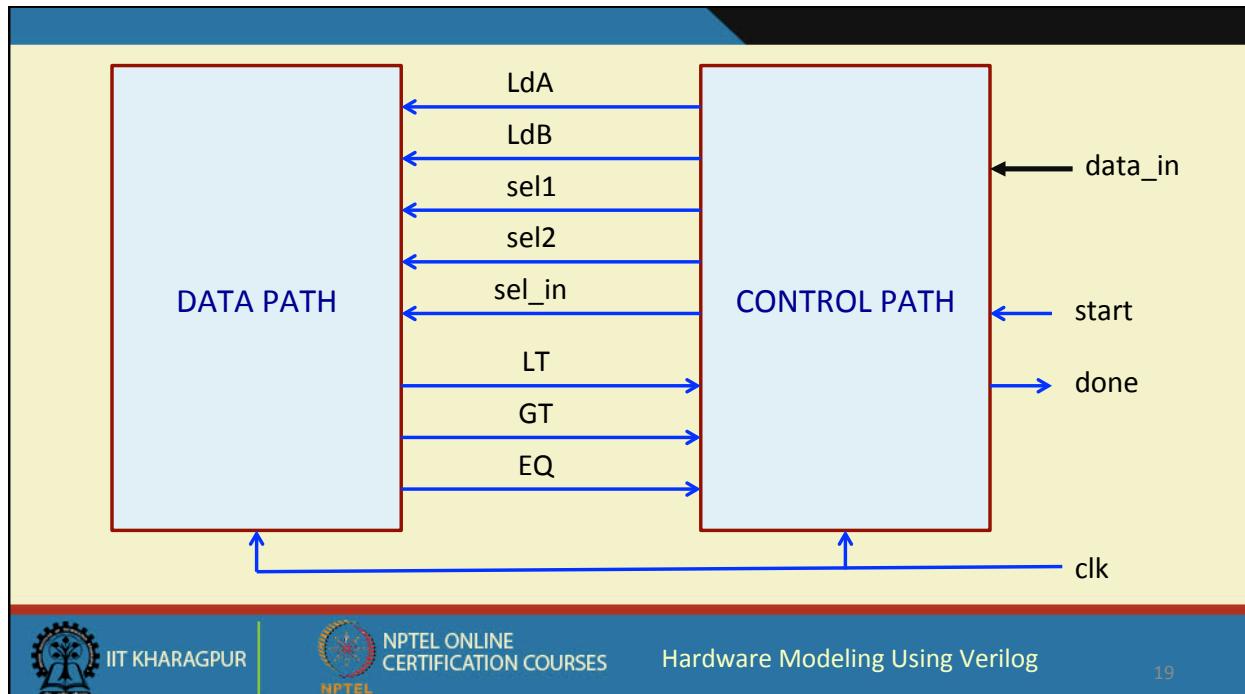


IIT KHARAGPUR

NPTEL ONLINE
CERTIFICATION COURSES

Hardware Modeling Using Verilog

18



```

module GCD_datapath (gt, lt, eq, ldA, ldB, sel1, sel2, sel_in,
data_in, clk);
    input ldA, ldB, sel1, sel2, sel_in, clk;
    input [15:0] data_in;
    output gt, lt, eq;
    wire [15:0] Aout, Bout, X, Y, Bus, SubOut;

    PIP0 A (Aout, Bus, ldA, clk);
    PIP0 B (Bout, Bus, ldB, clk);
    MUX MUX_in1 (X, Aout, Bout, sel1);
    MUX MUX_in2 (Y, Aout, Bout, sel2);
    MUX MUX_load (Bus, SubOut, data_in, sel_in);
    SUB SB (SubOut, X, Y);
    COMPARE COMP (lt, gt, eq, Aout, Bout);
endmodule

```

THE DATA PATH



IIT KHARAGPUR

NPTEL ONLINE
CERTIFICATION COURSES

Hardware Modeling Using Verilog

21

```

module PIP0 (data_out, data_in,
             load, clk);
    input [15:0] data_in;
    input load, clk;
    output reg [15:0] data_out;
    always @ (posedge clk)
        if (load) data_out <= data_in;
endmodule

module SUB (out, in1, in2);
    input [15:0] in1, in2;
    output reg [15:0] out;
    always @(*)
        out = in1 - in2;
endmodule

```

```

module COMPARE (lt, gt, eq, data1,
                data2);
    input [15:0] data1, data2;
    output lt, gt, eq;
    assign lt = data1 < data2;
    assign gt = data1 > data2;
    assign eq = data1 == data2;
endmodule

module MUX (out, in0, in1, sel);
    input [15:0] in0, in1;
    input sel;
    output [15:0] out;
    assign out = sel ? in1 : in0;
endmodule

```



IIT KHARAGPUR

NPTEL ONLINE
CERTIFICATION COURSES

Hardware Modeling Using Verilog

22

```

module controller (ldA, ldB, sel1, sel2, sel_in, done, clk, lt, gt, eq, start);
  input clk, lt, gt, eq, start;
  output reg ldA, ldB, sel1, sel2, sel_in, done;
  reg [2:0] state;
  parameter S0=3'b000, S1=3'b001, S2=3'b010, S3=3'b011, S4=3'b100, S5=3'b101;
  always @(posedge clk)
    begin
      case (state)
        S0: if (start) state <= S1;
        S1: state <= S2;
        S2: #2 if (eq) state <= S5;
              else if (lt) state <= S3;
              else if (gt) state <= S4;
        S3: #2 if (eq) state <= S5;
              else if (lt) state <= S3;
              else if (gt) state <= S4;
        S4: #2 if (eq) state <= S5;
              else if (lt) state <= S3;
              else if (gt) state <= S4;
        S5: state <= S5;
        default: state <= S0;
      endcase
    end

```

THE CONTROL PATH

```

always @(state)
begin
  case (state)
    S0: begin sel_in = 1; ldA = 1; ldB = 0; done = 0; end
    S1: begin sel_in = 1; ldA = 0; ldB = 1; end
    S2: if (eq) done = 1;
         else if (lt) begin
           sel1 = 1; sel2 = 0; sel_in = 0;
           #1 ldA = 0; ldB = 1;
         end
         else if (gt) begin
           sel1 = 0; sel2 = 1; sel_in = 0;
           #1 ldA = 1; ldB = 0;
         end
    S3: if (eq) done = 1;
         else if (lt) begin
           sel1 = 1; sel2 = 0; sel_in = 0;
           #1 ldA = 0; ldB = 1;
         end
         else if (gt) begin
           sel1 = 0; sel2 = 1; sel_in = 0;
           #1 ldA = 1; ldB = 0;
         end
  end
end

```

```

S4:      if (eq) done = 1;
          else if (lt) begin
              sel1 = 1; sel2 = 0; sel_in = 0;
              #1 ldA = 0; ldB = 1;
              end
          else if (gt) begin
              sel1 = 0; sel2 = 1; sel_in = 0;
              #1 ldA = 1; ldB = 0;
              end
      S5:      begin
              done = 1; sel1 = 0; sel2 = 0; ldA = 0;
              ldB = 0;
              end
          default: begin ldA = 0; ldB = 0; end
      endcase
  end

endmodule

```



THE TEST BENCH

```

module GCD_test;
  reg [15:0] data_in;
  reg clk, start;
  wire done;

  reg [15:0] A, B;

  GCD_datapath DP (gt, lt, eq, ldA, ldB, sel1, sel2, sel_in, data_in, clk);
  controller CON (ldA, ldB, sel1, sel2, sel_in, done, clk, lt, gt, eq, start);

  initial
    begin
      clk = 1'b0;
      #3 start = 1'b1;
      #1000 $finish;
    end

  always #5 clk = ~clk;

```



```

initial
begin
#12 data_in = 143;
#10 data_in = 78;
end

initial
begin
$monitor ($time, " %d %b", DP.Aout, done);
$dumpfile ("gcd.vcd"); $dumpvars (0, GCD_test);
end

endmodule

```

| | | |
|----|-----|---|
| 0 | x | x |
| 5 | x | 0 |
| 15 | 143 | 0 |
| 35 | 65 | 0 |
| 55 | 52 | 0 |
| 65 | 39 | 0 |
| 75 | 26 | 0 |
| 85 | 13 | 0 |
| 87 | 13 | 1 |

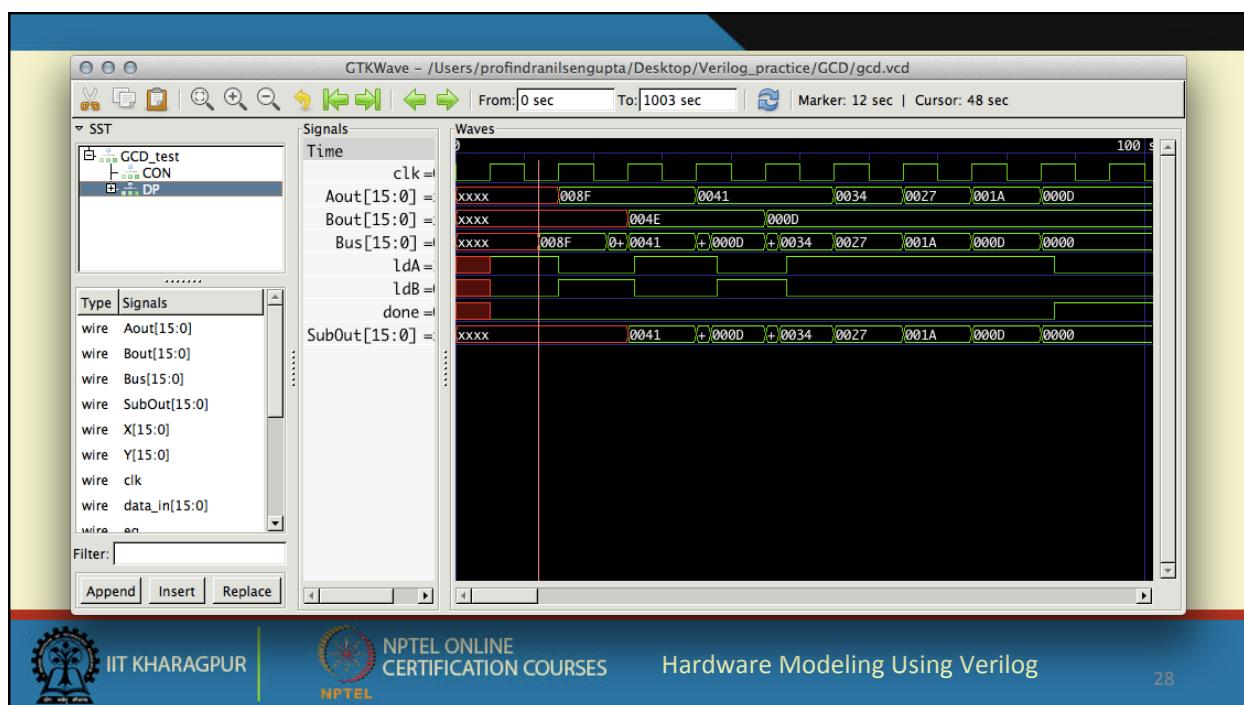


IIT KHARAGPUR

NPTEL ONLINE
CERTIFICATION COURSES

Hardware Modeling Using Verilog

27



IIT KHARAGPUR

NPTEL ONLINE
CERTIFICATION COURSES

Hardware Modeling Using Verilog

28

MODELING THE CONTROL PATH USING THE ALTERNATE APPROACH



IIT KHARAGPUR

NPTEL ONLINE
CERTIFICATION COURSES

Hardware Modeling Using Verilog

29

```
module controller (ldA, ldB, sel1, sel2, sel_in, done, clk, lt, gt, eq, start);
    input clk, lt, gt, eq, start;
    output reg ldA, ldB, sel1, sel2, sel_in, done;
    reg [2:0] state, next_state;
    parameter S0=3'b000, S1=3'b001, S2=3'b010, S3=3'b011, S4=3'b100, S5=3'b101;
    always @ (posedge clk)
        begin
            state <= next_state;
        end

```

**THE CONTROL
PATH**



IIT KHARAGPUR

NPTEL ONLINE
CERTIFICATION COURSES

Hardware Modeling Using Verilog

30

```

always @(state)
begin
  case (state)
    S0: begin sel_in = 1; ldA = 1; ldB = 0; done = 0; end
    S1: begin sel_in = 1; ldA = 0; ldB = 1; end
    S2: if (eq) begin done = 1; next_state = S5; end
         else if (lt) begin
           sel1 = 1; sel2 = 0; sel_in = 0; next_state = S3;
           #1 ldA = 0; ldB = 1;
         end
         else if (gt) begin
           sel1 = 0; sel2 = 1; sel_in = 0; next_state = S4;
           #1 ldA = 1; ldB = 0;
         end
    S3: if (eq) begin done = 1; next_state = S5; end
         else if (lt) begin
           sel1 = 1; sel2 = 0; sel_in = 0; next_state = S3;
           #1 ldA = 0; ldB = 1;
         end
         else if (gt) begin
           sel1 = 0; sel2 = 1; sel_in = 0; next_state = S4;
           #1 ldA = 1; ldB = 0;
         end
    S4: if (eq) begin done = 1; next_state = S5; end
         else if (lt) begin
           sel1 = 1; sel2 = 0; sel_in = 0; next_state = S3;
           #1 ldA = 0; ldB = 1;
         end
         else if (gt) begin
           sel1 = 0; sel2 = 1; sel_in = 0; next_state = S4;
           #1 ldA = 1; ldB = 0;
         end
    S5: begin
      done = 1; sel1 = 0; sel2 = 0; ldA = 0;
      ldB = 0; next_state = S5;
    end
  default: begin ldA = 0; ldB = 0; next_state = S0; end
  endcase
end

```

```

endmodule

```



END OF LECTURE 26



IIT KHARAGPUR



NPTEL
ONLINE
CERTIFICATION COURSES

Hardware Modeling Using Verilog

33



IIT KHARAGPUR



NPTEL
ONLINE
CERTIFICATION COURSES

Lecture 27: DATAPATH AND CONTROLLER DESIGN (PART 3)

PROF. INDRANIL SENGUPTA
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Example 3: Booth's Multiplication

- In the conventional shift-and-add multiplication, for n -bit multiplication, we iterate n times.
 - Add either 0 or the multiplicand to the $2n$ -bit partial product (depending on the next bit of the multiplier).
 - Shift the $2n$ -bit partial product to the right.
- Essentially we need n additions and n shift operations.
- Booth's algorithm is an improvement whereby we can avoid the additions whenever consecutive 0's or 1's are detected in the multiplier.
 - Makes the process faster.



IIT KHARAGPUR

NPTEL ONLINE
CERTIFICATION COURSES

Hardware Modeling Using Verilog

35

Basic Idea Behind Booth's Algorithm

- We inspect two bits of the multiplier (Q_i, Q_{i-1}) at a time.
 - If the bits are same (00 or 11), we only shift the partial product.
 - If the bits are 01, we do an addition and then shift.
 - If the bits are 10, we do a subtraction and then shift.
- Q_{-1} is assumed to be equal to 0.
- Significantly reduces the number of additions / subtractions.

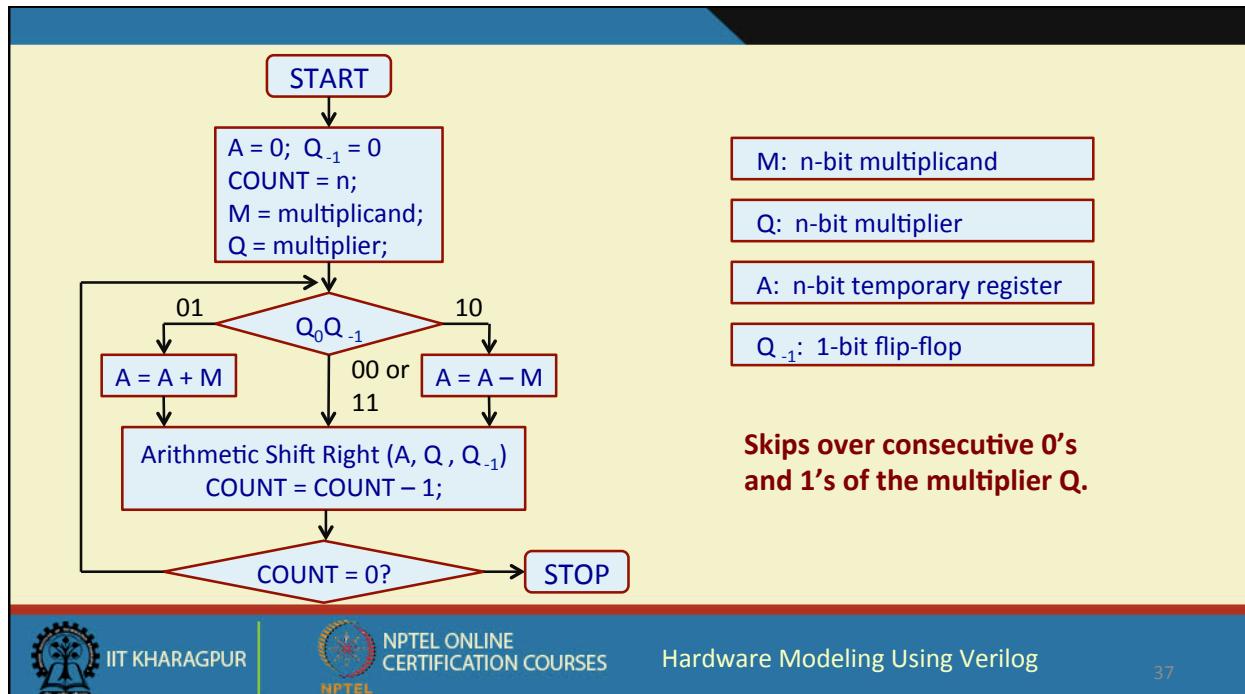


IIT KHARAGPUR

NPTEL ONLINE
CERTIFICATION COURSES

Hardware Modeling Using Verilog

36



IIT KHARAGPUR

NPTEL ONLINE
CERTIFICATION COURSES

Hardware Modeling Using Verilog

37

| A | Q | Q ₋₁ | |
|-----------|---------|-----------------|----------------|
| 0 0 0 0 0 | 0 1 1 0 | 1 0 | Initialization |
| 0 1 0 1 0 | 0 1 1 0 | 1 0 | A = A - M |
| 0 0 1 0 1 | 0 0 1 1 | 0 1 | Shift |
| 1 1 0 1 1 | 0 0 1 1 | 0 1 | A = A + M |
| 1 1 1 0 1 | 1 0 0 1 | 1 0 | Shift |
| 0 0 1 1 1 | 1 0 0 1 | 1 0 | A = A - M |
| 0 0 0 1 1 | 1 1 0 0 | 1 1 | Shift |
| 0 0 0 0 1 | 1 1 1 1 | 0 1 | Shift |
| 1 0 1 1 1 | 1 1 1 0 | 1 | A = A + M |
| 1 1 0 1 1 | 1 1 1 1 | 0 | Shift |

Example 1: (-10) x (13)

Assume 5-bit numbers.

M: $(10110)_2$
-M: $(01010)_2$
Q: $(01101)_2$

Product = -130
= $(1101111110)_2$



IIT KHARAGPUR

NPTEL ONLINE
CERTIFICATION COURSES

Hardware Modeling Using Verilog

38

| A | Q | Q_{-1} | |
|-------------|-------------|----------|--------------------|
| 0 0 0 0 0 0 | 0 1 1 1 0 | 0 0 | Initialization |
| 0 0 0 0 0 0 | 0 0 1 1 1 | 0 0 | Shift Step 1 |
| 0 0 0 0 0 0 | 0 0 0 1 1 | 1 0 | Shift Step 2 |
| 0 1 1 1 1 1 | 0 0 0 1 1 | 1 0 | $A = A - M$ Step 3 |
| 0 0 1 1 1 1 | 1 0 0 0 1 | 1 1 | Shift Step 4 |
| 0 0 0 1 1 1 | 1 1 0 0 0 | 1 1 | Shift Step 5 |
| 0 0 0 0 1 1 | 1 1 1 0 0 | 0 1 | $A = A + M$ Step 6 |
| 1 0 0 1 0 0 | 1 1 1 0 0 0 | 1 | |
| 1 1 0 0 1 0 | 0 1 1 1 0 0 | 0 | Shift |

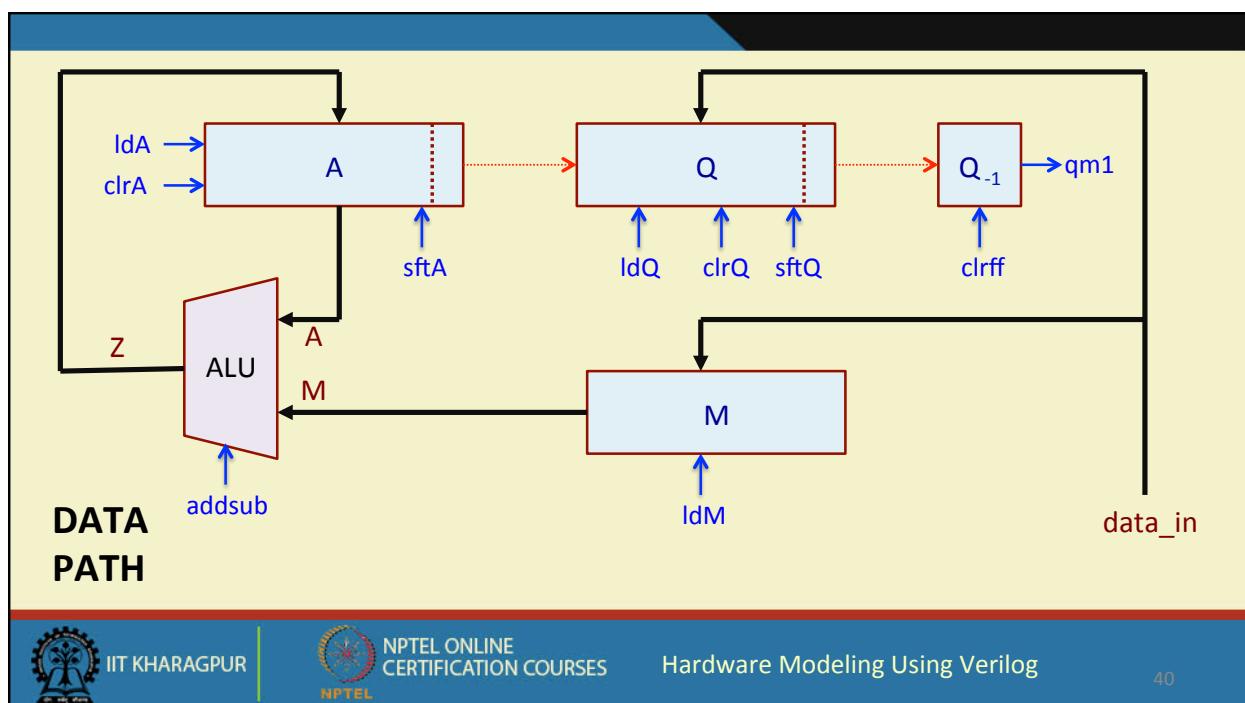


IIT KHARAGPUR

NPTEL
ONLINE
CERTIFICATION COURSES

Hardware Modeling Using Verilog

39

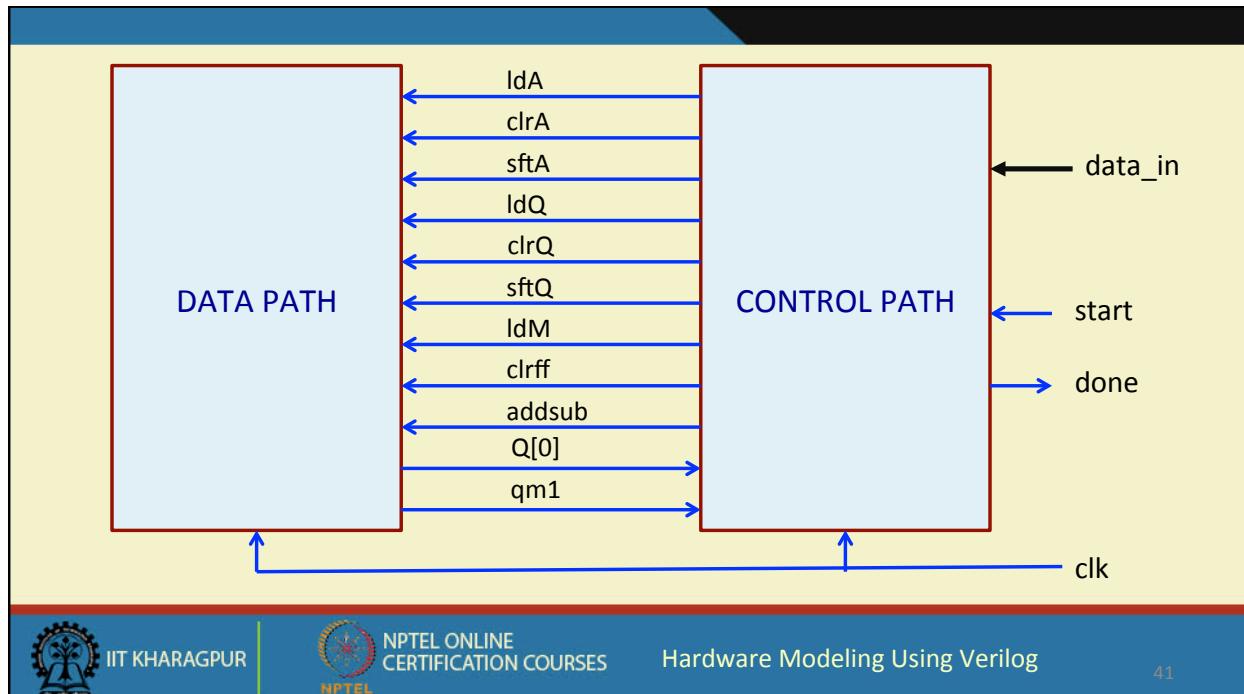


IIT KHARAGPUR

NPTEL
ONLINE
CERTIFICATION COURSES

Hardware Modeling Using Verilog

40

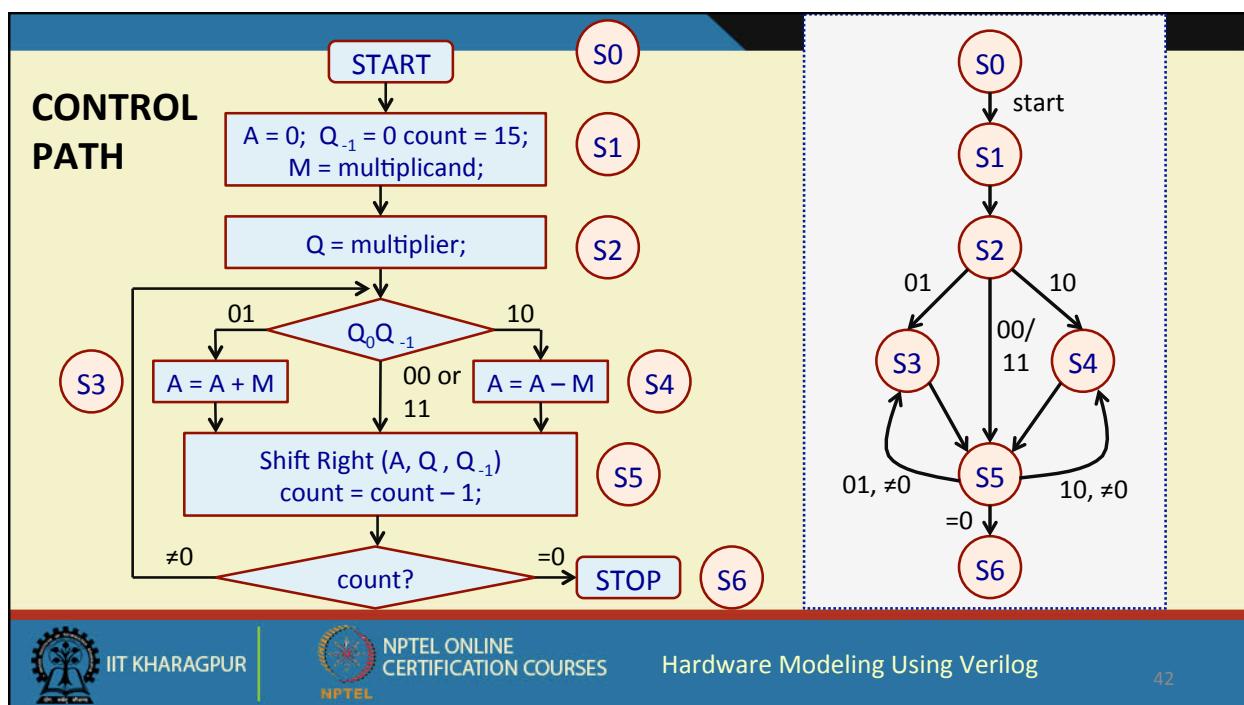


IIT KHARAGPUR

NPTEL ONLINE
CERTIFICATION COURSES

Hardware Modeling Using Verilog

41



IIT KHARAGPUR

NPTEL ONLINE
CERTIFICATION COURSES

Hardware Modeling Using Verilog

42

```

module BOOTH (ldA, ldQ, ldM, clrA, clrQ, clrrff, sftA, sftQ,
              addsub, decr, ldcnt, data_in, clk, qml, eqz);
    input ldA, ldQ, ldM, clrA, clrQ, clrrff, sftA, sftQ, addsub, clk;
    input [15:0] data_in;
    output qml, eqz;
    wire [15:0] A, M, Q, Z;
    wire [4:0] count;

    assign eqz = ~|count;

    shiftreg AR (A, Z, A[15], clk, ldA, clrA, sftA);
    shiftreg QR (Q, data_in, A[0], clk, ldQ, clrQ, sftQ);
    dff QM1 (Q[0], qml, clk, clrrff);
    PIP0 MR (data_in, M, clk, ldM);
    ALU AS (Z, A, M, addsub);
    counter CN (count, decr, ldcnt, clk);
endmodule

```

THE DATA PATH



IIT KHARAGPUR

NPTEL ONLINE
CERTIFICATION COURSES

Hardware Modeling Using Verilog

43

```

module shiftreg (data_out,data_in,
                 s_in, clk, ld, clr, sft);
    input s_in, clk, ld, clr, sft;
    input [15:0] data_in;
    output reg [15:0] data_out;

    always @ (posedge clk)
        begin
            if (clr) data_out <= 0;
            else if (ld)
                data_out <= data_in;
            else if (sft)
                data_out <= {s_in,data_out[15:1]};
            end
        endmodule

```

```

module PIP0 (data_out,data_in, clk, load);
    input [15:0] data_in;
    input load, clk;
    output reg [15:0] data_out;

    always @ (posedge clk)
        if (load) data_out <= data_in;
endmodule

module dff (d, q, clk, clr);
    input d, clk, clr;
    output reg q;

    always @ (posedge clk)
        if (clr) q <= 0;
        else q <= d;
endmodule

```



IIT KHARAGPUR

NPTEL ONLINE
CERTIFICATION COURSES

Hardware Modeling Using Verilog

44

```

module ALU (out, in1, in2, addsub);
    input [15:0] in1, in2;
    input addsub;
    output reg [15:0] out;

    always @(*)
    begin
        if (addsub == 0) out = in1 - in2;
        else out = in1 + in2;
    end
endmodule

```



```

module counter (data_out, decr, ldcnt, clk)
    input decr, clk;
    output [4:0] data_out;

    always @ (posedge clk)
    begin
        if (ldcnt) data_out < 5'b10000;
        else if (decr) data_out <= data_out - 1;
    end
endmodule

```



IIT KHARAGPUR

NPTEL ONLINE
CERTIFICATION COURSES

Hardware Modeling Using Verilog

45

```

module controller (ldA, clrA, sftA, ldQ, clrQ, sftQ, ldM, clrrff, addsub, start,
                  decr, ldcnt, done, clk, q0, qm1);
    input clk, q0, qm1, start;
    output reg ldA, clrA, sftA, ldQ, clrQ, sftQ, ldM, clrrff, addsub, decr, ldcnt, done;
    reg [2:0] state;
    parameter S0=3'b000, S1=3'b001, S2=3'b010, S3=3'b011, S4=3'b100, S5=3'b101, S6=3'b110;
    always @ (posedge clk)
    begin
        case (state)
            S0:   if (start) state <= S1;
            S1:   state <= S2;
            S2:   #2 if ({q0,qm1}==2'b01) state <= S3;
                   else if ({q0,qm1}==1'b10) state <= S4;
                   else state <= S5;
            S3:   state <= S5;
            S4:   state <= S5;
            S5:   #2 if (({q0,qm1}==2'b01) && !eqz) state <= S3;
                   else if (({q0,qm1}==2'b10) && !eqz) state <= S4;
                   else if (eqz) state <= S6;
            S6:   state <= S6;
            default: state <= S0;
        endcase
    end

```

**THE CONTROL
PATH**

```

always @(state)
begin
  case (state)
    S0:   begin clrA = 0; ldA = 0; sftA = 0; clrQ = 0; ldQ = 0; sftQ = 0;
          ldM = 0; clrf = 0; done = 0; end
    S1:   begin clrA = 1; clrf = 1; ldcnt = 1; ldM = 1; end
    S2:   begin clrA = 0; clrf = 0; ldcnt = 0; ldM = 0; ldQ = 1; end
    S3:   begin ldA = 1; addsub = 1; ldQ = 0; sftA = 0; sftQ = 0; decr = 0; end
    S4:   begin ldA = 1; addsub = 0; ldQ = 0; sftA = 0; sftQ = 0; decr = 0; end
    S5:   begin sftA = 1; sftQ = 1; ldA = 0; ldQ = 0; decr = 1; end
    S6:   done = 1;
    default: begin clrA = 0; sftA = 0; ldQ = 0; sftQ = 0; end
  endcase
end

```



- Test bench can be written similarly.
- Points to note:
 - The timing must be very clearly analyzed and signals activated at proper time instances in the test bench.
 - Otherwise, the simulation results will not come correct, though the module descriptions may be fine.
 - This cannot be taught ... requires practice and experience.



END OF LECTURE 27



IIT KHARAGPUR



NPTEL
ONLINE
CERTIFICATION COURSES

Hardware Modeling Using Verilog

49



IIT KHARAGPUR



NPTEL
ONLINE
CERTIFICATION COURSES

Lecture 28: SYNTHESIZABLE VERILOG

PROF. INDRANIL SENGUPTA
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

About the Verilog Language

- The language Verilog has a large number of features, most of which are supported by the simulation tools.
- Unfortunately, several of the language constructs are not supported by synthesis tools.
 - The language subset that can be synthesized is known as "*Synthesizable Verilog*" subset.
- Here we shall state the language features not supported by most of the synthesis tools.
 - Best be avoided if the objective is to map the design to hardware.



IIT KHARAGPUR

NPTEL ONLINE
CERTIFICATION COURSES

Hardware Modeling Using Verilog

51

Synthesis Rules for Combinational Logic

- The output of a combinational logic circuit at time t should depend only upon the inputs applied at time t .
- Rules to be followed:
 - Avoid technology dependent modeling (i.e. implement functionality, not timing).
 - There must not be any feedback in the combinational circuit.
 - For "*if...else*" or "*case*" constructs, the output of the combinational function must be specified for all possible input cases.
 - If the rules are not followed, the circuit may be synthesized as sequential.



IIT KHARAGPUR

NPTEL ONLINE
CERTIFICATION COURSES

Hardware Modeling Using Verilog

52

Styles for Synthesizable Combinational Logic

- The possible styles for modeling combinational logic are as follows:
 - Netlist of Verilog built-in primitives like gate instances (AND, OR, NAND, etc.).
 - Combinational UDP (*not all synthesis tools support this*).
 - Continuous assignments.
 - Functions.
 - Behavioral statements.
 - Tasks without event or delay control.
 - Interconnected modules of one or more of the above.



IIT KHARAGPUR

NPTEL ONLINE
CERTIFICATION COURSES

Hardware Modeling Using Verilog

53

(a) As a Gate Netlist

```
module example (x1, x2, x3, x4, y);
  input x1, x2, x3, x4;
  output y;
  wire w1, w2, w3;
  or (w1, x1, x2);
  or (w2, x3, x4);
  xor (w3, x3, x4);
  nand (y, w1, w2, w3);
endmodule
```

Shall be synthesized in terms of gates from some target technology library.

The gate netlist is often optimized during synthesis.



IIT KHARAGPUR

NPTEL ONLINE
CERTIFICATION COURSES

Hardware Modeling Using Verilog

54

(b) Using Continuous Assignment

```
module carry (cout, a, b, c);
    input a, b, c;
    output cout;

    assign cout = (a & b) | (b & c) | 
                  (c & a);
endmodule
```

Shall be mapped to gates or cells from some target technology library.

The Boolean equations are optimized during synthesis.



IIT KHARAGPUR



NPTEL
ONLINE
CERTIFICATION COURSES

Hardware Modeling Using Verilog

55

(c) Using Procedural Blocking Assignment

```
module mux2to1 (f, in0, in1, sel);
    input in0, in1, sel;
    output reg f;

    always @(in0 or in1 or sel)
        if (sel) f = in1;
        else      f = in0;
endmodule
```

Inputs to the behavior (*here in0, in1, sel*) must be included in the event control expression; otherwise, a latch will be inferred at the output.



IIT KHARAGPUR



NPTEL
ONLINE
CERTIFICATION COURSES

Hardware Modeling Using Verilog

56

(d) Using Functions in Verilog

```
module fulladder (s, cout, a, b, cin);
    input a, b, cin;
    output s, cout;
    assign s = sum(a, b, cin);
    assign cout = carry(a, b, cin);
endmodule
```

```
function sum;
    input x, y, z;
    begin
        sum = x ^ y ^ z;
    end
```

```
function carry;
    input x, y, z;
    begin
        carry = (x&y) | (y&z) | (z|x);
    end
```

A function in Verilog returns a single value.

Can be used to make a code more readable.

Typically used with “assign”.

Input arguments appear in same order.



IIT KHARAGPUR



NPTEL
ONLINE
CERTIFICATION COURSES

Hardware Modeling Using Verilog

57

```
module fulladder (s, cout, a, b, cin);
    input a, b, cin;
    output reg s, cout;

    always @ (a or b or cin)
        FA (s, cout, a, b, cin);

    task FA;
        output sum, carry;
        input A, B, C;
        begin
            #2 sum = A ^ B ^ C;
            carry = (A&B) | (B&C) | (C|A);
        end
    endtask
endmodule
```

(e) Using Tasks

The arguments must be specified in the same order as they appear in the task declaration.

More than one output value can be returned.



IIT KHARAGPUR



NPTEL
ONLINE
CERTIFICATION COURSES

Hardware Modeling Using Verilog

58

Difference between Function and Task

| Function | Task |
|--|--|
| A function can call another function but not another task. | A task can call other tasks and functions. |
| A function executes in 0 simulation time. | A task may execute in nonzero simulation time. |
| A function cannot contain any delay, event, or timing control statement. | A task can contain delay, event, or timing control statements. |
| A function always return a single value. | A task can pass multiple values through " <i>output</i> " and " <i>inout</i> " type arguments. |
| A function must have at least one input argument. | A task can have zero or more arguments of type " <i>input</i> ", " <i>output</i> ", or " <i>inout</i> ". |



IIT KHARAGPUR

NPTEL ONLINE
CERTIFICATION COURSES

Hardware Modeling Using Verilog

59

Constructs to Avoid for Combinational Synthesis

- Edge-dependent event control.
- Combinational feedback loops.
- Procedural or continuous assignment containing event or delay control.
- Procedural loops with timing.
- Data dependent loops.
- Sequential user defined primitives (UDPs).
- Other miscellaneous constructs like "*fork ... join*", "*wait*", "*disable*", etc.



IIT KHARAGPUR

NPTEL ONLINE
CERTIFICATION COURSES

Hardware Modeling Using Verilog

60

Summary: Synthesizable Verilog Constructs

- “*module* ... *endmodule*”
- Instantiation of a synthesizable module
- “*always*” construct
- “*assign*” statements
- Built-in gate primitives
- User defined primitives – combinational only
- “*parameter*” statement
- “*functions*” and “*tasks*”
- “*for*” loop
- Almost all operators
- Blocking and non-blocking assignments
- “*if ... else*”, “*case*”, “*casex*” and “*casez*”
- Bits and part select of vectors



IIT KHARAGPUR

NPTEL ONLINE
CERTIFICATION COURSES

Hardware Modeling Using Verilog

61

Non-Synthesizable Constructs

- “*initial*” construct
- Delays in assignments and test benches.
- “*time*” construct
- “*real*” data type
- The operators “*==*” and “*!=*”
- “*fork ... join*” constructs
- “*force ... release*” constructs
- Variables in loop control



IIT KHARAGPUR

NPTEL ONLINE
CERTIFICATION COURSES

Hardware Modeling Using Verilog

62

END OF LECTURE 28



IIT KHARAGPUR



NPTEL
ONLINE
CERTIFICATION COURSES

Hardware Modeling Using Verilog

63



IIT KHARAGPUR



NPTEL
ONLINE
CERTIFICATION COURSES

Lecture 29: SOME RECOMMENDED PRACTICES

PROF. INDRANIL SENGUPTA
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Naming Conventions

- File Naming
 - A file must contain only one design unit, contained in a single “*module ... endmodule*” construct.
 - All Verilog files must have an extension of **.v**.
- Naming of variables, signals and other objects
 - Names must be composed of alphanumeric characters or underscores.
 - Names must start with a letter, and not a number or underscore.
 - All names must be unique irrespective of case.
 - Parameters and constant names must be given in UPPER CASE (e.g. **PI**, **DELAY**, etc.).
 - Signals and variable names must be in lower case, and must be meaningful names.
 - A constant name must describe the purpose of the constant (e.g. **reg_a_enable**).



IIT KHARAGPUR

NPTEL ONLINE
CERTIFICATION COURSES

Hardware Modeling Using Verilog

65

- Construct names such as modules, functions and tasks must also be meaningful.
- For names composed of several words, underscore must be used (e.g. **load_input**).
- When a signal uses active low polarity, it must use the suffix **_b** (e.g. **clear_b**).
- Signals that are used for clocking that do not have the word **clock** or **clk** already in their names, must use the suffix **_clk** (e.g. **bus_clk**).
- Unrelated signals must not be bundled into buses.



IIT KHARAGPUR

NPTEL ONLINE
CERTIFICATION COURSES

Hardware Modeling Using Verilog

66

Comments

- Comments are required to describe the functionality of a design unit.
 - Each file must contain a file header, that follows some convention.
 - The header must include the name of the file.
 - The header must include the highest level construct contained in the file.
 - Every file header must include the originating section or department, author, and author's email address.
 - Header must include release history whenever such a change is registered.
 - The header must contain a purpose section explaining the functionality of the module.
 - The header must contain information describing the parameters being used in the construct.
 - The number of clock domains and clocking strategy must be documented.
 - Critical timing including external timing relationships must be documented.



IIT KHARAGPUR

NPTEL ONLINE
CERTIFICATION COURSES

Hardware Modeling Using Verilog

67

Coding Style

- Good coding style helps in easy understanding of the code and maintainability.
 - Write code with proper indentation in a tabular format.
 - A constant indentation of *2 to 4 spaces* must be used for code alignment; do not use tab stops (tab stops interpretation varies from system to system).
 - Use spaces and empty lines to increase the readability of code.
 - One line must not contain more than one Verilog statement.
 - Use one line comments using “//”; avoid multi-line comments using “/* ... */”.
 - Keep line length less than 80 characters, so as to avoid line wraps.
 - When declaring ports, declare *one port per line*. Descriptive comment must follow each port listing.



IIT KHARAGPUR

NPTEL ONLINE
CERTIFICATION COURSES

Hardware Modeling Using Verilog

68

Module Partitioning

- Used for reducing complexity, and also to minimize the chances of error.
 - Procedure, tasks and functions must not access or modify signals / variables not passed as parameter to the module.
 - If we use a gated clock, internally generated clock, or use both edges of a clock, the clock generation circuitry must be kept in a separate module at the top level or at the same logical level in the hierarchy as the block to which the clocks apply.
 - Separate clock domains (e.g. *slow clock* and *fast clock*) must be partitioned into separate blocks.
 - The design should be partitioned so as to minimize the number of interface signals.
 - Do not mix structural and behavioral RTL code within a construct.
 - State machines and asynchronous logic must be partitioned in a separate block.



IIT KHARAGPUR

NPTEL ONLINE
CERTIFICATION COURSES

Hardware Modeling Using Verilog

69

General Coding Techniques

- Some of the general guidelines for coding are as follows.
 - The expression in a condition must be a 1-bit value.
Replace “*if (bus) data_avail = 1*” by “*if (bus > 0) data_avail = 1*”.
 - Do not assign signals to “*x*”.
 - Do not infer latches in functions; a function is supposed to synthesize combinational logic.
 - Operand sizes should match.
 - Use parentheses in complex expressions.



IIT KHARAGPUR

NPTEL ONLINE
CERTIFICATION COURSES

Hardware Modeling Using Verilog

70

General Guidelines for Synthesis

- Some guidelines for synthesizable blocks are as follows.
 - All “*always*” blocks inferring combinational logic or a latch must have a sensitivity list containing all input signals.
 - Only one clock must be used in an “*always*” block.
 - “*wait*” and “#*delay*” statements must not be used.
 - Conditional expressions must be specified completely; i.e. value must be assigned to a variable under all conditions.
 - The “*initial*” statement must not be used.
 - Expressions must not be used in port connections.
 - Verilog user defined primitives must not be used.
 - Use non-blocking assignments in edge-sensitive constructs.

In *wait(expression)*, the expression is evaluated. If false, execution is suspended until it becomes true.



IIT KHARAGPUR

NPTEL ONLINE
CERTIFICATION COURSES

Hardware Modeling Using Verilog

71

- The internal “*wire*” declarations must follow the port I/O declarations at the top of the module.
- Use explicit port references in module instantiations.

```
full_adder FA (.sum(S),  
                 .cout(Co),  
                 .in1(A),  
                 .in2(B),  
                 .cin(Ci) );
```

- Use of “*cased*” statement is not allowed, which treats “*x*” and “*z*” states as don’t cares in synthesis.
- Use parameters for state encoding.



IIT KHARAGPUR

NPTEL ONLINE
CERTIFICATION COURSES

Hardware Modeling Using Verilog

72

- Only use ports of type “*input*” or “*output*”; ports of type “*inout*” (bidirectional) should be avoided.
- In procedural blocks, blocking assignments should be used for modeling combinational logic.
- The “*default*” case assignment must be used for all combinational logic case statement descriptions.



IIT KHARAGPUR

NPTEL ONLINE
CERTIFICATION COURSES

Hardware Modeling Using Verilog

73

An Example

```
// Copyright (c) 2017 IIT Kharagpur
// -----
// FILE NAME: counter.v
// TYPE: module
// DEPARTMENT: Computer Sceience and Engineering
// AUTHOR: Prof. Indranil Sengupta
// AUTHOR'S EMAIL: indranil@cse.iitkgp.ernet.in
// -----
// Release history
// VERSION DATE AUTHOR DESCRIPTION
// 1.0 10/08/2016 indranil Initial version
// 2.0 12/07/2017 subir Updated version with clear
// 2.1 16/08/2017 indranil Asynchronous clear
// -----
// KEYWORDS: binary counter, asynchronous clear
// -----
// PURPOSE: 16-bit binary counter
```



IIT KHARAGPUR

NPTEL ONLINE
CERTIFICATION COURSES

Hardware Modeling Using Verilog

74

```
module counter (
    data,
    clear,
    clock
);

output reg [15:0] data; // The 16-bit count value

input clear;           // Asynchronous clear
input clock;          // Counter clock

// 16-bit binary counter with asynchronous clear
always @(posedge clock or negedge clear)
    if (!clear)
        data <= 16'b0000000000000000;
    else
        data <= data + 1;

endmodule
```



IIT KHARAGPUR

NPTEL ONLINE
CERTIFICATION COURSES

Hardware Modeling Using Verilog

75

END OF LECTURE 29



IIT KHARAGPUR

NPTEL ONLINE
CERTIFICATION COURSES

Hardware Modeling Using Verilog

76