VERILOG

Swapneel Chandrakant Mhatre

AGENDA

- Front-End Design Steps of Integrated Circuits (ICs)
- Back-End Design Steps of Integrated Circuits (ICs)
- Hardware Description Languages (HDLs)
- Tools for Simulation and Synthesis
- Modeling in Verilog
- Gate-Level Modeling (Structural Modeling)
- Dataflow Modeling
- Behavioural Modeling
- Stimulus (Testbench)

AGENDA

- Steps to Use ModelSim – Intel FPGA Starter Edition 10.5b
- NOT Gate (Using Gate-Level Modeling)
- NOT Gate (Using Dataflow Modeling)
- NOT Gate (Using Behavioural Modeling)
- Stimulus for NOT Gate
- AND Gate (Using Gate-Level Modeling)
- AND Gate (Using Dataflow Modeling)
- AND Gate (Using Behavioural Modeling)
- Stimulus for AND Gate
- 16-Bit NOT Gate
- Stimulus for 16-Bit NOT Gate
- 16-Bit AND Gate
- Stimulus for 16-Bit AND Gate

AGENDA

- Instruction Cycle, Machine Cycle and T-State
- Fetch
- Decode
- Execute
- Verilog Code Segment

AGENDA

- Multiplication Using Successive Addition
- Multiplication Using Add and Shift Method

FRONT-END DESIGN STEPS OF INTEGRATED CIRCUITS (ICs)

- Modeling
    - Input: Circuit behaviour (Design)
    - Output: Hardware Description Language (HDL) program
- Simulation
    - Input: Hardware Description Language (HDL) program and testbench (stimulus)
    - Output: Simulated output of the circuit (may be represented in the form of text or waveform)
- Synthesis
    - Input: Hardware Description Language (HDL) program
    - Output: Actual hardware diagram [RTL (register transfer level) design, gate level netlist and mapping to the standard cells]

BACK-END DESIGN STEPS OF INTEGRATED CIRCUITS (ICs)

- Layout
- Floor Planning
- Routing

HARDWARE DESCRIPTION LANGUAGES (HDLs)

- VHDL [VHSIC (Very High Speed Integrated Circuit) Hardware Description Language]
- Verilog
- SystemC
- SystemVerilog

TOOLS FOR SIMULATION AND SYNTHESIS

- Tool for Simulation
    - ModelSim
- Tool for Synthesis
    - LeonardoSpectrum
- Tool for all the Design Steps of ICs except Simulation
    - Xilinx ISE (Integrated Software Environment)
- Tool for all the Design Steps of ICs including Simulation
    - Xilinx Vivado Design Suite

MODELING IN VERILOG

- Switch-Level Modeling
- Gate-Level Modeling (Structural Modeling)
- Dataflow Modeling
- Behavioural Modeling

GATE-LEVEL MODELING (STRUCTURAL MODELING)

```
module nand2(c,a,b);


output c;
input a,b;
```

GATE-LEVEL MODELING (STRUCTURAL MODELING)

```
nand(c,a,b);


endmodule
```

DATAFLOW MODELING

```
module nand2(c,a,b);


output c;
input a,b;
```

DATAFLOW MODELING

```
assign c=~(a&b);


endmodule
```

BEHAVIOURAL MODELING

```
module nand2(c,a,b);


output c;
input a,b;


reg c;
```

BEHAVIOURAL MODELING

```
always@(a,b)
begin
   c=~(a&b);
end

endmodule
```

STIMULUS (TESTBENCH)

```
module stimulus;


wire c;
reg a,b;


nand2 nand2_1(c,a,b);
```

STIMULUS (TESTBENCH)

```
initial
begin
        a=1'b0; b=1'b0;
  #10 a=1'b0; b=1'b1;
  #10 a=1'b1; b=1'b0;
  #10 a=1'b1; b=1'b1;
end


endmodule
```

STEPS TO USE MODELSIM – INTEL FPGA STARTER EDITION 10.5b

- File – New – Project
  - Project Name: _____
  - Project Location: /home/student
  - Default library name: work
- In ModelSim – Intel FPGA starter edition 10.5b, project is created with extension .mpf.

STEPS TO USE MODELSIM – INTEL FPGA STARTER EDITION 10.5b

- Add items to the Project
    - Create New File
        - File Name: _____
        - Add file as type: Verilog
    - Add Existing File
    - Close
- Create new file for stimulus.
- File – New – Source – Verilog
- If file type or source is selected as Verilog, file is created with extension .v.

STEPS TO USE MODELSIM – INTEL FPGA STARTER EDITION 10.5b

- Write program and save file.

STEPS TO USE MODELSIM – INTEL FPGA STARTER EDITION 10.5b

- Project – Add to project – Existing file

STEPS TO USE MODELSIM – INTEL FPGA STARTER EDITION 10.5b

- Compile – Compile All

STEPS TO USE MODELSIM – INTEL FPGA STARTER EDITION 10.5b

- Simulate – Start Simulation
- Select library tab.
- Select work – stimulus.
- Design Unit(s): work.stimulus

STEPS TO USE MODELSIM – INTEL FPGA STARTER EDITION 10.5b

- Select sim tab.
- Add – To Wave – All items in design
- Run / Restart, Run (Zoom In / Zoom Out)
- Text output (if any) is displayed in transcript window.

STEPS TO USE MODELSIM – INTEL FPGA STARTER EDITION 10.5b

- Simulate – End Simulation

NOT GATE
(USING GATE-LEVEL MODELING)

```
module not_gate(b,a);


output b;
input a;
```

NOT GATE
(USING GATE-LEVEL MODELING)

```
// not(b,a);
nand(b,a,a);


endmodule
```

NOT GATE
(USING DATAFLOW MODELING)

```
module not_gate(b,a);


output b;
input a;
```

NOT GATE
(USING DATAFLOW MODELING)

```
// assign b=~a;
assign b=~(a&a);

endmodule
```

NOT GATE (USING BEHAVIOURAL MODELING)

```
module not_gate(b,a);


output b;
input a;


reg b;
```

NOT GATE (USING BEHAVIOURAL MODELING)

```
always@(a)
begin
//   b=~a;
   b=~(a&a);
end


endmodule
```

STIMULUS FOR NOT GATE

```
module stimulus;


wire b;
reg a;


not_gate not_gate1(b,a);
```

STIMULUS FOR NOT GATE

```
initial
begin
   $monitor(“a = %b, b = %b,a,b);

         a=1'b0;
   #10 b=1'b1;
end

endmodule
```

AND GATE (USING GATE-LEVEL MODELING)

```
module and_gate(c,a,b);


output c;
input a,b;


wire x;
```

AND GATE
(USING GATE-LEVEL MODELING)

```
// and(c,a,b);
nand(x,a,b);
nand(c,x,x);

endmodule
```

AND GATE (USING DATAFLOW MODELING)

```
module and_gate(c,a,b);


output c;
input a,b;
```

AND GATE (USING DATAFLOW MODELING)

```
// assign c=a&b;
assign c=~((~(a&b))&(~(a&b)));

endmodule
```

AND GATE (USING BEHAVIOURAL MODELING)

```
module and_gate(b,a);


output reg c;
input a,b;
```

AND GATE (USING BEHAVIOURAL MODELING)

```
always@(a,b)
begin
//    c=a&b;
   c=~((~(a&b))&(~(a&b)));
end


endmodule
```

STIMULUS FOR AND GATE

```
module stimulus;


wire c;
reg a,b;


and_gate and_gate1(c,a,b);
```

STIMULUS FOR AND GATE

```
always
begin
   $monitor(“a = %b, b = %b, c = %b”,a,b,c);


   #10 a=1'b0; b=1'b0;
   #10 a=1'b0; b=1'b1;
   #10 a=1'b1; b=1'b0;
   #10 a=1'b1; b=1'b1;
end


endmodule
```

16-BIT NOT GATE

```
module not_gate_16(b,a);


output [15:0] b;
input [15:0] a;
```

16-BIT NOT GATE

```
not_gate not_gate_1 [15:0] (b,a);


endmodule
```

STIMULUS FOR 16-BIT NOT GATE

```
module stimulus;


wire [15:0] b;
reg [15:0] a;

integer i;


not_gate_16 not_gate_16_1(b,a);
```

STIMULUS FOR 16-BIT NOT GATE

```
initial
begin
   i=0;
   while(i<=65535)
   begin
      a=i;
      #10 i=i+1;
   end
end


endmodule
```

16-BIT AND GATE

```
module and_gate_16(c,a,b);


output [15:0] c;
input [15:0] a,b;
```

16-BIT AND GATE

```
and_gate and_gate_1 [15:0] (c,a,b);


endmodule
```

STIMULUS FOR 16-BIT AND GATE

```
module stimulus;


wire [15:0] c;
reg [15:0] a,b;


integer i,j;


and_gate_16 and_gate_16_1(c,a,b);
```

STIMULUS FOR 16-BIT AND GATE

```
always
begin
   for(i=0;i<=65535;i=i+1)
      for(j=0;j<=65535;j=j+1)
      begin
         #10 a=i; b=j;
      end
end


endmodule
```

INSTRUCTION CYCLE, MACHINE CYCLE AND T-STATE

- Instruction cycle consists of machine cycles.
- Machine cycles:
  - Fetch
  - Decode
  - Execute
- Machine cycle consists of T-states (clock cycles).

FETCH

- Output address contained in the program counter (PC).
- Assert memory read.
- Read data (instruction) into instruction register (IR).
- PC = PC + 2 (if length of the instruction is 2 bytes)

DECODE

- Read IR.
- Consider IR to be 16-bit.

  opcode = IR15 – IR12

  R3 = IR11 – IR8

  R1 = IR7 – IR4

  R2 = IR3 – IR0
- If opcode = 0000, operation = 0 (multiplication)

  If opcode = 0001, operation = 1 (division, get quotient)

  If opcode = 0010, operation = 2 (division, get remainder)

EXECUTE

- Read contents of R1 and R2.
- Send contents of R1 and R2 to the arithmetic logic unit (ALU).
- If operation = 0, send multiply signal to ALU.

  If operation = 1, send divide signal to ALU, get quotient.

  If operation = 2, send divide signal to ALU, get remainder.
- Get result from the ALU.
- Store result in R3.

VERILOG CODE SEGMENT

```
module computer;

for (machine_cycle=1; machine_cycle<=3;
machine_cycle=machine_cycle+1)

// FETCH, DECODE, EXECUTE

case (machine_cycle)

   1 : // FETCH

   2 : // DECODE

   3 : // EXECUTE

endcase

endmodule
```

MULTIPLICATION USING SUCCESSIVE ADDITION

```
        mov ax,0000h
        mov bh,00h
        mov bl,[m]      ; MULTIPLICAND
        mov cl,[q]      ; MULTIPLIER
next:   add ax,bx
        dec cl
        jnz next
```

MULTIPLICATION USING ADD AND SHIFT METHOD

```
        mov bx,0        ; PARTIAL PRODUCT
        mov bl,[q]      ; MULTIPLIER
        mov ax,0
        mov ah,[m]      ; MULTIPLICAND
        mov cl,8
next:   test bl,01h
        jz skip
        add bx,ax       ; ADD
skip:   shr bx,1        ; SHIFT
        dec cl
        jnz next
                        ; AT THIS POINT, THE PARTIAL PRODUCT BECOMES THE FINAL PRODUCT.
```