Discussion 5

Wei-Hsiang Ma

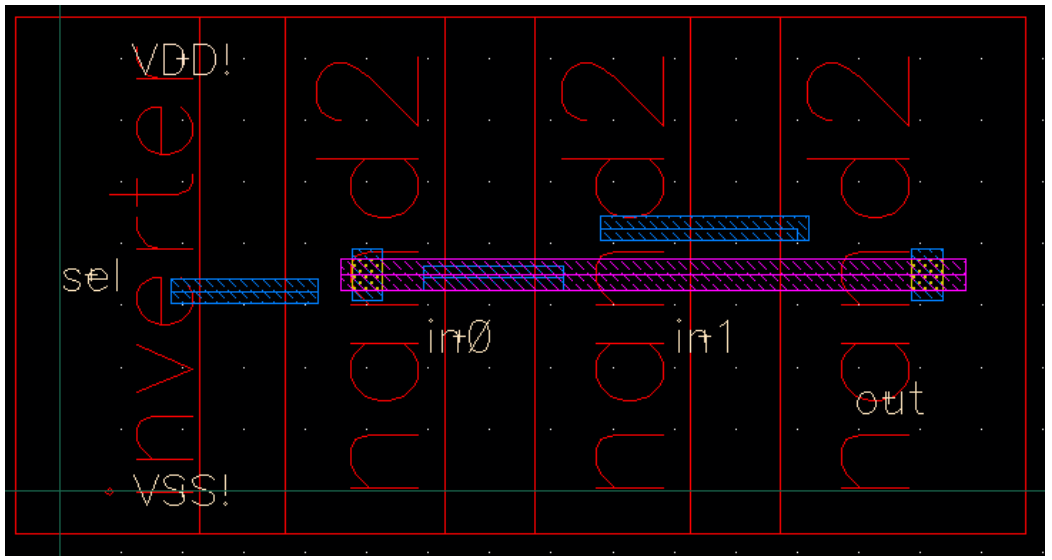
2009/02/13

Today

- GSI's CAD1 / CAD2
- Synthesis Design Flow
- Verilog
- Tutorial 2 demo (page1 – page2)

GSI's CAD1

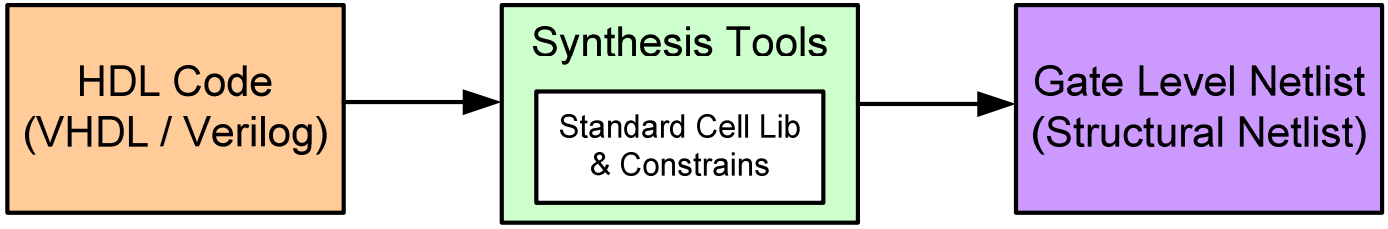
- Tips:
  - Use "**OUTLINE**" layer to define the boundary
  - Make "**GRLOGIC**" fit the cell width
  - Overlap **VDD/VSS** labels
  - Do connections on the higher level
  - Use minimum width wire for connections
  - Area: 6um x 2.8 um



GSI's CAD2

- Tips:
  - Use M1 as much as possible (for cell level)
  - Use higher level metal for global signals (such as Reset)
  - Area: 12um x 3.6um

Synthesis Flow



add2.v

```
module add2 (Y, A, B);
input [1:0] A, B;
output [1:0] Y;

assign Y[1:0] = A[1:0] + B[1:0];

endmodule
```

Constrains:

set input delay
set output delay
set output load
set clock period

Std Cells:

INVX1TR
INVX2TR
INVX3TR
AND2X1TR
AND2X2TR
AND2X3TR
OR2X1TR
OR2X2TR
OR2X3TR
XOR2X1TR
XOR2X2TR
XOR2X3TR
XOR3X1TR
XOR3X2TR
XOR3X3TR

add2.nl.v

```
module add2 (Y, A, B);
input [1:0] A, B;
output [1:0] Y;

XORX2X1TR I1 (.Y(Y[0]), .A(A[0]), .B(B[0]));
AND2X2TR I0 (.Y(C0), .A(A[0]), .B(B[0]));
XOR3X2TR I2 (.Y(Y[1]), .A(A[1]), .B(B[1]), . C(C0));

endmodule
```

- Synthesis tools will choose the size of the std cell according to the design constrains

Verilog (Sequential Logic)

DFF without RESET

```
module DFF (Q, Q_b, D, CK);
input D, CK;
output Q, Q_b;

reg Q, Q_b;

always @(posedge CK)
begin
Q <= D;
Q_b <= ~D;
end

endmodule
```

DFF with Sync RESET

```
module DFF (Q, Q_b, D, CK, R);
Input D, CK,R;
output Q, Q_b;

reg Q, Q_b;

always @(posedge CK)
begin
  If (R == 1'b0)
  begin
  Q <= 1'b0;
  Q_b <= 1'b1;
  end

  else
  begin
  Q <= D;
  Q_b <= ~D;
  end
end

endmodule
```

DFF with Async RESET

```
module DFF (Q, Q_b, D, CK, R);
Input D, CK,R;
output Q, Q_b;

reg Q, Q_b;

always @(posedge CK or negedge R)
begin
  If (R == 1'b0)
  begin
  Q <= 1'b0;
  Q_b <= 1'b1;
  end

  else
  begin
  Q <= D;
  Q_b <= ~D;
  end
end

endmodule
```

- Declare Q, Q_b as register
- Use “<=“ in always block

Verilog (Combinational Logic)

Use Assign	**Use Always block**
module mult (Y, A, B); input [1:0] A, B; output [3:0] Y; **assign Y = A * B;** endmodule	module mult (Y, A, B); input [1:0] A, B; output [3:0] Y; **reg [3:0] Y;** **always@ *** **begin** **Y = A * B;** **end** endmodule

- Tips for using always block
  - Declare Y as register, even though it is not.
  - Use “= “ instead of “ <= “

Verilog (Seq + Com Logic)

```
module mult (Y, A, B, CK);
input [1:0] A, B;
Input CK;
output [3:0] Y;

reg [3:0] Y;

assign temp = A * B;

always @(posedge CK)
begin
Y <= temp;
end

endmodule
```

How to verify your Verilog before synthesizing it ?

- Create “**functional**” view
- Paste your behavioral code into the “**functional**” view
- Create an empty “**schematic**” view
- When doing NCverilog simulation, choose “**functional**” instead of “**schematic**”
- Modify your testbench and simulate it

Demo

- Mult.v (behavioral verilog code)
- Behavioral simulation (NCVerilog)
- Mult.tcl (define general constrains)
- Timing.tcl (define clock constrains)
- Mult.nl.v (structural netlist)
- Mult.dc.rpt (read report)

What you should do after demo

- Go through the synthesis flow by yourself!