module counter (clk, clr, load, in, count);
parameter width=8;
input clk, clr, load;
inout [width-1 : 0] in;
output [width-1 : 0] count;
reg [width-1 : 0] tmp;

always @(posedge clk or negedge clr)
begin
  if (!clr)
    tmp = 0;
  else if (load)
    tmp = in;
  else
    tmp = tmp + 1;
end
assign count = tmp;
endmodule

Figure 11.1: Simple counter behavioral Verilog code
module counter ( clk, clr, load, in, count );
  input [7:0] in;
  output [7:0] count;
  input clk, clr, load;
  wire  N5, N6, N7, N8, N9, N10, N11, N12, N13, N14, N15, N16, N17, N18, N19,
        N20, n6, n7, n8, n9, n10, n11, n12, n13, n14;

  MUX2X2 U3 ( .A(N11), .B(in[7]), .S(load), .Y(N19) );
  MUX2X2 U4 ( .A(N20), .B(in[6]), .S(load), .Y(N18) );
  MUX2X2 U5 ( .A(N10), .B(in[5]), .S(load), .Y(N17) );
  MUX2X2 U6 ( .A(N9), .B(in[4]), .S(load), .Y(N16) );
  MUX2X2 U7 ( .A(N8), .B(in[3]), .S(load), .Y(N15) );
  MUX2X2 U8 ( .A(N7), .B(in[2]), .S(load), .Y(N14) );
  MUX2X2 U9 ( .A(N6), .B(in[1]), .S(load), .Y(N13) );
  MUX2X2 U10 ( .A(N5), .B(in[0]), .S(load), .Y(N12) );
  DCBX1 tmp_reg_0_ ( .D(N12), .CLK(clk), .CLR(clr), .Q(count[0]), .QB(N5) );
  DCBX1 tmp_reg_1_ ( .D(N13), .CLK(clk), .CLR(clr), .Q(count[1]), .QB(n6) );
  DCBX1 tmp_reg_2_ ( .D(N14), .CLK(clk), .CLR(clr), .Q(count[2]) );
  DCBX1 tmp_reg_3_ ( .D(N15), .CLK(clk), .CLR(clr), .Q(count[3]), .QB(n8) );
  DCBX1 tmp_reg_4_ ( .D(N16), .CLK(clk), .CLR(clr), .Q(count[4]) );
  DCBX1 tmp_reg_5_ ( .D(N17), .CLK(clk), .CLR(clr), .Q(count[5]), .QB(n11) );
  DCBX1 tmp_reg_6_ ( .D(N18), .CLK(clk), .CLR(clr), .Q(count[6]) );
  DCBX1 tmp_reg_7_ ( .D(N19), .CLK(clk), .CLR(clr), .Q(count[7]) );
  AOI2XX1 U11 ( .A(count[0]), .B(count[1]), .C(n6), .D(N5), .Y(n6) );
  NOR2XX1 U12 ( .A(N5), .B(n6), .Y(n7) );
  XOR2XX1 U13 ( .A(count[2]), .B(n7), .Y(n7) );
  NAND2XX1 U14 ( .A(count[2]), .B(n7), .Y(n9) );
  MUX2XX1 U15 ( .A(count[3]), .B(n8), .S(n9), .Y(N8) );
  NOR2XX1 U16 ( .A(n9), .B(n8), .Y(n10) );
  XOR2XX1 U17 ( .A(count[4]), .B(n10), .Y(N9) );
  NAND2XX1 U18 ( .A(count[4]), .B(n10), .Y(n12) );
  MUX2XX1 U19 ( .A(count[5]), .B(n11), .S(n12), .Y(N10) );
  NOR2XX1 U20 ( .A(n12), .B(n11), .Y(n13) );
  XOR2XX1 U21 ( .A(n13), .B(count[6]), .Y(N20) );
  NAND2XX1 U22 ( .A(count[6]), .B(n13), .Y(n14) );
  XNOR2XX1 U23 ( .A(count[7]), .B(n14), .Y(N11) );
endmodule

Figure 11.2: Simple synthesized counter Verilog code using the example.lib cell library
Figure 11.3: Timing information (.sdc file) for the counter example
Figure 11.4: Main SOC Encounter GUI
**Figure 11.5:** Design Import dialog box, Basic tab
Figure 11.6: Design Import IPO/CTS tab
Figure 11.7: Design Import Power tab
Figure 11.8: The Specify Floorplan dialog box, Basic tab
Figure 11.9: The Specify Floorplan dialog box, Advanced tab
Figure 11.10: Main design window after floorplanning
Figure 11.11: Dialog box for adding power and ground rings around your cell
Figure 11.12: Dialog box for planning power stripes
Figure 11.13: Advanced Tab of Dialog box for planning power stripes
**Figure 11.14**: Floorplan after power rings and stripes have been generated and connected to the cell rows
Figure 11.15: Placement dialog box
Figure 11.16: View after placement of the cells
Figure 11.17: Dialog box for timing optimization
### optDesign Final Summary

<table>
<thead>
<tr>
<th>Setup mode</th>
<th>all</th>
<th>reg2reg</th>
<th>in2reg</th>
<th>reg2out</th>
<th>in2out</th>
<th>clkgate</th>
</tr>
</thead>
<tbody>
<tr>
<td>WNS (ns):</td>
<td>-0.329</td>
<td>-0.329</td>
<td>0.981</td>
<td>0.681</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>TNS (ns):</td>
<td>-1.138</td>
<td>-1.138</td>
<td>0.000</td>
<td>0.000</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>Violating Paths:</td>
<td>5</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>All Paths:</td>
<td>24</td>
<td>8</td>
<td>16</td>
<td>8</td>
<td>N/A</td>
<td>N/A</td>
</tr>
</tbody>
</table>

**Figure 11.18:** Initial pre-CTS timing results

---

**Figure 11.19:** Generating a clock tree specification
Figure 11.20: Synthesize Clock Tree dialog box
Figure 11.21: Dialog box to display the clock tree
Figure 11.22: Design display showing the clock tree
**optDesign Final Summary**

<table>
<thead>
<tr>
<th>Setup mode</th>
<th>all</th>
<th>reg2reg</th>
<th>in2reg</th>
<th>reg2out</th>
<th>in2out</th>
<th>clkgate</th>
</tr>
</thead>
<tbody>
<tr>
<td>WNS (ns):</td>
<td>-0.293</td>
<td>-0.293</td>
<td>1.489</td>
<td>0.117</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>TNS (ns):</td>
<td>-1.115</td>
<td>-1.115</td>
<td>0.000</td>
<td>0.000</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>Violating Paths:</td>
<td>5</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>All Paths:</td>
<td>24</td>
<td>8</td>
<td>16</td>
<td>8</td>
<td>N/A</td>
<td>N/A</td>
</tr>
</tbody>
</table>

**Figure 11.23:** Timing results after the second (Post-CTS) optimization

**Figure 11.24:** NanoRoute dialog box
Figure 11.25: The counter circuit after using NanoRoute to do final routing
Figure 11.26: An error in pin placement after final routing
**Figure 11.27**: Design display after fixing the pin routing error

**Figure 11.28**: Final post-route timing optimization results
Figure 11.29: Filler cell dialog box
Figure 11.30: Final cell layout after filler cells are added
Figure 11.31: A zoomed view of a final routed cell
Figure 11.32: Dialog box for verifying connectivity
Figure 11.33: Dialog box for checking geometry
Figure 11.34: Dialog box for exporting DEF
Figure 11.35: Dialog box for importing DEF files to icfb
Figure 11.36: Search dialog box for finding all the cell abstracts
Figure 11.37: Search dialog box for replacing abstracts with layouts
Figure 11.38: Dialog box for importing structural Verilog into a new schematic view
Figure 11.39: Schematic after importing the structural counter from SOC Encounter into icfb
Figure 11.40: Symbol that is created for the counter
Figure 11.41: Configuration file for reading in the counter example
# Pin assignments for counter example
Pin: clk E
Pin: clr E
Pin: load E
Pin: in[1] S
Pin: in[0] S
Pin: count[1] N
Pin: count[0] N

**Figure 11.42:** Example counter.io file that places I/O on specific sides of the macro
```tcl
# SOC Encounter Top-level Command script
# (Erik Brunvand, 2008)
# set the BASENAME for the config files. This will also
# be used for the .lib, .lef, .v, and .spef files
set BASENAME "counter"

# set the name of the filler cells
set fillerCells [list FILL FILL2 FILL4 FILL8]

set usepct 0.70 ;# percent utilization in placing cells
set rowgap 18 ;# gap between pairs of std cell rows
set aspect 0.50 ;# aspect ratio of overall cell (1.0 is square)
# less than 1.0 is landscape, greater than 1.0 is portrait

# You may not have to change things below this line - but check!
set clockBufName inv ;# Footprint of inverter in .lib file

# Note that all these numbers should be divisible by 3 so
# that they fit on the lambda grid
set pwidth 9.9 ;# power rail width
set pspace 1.8 ;# power rail space
set swidth 4.8 ;# power stripe width
set sspace 99 ;# power stripe spacing
set soffset 120 ;# power stripe offset to first stripe
set coregap 30.0 ;# gap between the core and the power rails

# Import design and floorplan from a config file
# If the config file is not named $BASENAME.conf, edit this line.
loadConfig $BASENAME.conf 0
commitConfig

# source the other files that operate on the circuit
source fplan.tcl ;# create the floorplan (might be done by hand...)
source pplan.tcl ;# create the power rings and stripes
source place.tcl ;# Place the cells and optimize (pre-CTS)
source cts.tcl ;# Create the clock tree, and optimize (post-CTS)
source route.tcl ;# Route the design using nanoRoute
source verify.tcl ;# Verify the design and produce output files
exit
```

**Figure 11.43:** A top-level script to execute the place and route process in *SOC Encounter*