CPU Project

VHDL Class
May 31, 2000
Trudy Lary
Project Goals:

1. Write VHDL code for a simple 16-bit CPU

2. Write testbenches for the major CPU blocks

3. Simulate testbenches using ModelTech’s Modelsim

4. Synthesize the VHDL into gates using Synopsys’ Design Compiler Technology Toshiba’s 0.25u Standard cell

5. Insert internal MUX scan and achieve 96% coverage using Mentor Graphics’ DFT Advisor (scan insertion) and Fastscan (ATPG)

6. Insert Memory BIST using Logic Vision

7. Insert Jtag (IEEE 1149.1) using Synopsys’ Testgen
Design Flow

Select Technology and Array

Select package

Calculate power pins

Generate package

Write VHDL code

Compile and simulate VHDL Code

Write Testbench and simulate

Synthesize VHDL code into gates

Simulate gate-level Design (WLM)

Insert internal scan

Insert Memory BIST

Insert Jtag

Repeat this section for all blocks in the design

Floorplan

Place & Route

Clock Tree Synthesis (to reduce clk skew)

Generate new SDF from actual wire lengths

Simulate with SDF from layout

Resynthesize if needed using SDF in Design Compiler

Generate Test Vectors for tester:
Functional test vectors
Internal scan test vectors
Jtag test vectors
MBIST test vectors

Send design to foundry
## TC240C 0.25u ARRAY SELECTION - TLM

<table>
<thead>
<tr>
<th>Part No</th>
<th>TNo</th>
<th>ChipSize</th>
<th>Usable Grid</th>
<th>Usable Gates</th>
<th>MaxPads</th>
<th>IOSlots</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>X</td>
<td>Y</td>
<td>80um</td>
<td>125um</td>
<td>50um</td>
</tr>
<tr>
<td>TC240C24</td>
<td>T8G00-TLM</td>
<td>4,600</td>
<td>4,600</td>
<td>908,000</td>
<td>263,000</td>
<td>164</td>
</tr>
<tr>
<td>TC240C26</td>
<td>T8G01-TLM</td>
<td>5,430</td>
<td>5,430</td>
<td>1,465,000</td>
<td>425,000</td>
<td>204</td>
</tr>
<tr>
<td>TC240C28</td>
<td>T8G02-TLM</td>
<td>6,260</td>
<td>6,260</td>
<td>1,973,000</td>
<td>572,000</td>
<td>244</td>
</tr>
<tr>
<td>TC240C29</td>
<td>T8G03-TLM</td>
<td>6,670</td>
<td>6,670</td>
<td>2,352,000</td>
<td>682,000</td>
<td>264</td>
</tr>
<tr>
<td>TC240C2A</td>
<td>T8G04-TLM</td>
<td>7,090</td>
<td>7,090</td>
<td>2,667,000</td>
<td>773,000</td>
<td>284</td>
</tr>
<tr>
<td>TC240C2B</td>
<td>T8G05-TLM</td>
<td>7,500</td>
<td>7,500</td>
<td>3,087,000</td>
<td>895,000</td>
<td>304</td>
</tr>
<tr>
<td>TC240C2C</td>
<td>T8G06-TLM</td>
<td>7,920</td>
<td>7,920</td>
<td>3,383,000</td>
<td>981,000</td>
<td>324</td>
</tr>
<tr>
<td>TC240C2D</td>
<td>T8G07-TLM</td>
<td>8,330</td>
<td>8,330</td>
<td>3,839,000</td>
<td>1,113,000</td>
<td>344</td>
</tr>
<tr>
<td>TC240C2E</td>
<td>T8G08-TLM</td>
<td>8,740</td>
<td>8,740</td>
<td>4,340,000</td>
<td>1,259,000</td>
<td>364</td>
</tr>
<tr>
<td>TC240C2F</td>
<td>T8G09-TLM</td>
<td>9,160</td>
<td>9,160</td>
<td>4,860,000</td>
<td>1,409,000</td>
<td>384</td>
</tr>
<tr>
<td>TC240C30</td>
<td>T8G10-TLM</td>
<td>9,570</td>
<td>9,570</td>
<td>5,403,000</td>
<td>1,567,000</td>
<td>404</td>
</tr>
<tr>
<td>TC240C31</td>
<td>T8G11-TLM</td>
<td>9,990</td>
<td>9,990</td>
<td>5,995,000</td>
<td>1,739,000</td>
<td>424</td>
</tr>
<tr>
<td>TC240C32</td>
<td>T8G12-TLM</td>
<td>10,400</td>
<td>10,400</td>
<td>6,603,000</td>
<td>1,915,000</td>
<td>444</td>
</tr>
<tr>
<td>TC240C34</td>
<td>T8G13-TLM</td>
<td>11,230</td>
<td>11,230</td>
<td>7,916,000</td>
<td>2,296,000</td>
<td>484</td>
</tr>
<tr>
<td>TC240C36</td>
<td>T8G14-TLM</td>
<td>12,060</td>
<td>12,060</td>
<td>8,888,000</td>
<td>2,578,000</td>
<td>524</td>
</tr>
<tr>
<td>TC240C38</td>
<td>T8G15-TLM</td>
<td>12,880</td>
<td>12,880</td>
<td>10,371,000</td>
<td>3,008,000</td>
<td>564</td>
</tr>
<tr>
<td>TC240C3A</td>
<td>T8G16-TLM</td>
<td>13,710</td>
<td>13,710</td>
<td>11,941,000</td>
<td>3,463,000</td>
<td>604</td>
</tr>
<tr>
<td>TC240C3C</td>
<td>T8G17-TLM</td>
<td>14,540</td>
<td>14,540</td>
<td>13,641,000</td>
<td>3,956,000</td>
<td>644</td>
</tr>
<tr>
<td>TC240C3D</td>
<td>T8G18-TLM</td>
<td>17,440</td>
<td>17,440</td>
<td>19,428,000</td>
<td>5,634,000</td>
<td>784</td>
</tr>
<tr>
<td>TC240C3E</td>
<td>T8G19-TLM</td>
<td>21,830</td>
<td>21,830</td>
<td>31,739,000</td>
<td>9,204,000</td>
<td>996</td>
</tr>
</tbody>
</table>

### Notes:

- **Usable Grid** = \((\text{CoreSize X} / \text{width of grid}) \times (\text{CoreSize Y} / \text{height of grid}) \times \text{Utility}\)
- **Usable Gates** = **Usable Grid** \times (\text{Area factor} / \text{Size of ND2})
- (Size of ND2 = 4 grid, Area factor = EA chip size / CBIC chip size = 1 / 0.86 = 1.16)
- "Area factor" depend on the design. "1.16" is the "Area factor" in condition of 100K Gate[Random 60%, F/F 40%] and using average cell size ratio of all cells)
### TC240C 0.25u ARRAY SELECTION - QLM

**QLM (Quadruple-Layer Metal)**

<table>
<thead>
<tr>
<th>MASTER</th>
<th>ChipSize</th>
<th>Usable</th>
<th>Usable</th>
<th>MaxPads</th>
<th>IOSlots</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>TNumber</td>
<td>X</td>
<td>Y</td>
<td>Grid</td>
<td>Gates</td>
</tr>
<tr>
<td>WB</td>
<td>WB</td>
<td>TAB</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TC240C44</td>
<td>T8G00-QLM</td>
<td>4,600</td>
<td>4,600</td>
<td>994,000</td>
<td>288,000</td>
</tr>
<tr>
<td>TC240C46</td>
<td>T8G01-QLM</td>
<td>5,430</td>
<td>5,430</td>
<td>1,606,000</td>
<td>466,000</td>
</tr>
<tr>
<td>TC240C48</td>
<td>T8G02-QLM</td>
<td>6,260</td>
<td>6,260</td>
<td>2,182,000</td>
<td>633,000</td>
</tr>
<tr>
<td>TC240C49</td>
<td>T8G03-QLM</td>
<td>6,670</td>
<td>6,670</td>
<td>2,601,000</td>
<td>754,000</td>
</tr>
<tr>
<td>TC240C4A</td>
<td>T8G04-QLM</td>
<td>7,090</td>
<td>7,090</td>
<td>2,917,000</td>
<td>846,000</td>
</tr>
<tr>
<td>TC240C4B</td>
<td>T8G05-QLM</td>
<td>7,500</td>
<td>7,500</td>
<td>3,377,000</td>
<td>979,000</td>
</tr>
<tr>
<td>TC240C4C</td>
<td>T8G06-QLM</td>
<td>7,920</td>
<td>7,920</td>
<td>3,716,000</td>
<td>1,078,000</td>
</tr>
<tr>
<td>TC240C4D</td>
<td>T8G07-QLM</td>
<td>8,330</td>
<td>8,330</td>
<td>4,217,000</td>
<td>1,223,000</td>
</tr>
<tr>
<td>TC240C4E</td>
<td>T8G08-QLM</td>
<td>8,740</td>
<td>8,740</td>
<td>4,767,000</td>
<td>1,383,000</td>
</tr>
<tr>
<td>TC240C4F</td>
<td>T8G09-QLM</td>
<td>9,160</td>
<td>9,160</td>
<td>5,338,000</td>
<td>1,548,000</td>
</tr>
<tr>
<td>TC240C50</td>
<td>T8G10-QLM</td>
<td>9,570</td>
<td>9,570</td>
<td>5,935,000</td>
<td>1,721,000</td>
</tr>
<tr>
<td>TC240C51</td>
<td>T8G11-QLM</td>
<td>9,990</td>
<td>9,990</td>
<td>6,585,000</td>
<td>1,910,000</td>
</tr>
<tr>
<td>TC240C52</td>
<td>T8G12-QLM</td>
<td>10,400</td>
<td>10,400</td>
<td>7,253,000</td>
<td>2,103,000</td>
</tr>
<tr>
<td>TC240C54</td>
<td>T8G13-QLM</td>
<td>11,230</td>
<td>11,230</td>
<td>8,695,000</td>
<td>2,522,000</td>
</tr>
<tr>
<td>TC240C56</td>
<td>T8G14-QLM</td>
<td>12,060</td>
<td>12,060</td>
<td>9,808,000</td>
<td>2,844,000</td>
</tr>
<tr>
<td>TC240C58</td>
<td>T8G15-QLM</td>
<td>12,880</td>
<td>12,880</td>
<td>11,444,000</td>
<td>3,319,000</td>
</tr>
<tr>
<td>TC240C5A</td>
<td>T8G16-QLM</td>
<td>13,710</td>
<td>13,710</td>
<td>13,177,000</td>
<td>3,821,000</td>
</tr>
<tr>
<td>TC240C5C</td>
<td>T8G17-QLM</td>
<td>14,540</td>
<td>14,540</td>
<td>15,052,000</td>
<td>4,365,000</td>
</tr>
<tr>
<td>TC240C5D</td>
<td>T8G18-QLM</td>
<td>17,440</td>
<td>17,440</td>
<td>21,548,000</td>
<td>6,249,000</td>
</tr>
<tr>
<td>TC240C5E</td>
<td>T8G19-QLM</td>
<td>21,830</td>
<td>21,830</td>
<td>35,202,000</td>
<td>10,209,000</td>
</tr>
</tbody>
</table>

**Notes:**

Usable Grid = (CoreSize X / width of grid) * (CoreSize Y / height of grid)* Utility
(Utilities of CBIC are the same values as EA)

Usable Gates = Usable Grid * (Area factor / Size of ND2)
(Size of ND2 = 4 grid, Area factor = EA chip size / CBIC chip size = 1 / 0.86 = 1.16)
("Area factor" depend on the design. "1.16" is the "Area factor" in condition of 100K Gate[Random 60%,F/F 40%] and using average cell size ratio of all cells)
How to calculate the number of power and ground pins needed:

\[ VSS = \text{MAX} \ (N1,N2) = \text{Number of Ground pins for Output Buffers} \]

\[ N1 = (\text{Sum}\_\text{drive}\_\text{coeff}\_\text{SSO}) \times KL \times C / 6 \]

\[ N2 = (\text{Sum}\_\text{fcv}\_3.3V\_\text{output} + \text{Sum}\_\text{IOL}\_3.3V\_\text{output}) / (T \times I(VSS)) \]

\[ VDD(2.5V) = \text{MAX}(N1, N2) = \text{Power for 2.5V Output Buffers} \]

\[ N1 = (\text{Sum}\_\text{drive}\_\text{coeff}\_\text{SSO}) \times KL \times C / 8 \]

\[ N2 = (\text{Sum}\_\text{fcv}\_2.5V\_\text{output}) / (T \times I(VDD)) \]

\[ VSS2 = \text{MAX}(N1, N2) = \text{Ground for Input Buffers and Internal Array} \]

\[ N1 = (\text{Sum}\_\text{Coeff}\_\text{SSI}) \times KL \times C / 43 \]

\[ N2 = (\text{Sum}\_\text{fcv}\_\text{Input}\_\text{Buffers} + \text{Sum}\_\text{pass}\_\text{current}\_3.3V\_\text{TTL}\_\text{inputs} + \text{Sum}\_\text{fcv}\_\text{internal}\_\text{cells}) / (T \times I(VSS2)) \]

\[ VDD2(2.5V) = \text{MAX}(N1,N2) \]

\[ N1 = (\text{Sum}\_\text{drive}\_\text{coeff}\_\text{SSI}\_2.5v) \times KL \times C / 43 \]

\[ N2 = (\text{Sum}\_\text{fcv}\_\text{SSI}\_2.5V + \text{Sum}\_\text{fcv}\_\text{internal}\_\text{cells}) / (T \times I(VDD2)) \]

\[ VSS3, VDD3 \text{ same as VSS2, VDD2 except 3.3V} \]

\[ KL = \text{Package Coefficient} \]

\[ T = \text{Temperature} \]

Simultaneous switching =&gt; Switching within 3ns
Power and Ground Pin Calculations:
VSS1 = Ground for Output Buffers
VDD = Power for Output Buffers (2.5V)

**TC240C**

<table>
<thead>
<tr>
<th>VSS1</th>
<th>(OUTPUTS ONLY)</th>
<th>Qty</th>
<th>Freq.</th>
<th>Load</th>
<th>V</th>
<th>Sw. Prob</th>
<th>I (A)</th>
<th>I (VSS)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>N1</strong></td>
<td>4mA</td>
<td>51</td>
<td>1.00</td>
<td>1.3</td>
<td>8.2875</td>
<td>8</td>
<td>BT4</td>
<td></td>
</tr>
<tr>
<td>TOTAL</td>
<td>51</td>
<td></td>
<td></td>
<td></td>
<td>8.2875</td>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>N2</strong></td>
<td>22</td>
<td>35.33</td>
<td>100</td>
<td>3.3</td>
<td>0.50</td>
<td>0.1282</td>
<td>0.09</td>
<td>1</td>
</tr>
<tr>
<td>TOTAL</td>
<td>22</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>8</td>
</tr>
</tbody>
</table>

**VDD**

<table>
<thead>
<tr>
<th>Qty</th>
<th>Freq.</th>
<th>Load</th>
<th>V</th>
<th>Sw. Prob</th>
<th>I (A)</th>
<th>I (VDD)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(OUTPUTS) 4mA</td>
<td>51</td>
<td>1.00</td>
<td>1.3</td>
<td>5.525</td>
<td>6</td>
<td>BT4</td>
</tr>
<tr>
<td>TOTAL</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(OUTPUTS) 4mA</td>
<td>22</td>
<td>35.33</td>
<td>100</td>
<td>3.3</td>
<td>0.50</td>
<td>0.1282</td>
</tr>
<tr>
<td>TOTAL</td>
<td>22</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Power and Ground Pin Calculations:
VSS2 = Ground for Input Buffers and Internal Arrays (2.5V)

<table>
<thead>
<tr>
<th>VSS2</th>
<th>Total</th>
<th>FF</th>
<th>Random</th>
<th>RL</th>
</tr>
</thead>
<tbody>
<tr>
<td>N2</td>
<td>Freq.</td>
<td>Gates</td>
<td>%flops</td>
<td># flops</td>
</tr>
<tr>
<td>Core Gates</td>
<td>35.33</td>
<td>2500</td>
<td>50%</td>
<td>125</td>
</tr>
<tr>
<td>SUBTOTAL</td>
<td>2500</td>
<td>125</td>
<td>1250</td>
<td></td>
</tr>
</tbody>
</table>

Core SRAMs

<table>
<thead>
<tr>
<th>RAMQF</th>
<th>Qty.</th>
<th>Freq.</th>
<th>Sw. Prob.</th>
</tr>
</thead>
<tbody>
<tr>
<td>4x16</td>
<td>1</td>
<td>35.33</td>
<td>0.7</td>
</tr>
</tbody>
</table>

-------- Power values are mW --------

<table>
<thead>
<tr>
<th>FF</th>
<th>Clock</th>
<th>RL</th>
<th>Total gate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Power</td>
<td>Power</td>
<td>Power</td>
<td>Power</td>
</tr>
<tr>
<td>13.45852</td>
<td>10.98763</td>
<td>18.54825</td>
<td>42.9944</td>
</tr>
<tr>
<td>13.45852</td>
<td>10.98763</td>
<td>18.54825</td>
<td>42.9944</td>
</tr>
<tr>
<td>(w/o future growth)</td>
<td>42.9944</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

RAM Power

| 18.66612 | 18.66612 |

CORE TOTAL

| 61.6605 |
| (w/o future growth) | 61.66052 |

TOTAL 0
Power and Ground Pin Calculations:
VSS3 = Ground for Input and Output Buffers
VDD3 = Power for Input and Output Buffers (3.3V)

### VSS3

<table>
<thead>
<tr>
<th>(INPUTS ONLY)</th>
<th>SSI</th>
<th>Drive Coeff.</th>
<th>Margin</th>
<th>N1</th>
<th>Cell Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>N1</td>
<td>50</td>
<td>0.50</td>
<td>1.3</td>
<td>4</td>
<td>IBUF</td>
</tr>
<tr>
<td>TOTAL</td>
<td>50</td>
<td></td>
<td>4.0625</td>
<td>4</td>
<td>4</td>
</tr>
</tbody>
</table>

### VDD3

<table>
<thead>
<tr>
<th>(INPUTS ONLY)</th>
<th>SSI</th>
<th>Drive Coeff.</th>
<th>Margin</th>
<th>N1</th>
<th>Cell Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>N1</td>
<td>50</td>
<td>0.50</td>
<td>1.3</td>
<td>4</td>
<td>IBUF</td>
</tr>
<tr>
<td>TOTAL</td>
<td>50</td>
<td></td>
<td>4.0625</td>
<td>4</td>
<td>4</td>
</tr>
</tbody>
</table>

(Output) (Core) (Input) (Output/Core) (Input)

<table>
<thead>
<tr>
<th>VSS1</th>
<th>VSS2</th>
<th>VSS3</th>
<th>VDD</th>
<th>VDD3</th>
<th>TOTAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>0</td>
<td>4</td>
<td>6</td>
<td>4</td>
<td>22</td>
</tr>
</tbody>
</table>

Total VSS-type 12
Total VDD-type 10 22
PACKAGE: QFP120-P-2828-0.80A
ARRAY: T8G00TW8
BONDING TYPE: STANDARD
CHIP_SIZE: 4.600mmX4.600mm
PRODUCT_NAME: CPU_CORE
Synthesis using Design Compiler

1. Generate Synopsys library and add any special cells (PLL, GTL buffers, etc…)

2. Load your design

3. Set a wire load model and technology

4. Constraint your design, over constraint your design to include boundary and internal scan.

5. Synthesis you design to gates

6. Optimize your design for speed or area
INSTRUCTION LATCH SYNTHESIS TRANSCRIPT

Initializing...
dc_shell> include scripts/instr.do
Error: Include file 'scripts/instr.do' could not be opened. (UI-20)
0
dc_shell> ls scripts
.
   alu.do          dont_use          tap_control.do
..                  command.log    instrLatch.do
0
dc_shell> read -f vhdl vhdl/instr_latch.vhd
Current design is now
'/home/beaver_c/trc/projects/training/vhdl/syn/vhdl/instr_latch.db:instr_latch'
"instr_latch")
set_wire_load_model -name raw_tlm_75
create_clock clk -period 10
set_input_delay 1 -clock clk all_inputs() - clk
set_output_delay 1 -clock clk all_outputs()
include scripts/dont_use
set_dont_use (tc240c.db_WCCOM25:tc240c/CFD*EX*)
set_dont_use (tc240c.db_WCCOM25:tc240c/CFD*EAX*)
compile
Information: Evaluating DesignWare library utilization: (UISN-27)

============================================================================
<table>
<thead>
<tr>
<th>DesignWare Library</th>
<th>Available</th>
</tr>
</thead>
<tbody>
<tr>
<td>DesignWare-Basic</td>
<td>*</td>
</tr>
<tr>
<td>DesignWare-Foundation</td>
<td></td>
</tr>
</tbody>
</table>
============================================================================

Warning: Setting attribute 'fix_multiple_port_nets' on design 'instr_latch'. (UIO-59)
INSTRUCTION LATCH SYNTHESIS TRANSCRIPT CONTINUED

Loading target library 'tc240c'
Loading design 'instr_latch'

Beginning Resource Allocation (constraint driven)
---------------------------------------------
Allocating blocks in 'instr_latch'
Allocating blocks in 'instr_latch'

Beginning Pass 1 Mapping
------------------------
Structuring 'instr_latch'
Mapping 'instr_latch'

Beginning Mapping Optimizations (Medium effort)
-----------------------------------------------

Beginning Delay Optimization Phase
----------------------------------

ELAPSED WORST NEG TOTAL NEG DESIGN
TIME AREA SLACK SLACK RULE COST
--------- --------- --------- --------- ----------
Beginning Area-Recovery Phase (cleanup)
--------------------------------------

ELAPSED WORST NEG TOTAL NEG DESIGN
TIME AREA SLACK SLACK RULE COST
--------- --------- --------- --------- ----------
Optimization Complete
----------------------------------------
Transferring design 'instr_latch' to database
'instr_latch.db'
Current design is 'instr_latch'.
1
report_timing
Information: Updating design information... (UID-85)

******************************************************************************
Report : timing
- path full
- delay max
- max_paths 1
Design : instr_latch
Version: 1999.10-4
Date : Wed May 31 11:26:12 2000
******************************************************************************

Operating Conditions: WCCOM25 Library: tc240c
Wire Load Model Mode: top
Startpoint: instr(0) (input port clocked by clk)
Endpoint: currentInstr1_reg(0) (rising edge-triggered flip-flop clocked by clk)
Path Group: clk
Path Type: max

Des/Clust/Port Wire Load Model Library
----------------------------------------
instr_latch raw_tlm_75 tc240c

Point Incr Path
--------------------
clk clk (rise edge) 0.00 0.00
clock network delay (ideal) 0.00 0.00
input external delay 1.00 1.00 r
instr(0) (in) 0.00 1.00 r
U55/Z (CIvx2) 0.04 1.04 f
U64/Z (CMX12X1) 0.17 1.22 r
current_instr1_reg(0)/D (CFD1XL) 0.00 1.22 r
data arrival time 1.22
INSTRUCTION LATCH SYNTHESIS TRANSCRIPT CONTINUED

clock clk (rise edge) 10.00 10.00
clock network delay (ideal) 0.00 10.00
current_instr_reg(0)/CP (CFD1XL) 0.00 10.00 r
library setup time -0.49 9.51
data required time 9.51

-----------------------------------------
data required time 9.51
data arrival time -1.22

-----------------------------------------

slack (MET) 8.29

write -f vhdl -o mapped/inst_latch.vhd
report_area

****************************************
Report : area
Design : instr_latch
Version: 1999.10-4
Date : Wed May 31 11:26:12 2000
****************************************

Library(s) Used:

tc240c (File:
/home/espresso_d/R1.8A.01/lib/synopsys/
lib/tc240c/tc240c.db_WCCOM25)

Number of ports: 98
Number of nets: 258
Number of cells: 160
Number of references: 3

Combinational area: 160.000000
Noncombinational area: 256.000000

Total cell area: 416.000000
quit
Thank you...
What is internal Scan used for

Use to increase test coverage of chip by making as many nodes as possible:

Controllable

Observable

Then you can generate a pattern to detect a fault and propagate the fault to an output

Example:

Forcing a ‘1’ at either A or B will test for a stuck-at-zero fault on Z

\[\begin{array}{c}
\text{A} \\
\text{OR gate} \\
\text{B} \\
\text{Z} \\
\end{array}\]

Stuck-at-zero fault
How to Make nodes controllable and observable using scan/test insertion

Make all sequential elements controllable and observable by replacing DFF with ScanFF.

Insert test logic to control internal resets, clocks and ram write enables not controllable externally.

Example:

Before Scan  After Scan

Before test logic insertion uncontrollable reset

After test logic insertion, reset is controllable
Tools in the industry for internal scan insertion

Mentor Graphics’
  DFT Advisor - Scan and test insertion
  Fastscan - ATPG
  BSD Architect - JTAG insertion
  Fastscan Microtest - Memory testing ATPG

Synopsys’s
  Test Compiler (Integrated with Design Compiler Synthesis too)
  Testgen - Internal scan & Jtag insertion, ATPG
  Tetramax - ATPG
  DFT Compiler (New scan insertion tool integrated with Design Compiler)
INSTRUCTION LATCH DFT ADVISOR DOFILE & TRANSCRIPT

add clocks 0 /clk
set sys mode dft
run
insert test logic
write netlist scan/instr_scan.vhd -vhdl -replace
write atpg setup scripts/instr_scan -replace
exit -d

// Begin scan chain identification process, memory elements = 64.
// Begin scannable cell rules checking for 64 non-scan memory elements.
// 64 non-scan memory elements identified as scannable.
// Begin scan clock rules checking.
// 1 scan clock/set/reset lines have been identified.
// All scan clocks successfully passed off-state check.
// All scan clocks successfully passed capture ability check.
// command: run
// Number of targeted sequential instances = 64
// Performing scan identification ...
// Total sequential instances identified = 64
// command: insert test logic
// Warning: Flattened model has been freed
// command: write netlist scan/instr_scan.vhd -vhdl -replace
// Writing VHDL netlist ...
// command: write atpg setup scripts/instr_scan -replace
// command: exit -d
21 cascade:/home/cascade_c/laryt/training/vhdl/atpg>
fastscan scan/instr_scan.vhd -vhdl -lib tc240ct.fs_lib -top instr_latch
-dofile scripts/instr_fs.do -nogui -log log/instr_fs.log -replace
// FastScan v8.6_4.7  Fri Oct  8 19:35:34 PDT 1999
// Copyright (c) Mentor Graphics Corporation, 1982-1998, All Rights Reserved.
// UNPUBLISHED, LICENSED SOFTWARE.
// CONFIDENTIAL AND PROPRIETARY INFORMATION WHICH IS THE
// PROPERTY OF MENTOR GRAPHICS CORPORATION OR ITS LICENSORS.
//
// Mentor Graphics software executing under Sun SPARC SunOS.
///
// Compiling library ...
// Reading VHDL Netlist ...
// Elaboration phase ...
// Circuit generation phase ...
// Cleanup phase ...
// command: dofile scripts/instr_scan.dofile
// command: add scan groups grp1 scripts/instr_scan.testproc
// command: add scan chains chain1 grp1 scan_in1 CURRENT_INSTR2(0)
// command: add clocks 0 CLK
// command: set system mode atpg
// Flattening process completed, design_cells=160 leaf_cells=160
// library_primitives=352 sim_gates=720 PIs=36 POs=64.
// Begin circuit learning analyses.
// equivalent gates=0 classes=0 CPU time=0.00 sec.
// Learning completed, implications=0, tied_gates=128, CPU time=0.00 sec.
// Begin scan chain identification process, memory elements = 64.
// Reading group test procedure file scripts/instr_scan.testproc.
// Simulating load/unload procedure in grp1 test procedure file.
// Chain = chain1 successfully traced with scan_cells = 64.
// Begin scan clock rules checking.
// 1 scan clock/set/reset lines have been identified.

All scan clocks successfully passed off-state check.
All scan clocks successfully passed capture ability check.
Capture clock is set to CLK.
command: add faults -all
command: run
Simulation performed for #gates = 720 #faults = 1218
system mode = ATPG pattern source = internal patterns
#patterns test #faults #faults #eff. # test process
simulated coverage in list detected patterns patterns CPU time
begin random patterns: capture clock = /CLK, observe point = MASTER
32 98.94% 17 1201 17 17 0.00 sec
64 100.00% 0 17 3 20 0.00 sec
command: report stat

Statistics report
#faults #faults
fault class (coll.) (total)
----------------------- ------ ------
FU (full) 1220 1608
DS (det_simulation) 834 1218
DI (det_implication) 386 390
----------------------- ------ ------
test_coverage 100.00% 100.00%
fault_coverage 100.00% 100.00%
atpg_effectiveness 100.00% 100.00%
----------------------- ------ ------
#test_patterns 20
#simulated_patterns 96
CPU_time (secs) 1.6
----------------------- ------ ------
command: exit -d
Test circuitry for generating memory test patterns on-chip and verifying the memory response.

Many RAMs can share the same BIST logic to reduce overhead.

Data, addresses and control signals are generated simultaneously.

Memory outputs are fed to a signature register which provides compressed data used to determine if the memories are fault free.
TESTGEN JTAG SCAN INSERTION FLOW:
1. Create the following files:
   - TSC: Drives the scan insertion
   - BSR: List and order of signals to insertion boundary scan cells
   - TPI: Timing information for pattern generation
   - PINMAP: Needed to generate BSDL file

2. Compile the netlist and insert boundary scan and jtag controller
   - netcompile cpu_core.vhd -restart
   - lite
   - netcompile cpu_core.vhd

3. Verify the JTAG logic was inserted correctly
   - drc -jtagverify

4. Import the JTAG functional test patterns
   - testpat -import cpu_core.jtv

5. Create a target fault list and run the fault-simulation, then report the faults detected
   - faultlist -expect faultsim -expect faultrep -names -detected

6. Write out the test patterns for the tester in TSTL2 format
   - testpat -tester tstl2
Up and Coming CAD software to reduce the Design time

The most difficult task today is timing closure after layout, usually the customer must first synthesize using WLM, simulate then give the design to the ASIC vendor for layout.

If layout does not meet timing the designer needs to go back and re-synthesize and re-optimize the design to meet timing requirements, then repeat the layout step

Also the inserted test logic (internal scan, jtag, memory BIST) will cause timing some errors.

Solution: Insert test logic in the synthesis tool and get placement information at the front of the design cycle.

Physical Synthesis Tools: Run synthesis and placement at the same time this produces more realistic timing, WLM can be 30% off

Scan and Jtag insertion are integrated into the synthesis tool, so that the synthesis tool can optimize with the jtag and internal scan cells
library ieee;
use ieee.std_logic_1164.all;

tenity tap_control is port(
    trst, tms, tck: in std_logic;
    tap_out: out std_logic_vector(8 downto 0));
begin
state_comb:process(tms,trst,tck,present_state)
begin
if (trst = '0') then
    tap_out <= "00000000"
    next_state <= test_logic_reset;
else
end
case present_state is
  when test_logic_reset =>
    tap_out <= "101100100";
  if (tms = '0') then
    next_state <= run_test_idle;
  else
    next_state <= test_logic_reset;
  end if;
when run_test_idle =>
  tap_out <= "101100100";
  if (tms = '1') then
    next_state <= select_dr;
  else
    next_state <= run_test_idle;
  end if;
when select_dr =>
  tap_out <= "1011010100";
  if (tms = '0') then
    next_state <= capture_dr;
  else
    next_state <= select_ir;
  end if;
when capture_dr =>
  tap_out <= "101100100";
  if (tms = '0') then
    next_state <= shift_dr;
  else
    next_state <= exit1_dr;
  end if;
when shift_dr =>
  tap_out <= "110100110";
  if (tms = '1') then
    next_state <= exit1_dr;
  else
    next_state <= shift_dr;
  end if;
when exit1_dr =>
  tap_out <= "100100110";
  if (tms = '0') then
    next_state <= pause_dr;
  else
    next_state <= update_dr;
  end if;
when pause_dr =>
  tap_out <= "100100100";
  if (tms = '1') then
    next_state <= exit2_dr;
  else
    next_state <= pause_dr;
  end if;
when exit2_dr =>
  tap_out <= "110100100";
  if (tms = '1') then
    next_state <= update_dr;
  else
    next_state <= shift_dr;
  end if;
when update_dr =>
  tap_out <= "100101100";
  if (tms = '1') then
    next_state <= select_dr;
  else
    next_state <= run_test_idle;
  end if;
when select_ir => tap_out <= "110100100";
    if (tms = '0') then next_state <= capture_ir;
    else next_state <= test_logic_reset; end if;
when capture_ir => tap_out <= "101100100"
    if (tms = '0') then next_state <= shift_ir;
    else next_state <= exit1_ir; end if;
when shift_ir => tap_out <= "111110100"
    if (tms = '1') then next_state <= exit1_ir;
    else next_state <= shift_ir; end if;
when exit1_ir => tap_out <= "100100100"
    if(tms = '0') then next_state <= pause_ir;
    else next_state <= update_ir; end if;
when pause_ir => tap_out <= "101101100"
    if (tms = '1') then next_state <= exit2_ir;
    else next_state <= pause_ir; end if;
when exit2_ir => tap_out <= "111100100"
    if (tms = '1') then next_state <= update_ir;
    else next_state <= shift_ir; end if;
when update_ir => tap_out <= "101101100"
    if (tms = '1') then next_state <= select_dr;
    else next_state <= run_test_idle; end if;
end case;
end if;
end process state_comb;
state_clocked:process(tck) begin
    if rising_edge(tck) then present_state <= next_state;
    end if;
end process state_clocked;
end arch1;
library ieee;
use ieee.std_logic_1164.all;
package tap_pkg is
  component tap_control port(
    trst, tms, tck : in std_logic;
    tap_out : out std_logic_vector(8 downto 0));
  end component;
end tap_pkg;
library ieee;
use ieee.std_logic_1164.all;
entity tap_testbench is
end tap_testbench;
use work.tap_pkg.all;
arquitectura tap_test of tap_testbench is
  signal tck,tms,trst : std_logic;
  signal tap_out : std_logic_vector(8 downto 0);
  type test_vector is record
    tck,tms,tck,trst : std_logic;
    tap_out : std_logic_vector(8 downto 0);
  end record;
  type test_vector_array is array(natural range<>) of test_vector;
  constant test_vectors: test_vector_array := (
    -- test_logic_reset
    (trst => '0', tms => '1', tck => '1', tap_out => "000000000"), -- 0 ns
    (trst => '0', tms => '1', tck => '0', tap_out => "000000000"), -- 50 ns
    (trst => '1', tms => '0', tck => '1', tap_out => "101100100"), -- 100 ns
    (trst => '1', tms => '0', tck => '0', tap_out => "101100100"), -- 150 ns
    ...);
-- run_test_idle
(trst => '1', tms => '1', tck => '1', tap_out => "101100100"), -- 200 ns
(trst => '1', tms => '1', tck => '0', tap_out => "101100100"), -- 250 ns
-- select_dr
(trst => '1', tms => '0', tck => '1', tap_out => "100100100"), -- 300 ns
(trst => '1', tms => '0', tck => '0', tap_out => "100100100"), -- 350 ns
-- capture_dr
(trst => '1', tms => '0', tck => '1', tap_out => "101100100"), -- 400 ns
(trst => '1', tms => '0', tck => '0', tap_out => "101100100"), -- 450 ns
-- shift_dr
(trst => '1', tms => '1', tck => '1', tap_out => "110100110"), -- 500 ns
(trst => '1', tms => '1', tck => '0', tap_out => "110100110"), -- 550 ns
-- exit1_dr
(trst => '1', tms => '0', tck => '1', tap_out => "100100110"), -- 600 ns
(trst => '1', tms => '0', tck => '0', tap_out => "100100110"), -- 650 ns
-- pause_dr
(trst => '1', tms => '1', tck => '1', tap_out => "100100100"), -- 700 ns
(trst => '1', tms => '1', tck => '0', tap_out => "100100100"), -- 750 ns
-- exit2_dr
(trst => '1', tms => '1', tck => '1', tap_out => "110100100"), -- 800 ns
(trst => '1', tms => '1', tck => '0', tap_out => "110100100"), -- 850 ns
-- update_dr
(trst => '1', tms => '1', tck => '1', tap_out => "100101100"), -- 900 ns
(trst => '1', tms => '1', tck => '0', tap_out => "100101100"), -- 950 ns
-- select_dr
(trst => '1', tms => '1', tck => '1', tap_out => "100100100"), -- 1000 ns
(trst => '1', tms => '1', tck => '0', tap_out => "100100100"), -- 1050 ns
TAP CONTROLLER TESTBENCH: TABULAR APPROACH

-- select_ir
(trst => '1', tms => '0', tck => '1', tap_out => "110100100"), --1100 ns
(trst => '1', tms => '0', tck => '0', tap_out => "110100100"), --1150 ns
-- capture_ir
(trst => '1', tms => '0', tck => '1', tap_out => "101100100"), --1200 ns
(trst => '1', tms => '0', tck => '0', tap_out => "101100100"), --1250 ns
-- shift_ir
(trst => '1', tms => '1', tck => '1', tap_out => "111110100"), --1300 ns
(trst => '1', tms => '1', tck => '0', tap_out => "111110100"), --1350 ns
-- exit1_ir
(trst => '1', tms => '0', tck => '1', tap_out => "100100100"), --1400 ns
(trst => '1', tms => '0', tck => '0', tap_out => "100100100"), --1450 ns
-- pause_ir
(trst => '1', tms => '1', tck => '1', tap_out => "101100100"), --1500 ns
(trst => '1', tms => '1', tck => '0', tap_out => "101100100"), --1550 ns
-- exit2_ir
(trst => '1', tms => '1', tck => '1', tap_out => "111100100"), --1600 ns
(trst => '1', tms => '1', tck => '0', tap_out => "111100100"), --1650 ns
-- update_ir
(trst => '1', tms => '0', tck => '1', tap_out => "101101100"), --1700 ns
(trst => '1', tms => '0', tck => '0', tap_out => "101101100"), --1750 ns
-- run_test_idle
(trst => '1', tms => '0', tck => '1', tap_out => "101100100"), --1800 ns
(trst => '1', tms => '0', tck => '0', tap_out => "101100100") --1850 ns);
TAP CONTROLLER TESTBENCH: TABULAR APPROACH

begin
  uut: tap_control port map (tck => tck, trst => trst, tms => tms, tap_out => tap_out);
  verify: process
    variable vector: test_vector;
    variable errors: boolean := false;
    begin
      for i in test_vectors'range loop
        vector := test_vectors(i);
        tck <= vector.tck;
        trst <= vector.trst;
        tms <= vector.tms;
        wait for 50 ns;
        if tap_out /= vector.tap_out then
          assert false
            report "output is wrong value";
          errors := true;
        end if;
      end loop;
      assert not errors
        report "Test vectors failed"
        severity Error;
      assert errors
        report "Test vectors passed"
        severity Note;
      wait;
    end process;
  end verify;
end tap_test;
TAP CONTROL MODELSIM SIMULATION LIST REPORT

<table>
<thead>
<tr>
<th>ns</th>
<th>delta</th>
<th>/tap_testbench/uut/tms</th>
<th>/tap_testbench/uut/tck</th>
<th>/tap_testbench/uut/tap_out</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>+2</td>
<td>1 1 000000000 test_logic_reset test_logic_reset</td>
<td></td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>+1</td>
<td>1 0 000000000 test_logic_reset test_logic_reset</td>
<td></td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>+2</td>
<td>0 1 101100100 test_logic_reset run_test_idle</td>
<td></td>
<td></td>
</tr>
<tr>
<td>150</td>
<td>+1</td>
<td>0 0 101100100 test_logic_reset run_test_idle</td>
<td></td>
<td></td>
</tr>
<tr>
<td>200</td>
<td>+3</td>
<td>1 1 101100100 run_test_idle select_dr</td>
<td></td>
<td></td>
</tr>
<tr>
<td>250</td>
<td>+1</td>
<td>1 0 101100100 run_test_idle select_dr</td>
<td></td>
<td></td>
</tr>
<tr>
<td>300</td>
<td>+3</td>
<td>0 1 100100100 select_dr capture_dr</td>
<td></td>
<td></td>
</tr>
<tr>
<td>350</td>
<td>+1</td>
<td>0 0 100100100 select_dr capture_dr</td>
<td></td>
<td></td>
</tr>
<tr>
<td>400</td>
<td>+3</td>
<td>0 1 101100100 capture_dr shift_dr</td>
<td></td>
<td></td>
</tr>
<tr>
<td>450</td>
<td>+1</td>
<td>0 0 101100100 capture_dr shift_dr</td>
<td></td>
<td></td>
</tr>
<tr>
<td>500</td>
<td>+3</td>
<td>1 1 110100110 shift_dr exit1_dr</td>
<td></td>
<td></td>
</tr>
<tr>
<td>550</td>
<td>+1</td>
<td>1 0 110100110 shift_dr exit1_dr</td>
<td></td>
<td></td>
</tr>
<tr>
<td>600</td>
<td>+3</td>
<td>0 1 100100110 exit1_dr pause_dr</td>
<td></td>
<td></td>
</tr>
<tr>
<td>650</td>
<td>+1</td>
<td>0 0 100100110 exit1_dr pause_dr</td>
<td></td>
<td></td>
</tr>
<tr>
<td>700</td>
<td>+3</td>
<td>1 1 100100100 pause_dr exit2_dr</td>
<td></td>
<td></td>
</tr>
<tr>
<td>750</td>
<td>+1</td>
<td>1 0 100100100 pause_dr exit2_dr</td>
<td></td>
<td></td>
</tr>
<tr>
<td>800</td>
<td>+3</td>
<td>1 1 110100100 exit2_dr update_dr</td>
<td></td>
<td></td>
</tr>
<tr>
<td>850</td>
<td>+1</td>
<td>1 0 110100100 exit2_dr update_dr</td>
<td></td>
<td></td>
</tr>
<tr>
<td>900</td>
<td>+3</td>
<td>1 1 100100110 update_dr select_dr</td>
<td></td>
<td></td>
</tr>
<tr>
<td>950</td>
<td>+1</td>
<td>1 0 100100110 update_dr select_dr</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1000</td>
<td>+21</td>
<td>1 1 100100100 select_dr select_ir</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Instruction Latch Block Diagram

CONTROL

clk

Latch_instr

INSTRUCTION LATCH

Current_instr1

Current_instr2

32

32

32

PROGRAM COUNTER

DATAPATH

CONTROL
library ieee;
use ieee.std_logic_1164.all;
entity instr_latch is
  generic (instr_size: integer := 32);
  port ( clk, latch_instr: in std_logic;
         instr: in std_logic_vector(instr_size-1 downto 0);
         current_instr1: out std_logic_vector(instr_size-1 downto 0);
         current_instr2: out std_logic_vector(instr_size-1 downto 0) );
end instr_latch;

architecture arch1 of instr_latch is
begin
  process
  begin
    wait until clk'event and clk = '1';
    if (latch_instr = '1') then
      current_instr1 <= instr;
      current_instr2 <= instr;
    end if;
  end process;
end arch1;
INSTRUCTION LATCH TESTBENCH: PROCEDURAL APPROACH

library ieee;
use ieee.std_logic_1164.all;
package instr_pkgs is
  component instr_latch
    generic (instr_size : integer := 32);
    port (clk, latch_instr : in std_logic;
          instr : in std_logic_vector(instr_size-1 downto 0);
          current_inst1 : out std_logic_vector(instr_size-1 downto 0);
          current_inst2 : out std_logic_vector(instr_size-1 downto 0));
  end component;
end instr_pkgs;

library ieee;
use ieee.std_logic_1164.all;
use IEEE.std_logic_signed.all;
use IEEE.std_logic_arith.all;
use work.instr_pkgs.all;
architecture instr_test of instr_testbench is

library ieee;
use ieee.std_logic_1164.all;
use IEEE.std_logic_signed.all;
use IEEE.std_logic_arith.all;
use work.instr_pkgs.all;
arithmetic instr_test of instr_testbench is
INSTRUCTION LATCH TESTBENCH: PROCEDURAL APPROACH

component instr_latch
    port (clk, latch_instr : in std_logic;
    instr : in std_logic_vector(instr_size-1 downto 0);
    current_instr1, current_instr2 : out std_logic_vector(instr_size-1 downto 0));
end component;

signal clk : std_logic := '0';
signal latch_instr : std_logic := '1';
signal instr : std_logic_vector(instr_size-1 downto 0) := (others => '0');
signal current_instr1, current_instr2 : std_logic_vector(instr_size-1 downto 0);

begin
    uu1: instr_latch port map ( 
        clk => clk, latch_instr => latch_instr, instr => instr, 
        current_instr1 => current_instr1, current_instr2 => current_instr2);

    clk <= not(clk) after 50 ns;

verify:process
begin
    for i in 0 to 1000 loop
        wait until clk'event and clk='1' and latch_instr = '1';
        current_instr1 <= instr;
        current_instr2 <= instr;
        instr <= instr + 1;
    end loop;
end process;
end instr_test;
<table>
<thead>
<tr>
<th>ns</th>
<th>/instr_testbench/clk</th>
<th>delta</th>
<th>/instr_testbench/instr</th>
<th>latching</th>
<th>/instr_testbench/current_instr1</th>
<th>/instr_testbench/current_instr2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>00000000</td>
<td>XXXXXXXXXXXX XXXXXXXXXXXX</td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>00000001</td>
<td>00000000 00000000</td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>00000001</td>
<td>00000000 00000000</td>
<td></td>
</tr>
<tr>
<td>150</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>00000002</td>
<td>00000001 00000001</td>
<td></td>
</tr>
<tr>
<td>200</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>00000002</td>
<td>00000001 00000001</td>
<td></td>
</tr>
<tr>
<td>250</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>00000003</td>
<td>00000002 00000002</td>
<td></td>
</tr>
<tr>
<td>300</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>00000003</td>
<td>00000002 00000002</td>
<td></td>
</tr>
<tr>
<td>350</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>00000004</td>
<td>00000003 00000003</td>
<td></td>
</tr>
<tr>
<td>400</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>00000004</td>
<td>00000003 00000003</td>
<td></td>
</tr>
<tr>
<td>450</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>00000005</td>
<td>00000004 00000004</td>
<td></td>
</tr>
<tr>
<td>500</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>00000005</td>
<td>00000004 00000004</td>
<td></td>
</tr>
<tr>
<td>550</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>00000006</td>
<td>00000005 00000005</td>
<td></td>
</tr>
<tr>
<td>600</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>00000006</td>
<td>00000005 00000005</td>
<td></td>
</tr>
<tr>
<td>650</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>00000007</td>
<td>00000006 00000006</td>
<td></td>
</tr>
<tr>
<td>700</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>00000007</td>
<td>00000006 00000006</td>
<td></td>
</tr>
<tr>
<td>750</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>00000008</td>
<td>00000007 00000007</td>
<td></td>
</tr>
<tr>
<td>800</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>00000008</td>
<td>00000007 00000007</td>
<td></td>
</tr>
<tr>
<td>850</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>00000009</td>
<td>00000008 00000008</td>
<td></td>
</tr>
<tr>
<td>900</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>00000009</td>
<td>00000008 00000008</td>
<td></td>
</tr>
<tr>
<td>950</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0000000A</td>
<td>00000009 00000009</td>
<td></td>
</tr>
<tr>
<td>1000</td>
<td>21</td>
<td>0</td>
<td>1</td>
<td>0000000A</td>
<td>00000009 00000009</td>
<td></td>
</tr>
</tbody>
</table>
# ATPG TEST COVERAGE FOR INSTRUCTION LATCH

Statistics report

<table>
<thead>
<tr>
<th>fault class</th>
<th>#faults (coll.)</th>
<th>#faults (total)</th>
</tr>
</thead>
<tbody>
<tr>
<td>FU (full)</td>
<td>1220</td>
<td>1608</td>
</tr>
<tr>
<td>DS (det_simulation)</td>
<td>834</td>
<td>1218</td>
</tr>
<tr>
<td>DI (det_implication)</td>
<td>386</td>
<td>390</td>
</tr>
<tr>
<td>test_coverage</td>
<td>100.00%</td>
<td>100.00%</td>
</tr>
<tr>
<td>fault_coverage</td>
<td>100.00%</td>
<td>100.00%</td>
</tr>
<tr>
<td>atpg_effectiveness</td>
<td>100.00%</td>
<td>100.00%</td>
</tr>
<tr>
<td>#test_patterns</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>#simulated_patterns</td>
<td>96</td>
<td></td>
</tr>
<tr>
<td>CPU_time (secs)</td>
<td>1.6</td>
<td></td>
</tr>
</tbody>
</table>
library ieee;
use ieee.std_logic_1164.all;
entity datapath is
  generic (instr_size: integer := 32;
            data_size: integer := 16);
  port (clk, reset, reset_alu, read_a, read_b: in std_logic;
        sel_instr_or_b, sel_instr_or_alu: in std_logic;
        reg_a, reg_b, alu_result: in std_logic_vector(data_size-1 downto 0);
        current_instr: in std_logic_vector(instr_size-1 downto 0);
        latch_flags, alu_zero, alu_neg, alu_carry: in std_logic;
        psw_zero, psw_neg, psw_carry: in std_logic;
        zero_flag, neg_flag, carry_flag: out std_logic;
        alu_a, alu_b, reg_c: out std_logic_vector(data_size-1 downto 0));
end datapath;
architecture arch1 of datapath is
  signal pswl_zero, pswl_carry, pswl_neg: std_logic;
begin
  process
  begin
    wait until clk'event and clk = '1';
    -- either reset the alu output or transfer reg_a to alu_a
    if (reset_alu = '1') then alu_b <= (others => '0');
      elsif (read_a = '1') then alu_a <= reg_a;
    end if;
    if (reset_alu = '1') then  alu_b <= (others => '0');
      elsif (read_b = '1') then alu_b <= reg_b;
      elsif (sel_instr_or_b = '1') then alu_b <= "00000000" & current_instr(7 downto 0);
    end if;
    if (reset = '1') then pswl_zero  <= '0'; pswl_neg   <= '0'; pswl_carry <= '0';
      else pswl_zero  <= psw_zero; pswl_neg   <= psw_neg; pswl_carry <= psw_carry;
    end if;
  end process;
end architecture arch1;
-- Mux alu_result and current_instr data to reg_c
process (current_instr, alu_result, sel_instr_or_alu)
begin
  if (sel_instr_or_alu = '1') then
    reg_c <= current_instr(data_size-1 downto 0);
  else
    reg_c <= alu_result;
  end if;
end process;

-- Mux flags bwt popped and alu outputs
-- Return instrn alone requires popped flags
process (current_instr, pswl_zero, pswl_neg, pswl_carry,
        alu_zero, alu_neg, alu_carry)
begin
  if (current_instr(instr_size-1 downto 24) = "00001000") then
    zero_flag <= pswl_zero; neg_flag <= pswl_neg; carry_flag <= pswl_carry;
  else
    zero_flag <= alu_zero; neg_flag <= alu_neg; carry_flag <= alu_carry;
  end if;
end process;
end arch1;
ATPG TEST COVERAGE FOR DATAPATH

Statistics report

<table>
<thead>
<tr>
<th>fault class</th>
<th>#faults (coll.)</th>
<th>#faults (total)</th>
</tr>
</thead>
<tbody>
<tr>
<td>FU (full)</td>
<td>956</td>
<td>1340</td>
</tr>
<tr>
<td>DS (det_simulation)</td>
<td>640</td>
<td>1022</td>
</tr>
<tr>
<td>DI (det_implication)</td>
<td>214</td>
<td>216</td>
</tr>
<tr>
<td>UU (unused)</td>
<td>88</td>
<td>88</td>
</tr>
<tr>
<td>AU (atpg_untestable)</td>
<td>14</td>
<td>14</td>
</tr>
<tr>
<td>test_coverage</td>
<td>98.39%</td>
<td>98.88%</td>
</tr>
<tr>
<td>fault_coverage</td>
<td>89.33%</td>
<td>92.39%</td>
</tr>
<tr>
<td>atpg_effectiveness</td>
<td>100.00%</td>
<td>100.00%</td>
</tr>
</tbody>
</table>

#test_patterns        | 34              |
#simulated_patterns   | 288             |
CPU_time (secs)       | 1.5             |
REGISTER FILE BLOCK DIAGRAM

INSTRUCTION LATCH

Addr_A  Addr_B

clk  reset

write_reg_c

reg_a  reg_b  Addr_C  Reg_C

16  16  7  16

DATAPATH
library ieee;
use ieee.std_logic_1164.all;

entity regfile is
    generic (addr_size: integer := 6;
             data_size: integer := 16);
    port (clk,reset: in std_logic;
          write_c: in std_logic;
          reg_c: in std_logic_vector(data_size-1 downto 0);
          addr_a: in std_logic_vector(addr_size-1 downto 0);
          addr_b: in std_logic_vector(addr_size-1 downto 0);
          addr_c: in std_logic_vector(addr_size-1 downto 0);
          reg_a, reg_b: out std_logic_vector(data_size-1 downto 0)
          );
end regfile;
architecture arch1 of regfile is 
   type regfile is array (0 to 3) of std_logic_vector(data_size-1 downto 0);
   signal reg_array: regfile;
begin
   process
   begin
      wait until clk'event and clk = '1';
      if (reset = '1') then
         for i in 0 to 3 loop
            reg_array(i) <= (others => '0');
         end loop;
      end if;      
      if (write_c = '1') then
         reg_array(conv_integer(unsigned(addr_c(1 downto 0)))) <= reg_c;
      end if;
   end process;
   process (addr_a, reg_array)
   begin
      reg_a <= reg_array(conv_integer(unsigned(addr_a(1 downto 0))));
   end process;
   process (addr_b, reg_array)
   begin
      reg_b <= reg_array(conv_integer(unsigned(addr_b(1 downto 0))));
   end process;
end arch1;
ATPG TEST COVERAGE FOR REGFILE

Statistics report

<table>
<thead>
<tr>
<th>fault class</th>
<th>#faults (coll.)</th>
<th>#faults (total)</th>
</tr>
</thead>
<tbody>
<tr>
<td>FU (full)</td>
<td>1840</td>
<td>2624</td>
</tr>
<tr>
<td>DS (det_simulation)</td>
<td>1299</td>
<td>2081</td>
</tr>
<tr>
<td>DI (det_implication)</td>
<td>388</td>
<td>390</td>
</tr>
<tr>
<td>UU (unused)</td>
<td>152</td>
<td>152</td>
</tr>
<tr>
<td>RE (redundant)</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>test_coverage</th>
<th>fault_coverage</th>
<th>atpg_effectiveness</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>100.00%</td>
<td>91.68%</td>
<td>100.00%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>#test_patterns</th>
<th>#simulated_patterns</th>
<th>CPU_time (secs)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>46</td>
<td>224</td>
<td>1.8</td>
</tr>
</tbody>
</table>
## ALU TRUTH TABLE

<table>
<thead>
<tr>
<th>ALU_CTRL</th>
<th>FUNCTION NAME</th>
<th>OPERATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>000000</td>
<td>OP_ADD</td>
<td>A + B</td>
</tr>
<tr>
<td>000001</td>
<td>OP_ADD_PLUS_ONE</td>
<td>A + B + 1</td>
</tr>
<tr>
<td>000010</td>
<td>OP_A</td>
<td>A</td>
</tr>
<tr>
<td>000011</td>
<td>OP_A_PLUS_ONE</td>
<td>A + 1</td>
</tr>
<tr>
<td>010001</td>
<td>OP_A_AND_B</td>
<td>A &amp; B</td>
</tr>
<tr>
<td>010010</td>
<td>OP_NOT_A_AND_B</td>
<td>!A &amp; B</td>
</tr>
<tr>
<td>010011</td>
<td>OP_B</td>
<td>B</td>
</tr>
<tr>
<td>011000</td>
<td>OP_NOT_A_AND_NOT_B</td>
<td>!A &amp; !B</td>
</tr>
<tr>
<td>011001</td>
<td>OP_A_XNOR_B</td>
<td>A xnor B</td>
</tr>
<tr>
<td>011010</td>
<td>OP_NOT_A</td>
<td>!A</td>
</tr>
<tr>
<td>011011</td>
<td>OP_NOT_A_OR_B</td>
<td>!A</td>
</tr>
<tr>
<td>000100</td>
<td>OP_SUB_MINUS_ONE</td>
<td>A - B - 1</td>
</tr>
<tr>
<td>000101</td>
<td>OP_SUB</td>
<td>A - B</td>
</tr>
<tr>
<td>000110</td>
<td>OP_A_MINUS_ONE</td>
<td>A - 1</td>
</tr>
<tr>
<td>010100</td>
<td>OP_A_AND_NOT_B</td>
<td>A &amp; !B</td>
</tr>
<tr>
<td>010110</td>
<td>OP_A_XOR_B</td>
<td>A XOR B</td>
</tr>
<tr>
<td>010111</td>
<td>OP_A_OR_B</td>
<td>A</td>
</tr>
<tr>
<td>011100</td>
<td>OP_NOT_B</td>
<td>!B</td>
</tr>
<tr>
<td>011101</td>
<td>OP_A_OR_NOT_B</td>
<td>A</td>
</tr>
<tr>
<td>011110</td>
<td>OP_A_NAND_B</td>
<td>A NAND B</td>
</tr>
<tr>
<td>010000</td>
<td>OP_ALL_ZEROS</td>
<td>‘0’</td>
</tr>
<tr>
<td>011111</td>
<td>OP_ALL_ONES</td>
<td>‘1’</td>
</tr>
</tbody>
</table>
library ieee;
use ieee.std_logic_1164.all;
package alu_pkgs is
-- ALU functions
constant op_add : std_logic_vector(5 downto 0) := "000000";
constant op_add_plus_one : std_logic_vector(5 downto 0) := "000001";
constant op_a : std_logic_vector(5 downto 0) := "000010";
constant op_a_plus_one : std_logic_vector(5 downto 0) := "000011";
constant op_a_and_b : std_logic_vector(5 downto 0) := "010001";
constant op_nota_and_b : std_logic_vector(5 downto 0) := "010010";
constant op_b : std_logic_vector(5 downto 0) := "010011";
constant op_nota_and_notb : std_logic_vector(5 downto 0) := "011000";
constant op_a_xor_b : std_logic_vector(5 downto 0) := "010110";
constant op_a_or_b : std_logic_vector(5 downto 0) := "010111";
constant op_notb : std_logic_vector(5 downto 0) := "011100";
constant op_a_or_notb : std_logic_vector(5 downto 0) := "011101";
constant op_a_nand_b : std_logic_vector(5 downto 0) := "011110";
constant op_all_zeros : std_logic_vector(5 downto 0) := "010000";
constant op_all_ones : std_logic_vector(5 downto 0) := "011111";
end alu_pkgs;
library ieee;
use ieee.std_logic_1164.all;
use IEEE.std_logic_signed.all;
use IEEE.std_logic_arith.all;
use work.alu_pkgs.all;

entity alu is
-- Generic data_size allows the user to easily change the data_size of the alu
  generic (data_size: integer := 16);
  port( clk,reset : in std_logic;
        alu_a, alu_b : in std_logic_vector(data_size-1 downto 0);
        alu_ctrl : in std_logic_vector(5 downto 0);
        latch_result, latch_flag : in std_logic;
        alu_result : out std_logic_vector(data_size-1 downto 0);
        zero_flag, neg_flag, carry_flag : out std_logic);
end alu;

architecture arch1 of alu is
  signal op_result, result : std_logic_vector(data_size-1 downto 0);
  signal alu_zero, alu_carry, signal alu_neg : std_logic;

begin
  alu: process (alu_a, alu_b, alu_ctrl)
  begin
    case alu_ctrl is
      when op_add => result <= alu_a + alu_b;
      when op_add_plus_one => result <= alu_a + alu_b + '1';
      when op_a => result <= alu_a;
    end case;
  end process;
end arch1;
when op_a_plus_one => result <= alu_a + '1';
when op_a_and_b => result <= alu_a and alu_b;
when opNota_and_b => result <= not alu_a and alu_b;
when op_b => result <= alu_b;
when opNota_and_notb => result <= not alu_a and not alu_b;
when op_a_xnor_b => result <= not (alu_a xor alu_b);
when op_nota => result <= not alu_a;
when op_nota_or_b => result <= not alu_a or alu_b;
when op_sub_minus_one => result <= alu_a - alu_b - '1';
when op_sub => result <= alu_a - alu_b;
when op_a_minus_one => result <= alu_a - '1';
when op_a_and_notb => result <= alu_a and not alu_b;
when op_a_xor_b => result <= alu_a xor alu_b;
when op_notb => result <= not alu_b;
when op_a_or_notb => result <= alu_a or not alu_b;
when op_a_or_b => result <= alu_a or alu_b;
when op_a_nand_b => result <= alu_a nand alu_b;
when op_all_zeros => result <= (others => '0');
when op_all_ones => result <= (others => '1');
when others => result <= (others => '0');

end case;

if (result = 0) then alu_zero <= '1';
else alu_zero <= '0';
end if;
if (result < 0) then alu_neg <= '1';
else alu_neg <= '0';
end if;
alu_carry <= '0';
op_result <= result;
end process;

process
begin
wait until clk'event and clk = '1';
if (latch_result = '1') then
alu_result <= op_result;
end if;
if (reset = '1') then
zero_flag <= '0';
neg_flag <= '0';
carry_flag <= '0';
elsif (latch_flag = '1') then
zero_flag <= alu_zero;
neg_flag <= alu_neg;
carry_flag <= alu_carry;
end if;
end process;
end arch1;
library ieee;
use ieee.std_logic_1164.all;
package alu1_pkgs is
    component alu
        generic (data_size : integer := 16);
        port (clk, reset : in std_logic;
            alu_a, alu_b : in std_logic_vector(data_size-1 downto 0);
            alu_ctrl : in std_logic_vector(5 downto 0);
            latch_result, latch_flag : in std_logic;
        alu_result : out std_logic_vector(data_size-1 downto 0);
            zero_flag, neg_flag, carry_flag : out std_logic);
    end component;
end alu1_pkgs;

library ieee;
use ieee.std_logic_1164.all;
useIEEE.std_logic_signed.all;
useIEEE.std_logic_arith.all;
use work.alu_pkgs.all;
use work.alu1_pkgs.all;
architecture alu_test of alu_testbench is
component alu
  port(clk, reset, latch_result, latch_flag : in std_logic;
       alu_a, alu_b : in std_logic_vector(data_size-1 downto 0);
       alu_ctrl : in std_logic_vector(5 downto 0);
       alu_result : out std_logic_vector(data_size-1 downto 0);
       zero_flag, neg_flag, carry_flag: out std_logic);
end component;

signal alu_a, alu_b : std_logic_vector(data_size-1 downto 0) := (others => '0');
signal alu_result : std_logic_vector(data_size-1 downto 0);
signal alu_ctrl : std_logic_vector(5 downto 0) := "000000";
signal clk, reset : std_logic := '0';
signal latch_result, latch_flag : std_logic := '1';
signal zero_flag, neg_flag, carry_flag : std_logic;
signal temp : std_logic_vector(data_size*2-1 downto 0) := (others => '0');

begin
  alu port map (clk=>clk, reset=>reset, alu_a=>alu_a, alu_b=>alu_b, alu_ctrl=>alu_ctrl, latch_result=>latch_result,
                 latch_flag =>latch_flag, alu_result =>alu_result, zero_flag =>zero_flag, neg_flag =>neg_flag, carry_flag =>carry_flag);
  clk <= not(clk) after 50 ns;
  verify: process
  begin
    for j in 0 to 3 loop
      for i in 0 to no_of_vectors loop
        wait until clk'event and clk='1';
        alu_a <= temp(data_size*2-1 downto data_size);
        alu_b <= temp(data_size-1 downto 0);
        temp <= temp + 1;
      end loop;
      alu_ctrl <= alu_ctrl + 1;
    end loop;
  end process; end alu_test;
## ALU MODELSIM SIMULATION OF TESTBENCH

<table>
<thead>
<tr>
<th>ns</th>
<th>/alu_testbench/alu_a</th>
<th>/alu_testbench/clk</th>
<th>/alu_testbench/neg_flag</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>delta</td>
<td>/alu_testbench/alu_b</td>
<td>/alu_testbench/latch_result</td>
</tr>
<tr>
<td></td>
<td></td>
<td>/alu_testbench/temp</td>
<td>/alu_testbench/temp</td>
</tr>
<tr>
<td></td>
<td></td>
<td>/alu_testbench/alu_result</td>
<td>/alu_testbench/latch_flag</td>
</tr>
<tr>
<td></td>
<td></td>
<td>/alu_testbench/alu_ctrl</td>
<td>/alu_testbench/zero_flag</td>
</tr>
<tr>
<td></td>
<td></td>
<td>/alu_testbench/reset</td>
<td>/alu_testbench/carry_flag</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>X</td>
</tr>
<tr>
<td>50</td>
<td>+1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>100</td>
<td>+0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>150</td>
<td>+1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>200</td>
<td>+0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>250</td>
<td>+1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>300</td>
<td>+0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>350</td>
<td>+1</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>400</td>
<td>+0</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>450</td>
<td>+1</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>500</td>
<td>+0</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>550</td>
<td>+1</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>600</td>
<td>+0</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>650</td>
<td>+1</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>700</td>
<td>+0</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>750</td>
<td>+1</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>800</td>
<td>+0</td>
<td>0</td>
<td>7</td>
</tr>
<tr>
<td>850</td>
<td>+1</td>
<td>0</td>
<td>7</td>
</tr>
<tr>
<td>900</td>
<td>+0</td>
<td>0</td>
<td>8</td>
</tr>
<tr>
<td>950</td>
<td>+1</td>
<td>0</td>
<td>8</td>
</tr>
<tr>
<td>1000</td>
<td>+21</td>
<td>0</td>
<td>9</td>
</tr>
</tbody>
</table>
ATPG TEST COVERAGE ALU

Statistics report

<table>
<thead>
<tr>
<th>fault class</th>
<th>#faults (coll.)</th>
<th>#faults (total)</th>
</tr>
</thead>
<tbody>
<tr>
<td>FU (full)</td>
<td>1924</td>
<td>3288</td>
</tr>
<tr>
<td>DS (det_simulation)</td>
<td>1798</td>
<td>3160</td>
</tr>
<tr>
<td>DI (det_implication)</td>
<td>118</td>
<td>120</td>
</tr>
<tr>
<td>UU (unused)</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>RE (redundant)</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

| test_coverage        | 100.00%         | 100.00%         |
| fault_coverage       | 99.58%          | 99.76%          |
| atpg_effectiveness   | 100.00%         | 100.00%         |

#test_patterns         | 83              |
#simulated_patterns    | 83              |
CPU_time (secs)        | 6.2             |