Design for Test
Scan Test

Smith Text: Chapter 14.6
Mentor Graphics Documents:
“Scan and ATPG Process Guide”
“DFTAdvisor Reference Manual”
“Tessent Common Resources Manual for ATPG Products
Top-down test design flow
Sequential circuit testing problem

- External access only to PIs and POs
- Internal state is changed indirectly
- For N PIs and K state variables, must test \(2^{N+K}\) combinations
- Some states difficult to reach, so even more test vectors are needed
Design for Test (DFT)

Flip flop states are difficult to set from PIs A & B
DFT: Scan Design

- Flip flops replaced with “scan” flip flops
- Scan flip flops form a shift register in “scan mode”
- Flip flop states set via “scan input” sc_in
Scan-based test procedure

- Combinational logic inputs = \{X_1 \ldots X_k, Q_1 \ldots Q_n\}
  - X_1 \ldots X_k = primary inputs (PI’s)
  - Q_1 \ldots Q_n = flop-flop outputs

- Combinational logic outputs = \{Z_1 \ldots Z_m, D_1 \ldots D_n\}
  - Z_1 \ldots Z_m = primary outputs (PO’s)
  - D_1 \ldots D_n = flop-flop inputs

- Test procedure:
  1. Apply pattern to combinational logic inputs:
     a) Set scan enable $sc\_en = 1$ and shift pattern into $Q_1 \ldots Q_n$ via scan input $sc\_in$
     b) Apply a pattern to PI’s $X_1 \ldots X_k$
  2. Check combinational logic outputs:
     a) Check PO’s $Z_1 \ldots Z_m$
     b) Set $sc\_en = 0$ and clock the circuit to capture $D_1 \ldots D_n$ in the flip-flops
     c) Set $sc\_en = 1$ and shift out $Q_1 \ldots Q_n$ via scan output $sc\_out$ for verification
Scan type: mux_scan

Standard D flip flop with a mux to select system data vs scan data

ADK library “mux_scan” components:
  sff, sffr_ni, sffs, sffs_macro, sffsr_ni, sffsr_ni_macro
Scan type: clocked_scan

Separate clocks to load system data and scan data

ADK library: no "clocked_scan" component
Scan type: LSSD
(Level-sensitive scan design – IBM)

Three clocks:
1. sys_clock loads system data into the master latch (normal mode)
2. Aclk loads scan data into the master latch
3. Bclk captures master data in the slave latch to drive scan output

ADK library “lssd” components:
  lssd_latch/latchsr/latchr/latches/latches_ni/latchsr_ni
Full vs. partial scan

Full Scan Design

Partial Scan: Increase testability, without affecting critical timing/areas
Scan chain groups

- Scan chains operate in parallel from separate scan inputs to reduce the number of clock cycles to load/unload the chain
- Control from one procedure file
- Can use separate clocks or a common clock
DFT = test point insertion
Choosing a DFT solution

- Full Scan
- Partial Scan
- No Scan or Other DFT Techniques

Areas
- Area Overhead
- Test Generation Effort

- Combinational and Scan-Sequential ATPG (Tessent FastScan)
- Sequential ATPG
DFTadvisor/FastScan Design Flow

Source: ATPG Manual
DFT test flow and commands

DFTAdvisor

Commands:
- add pin constraints
- analyze control signals -auto_fix
  typically use defaults
- set system mode dft
- run
- Insert test logic -number 8
- write netlist <file_name> -verilog
- write atpg setup <file_name>

DFTAdvisor Commands (insert test logic)

FastScan

Commands:
- dofile <file_name>.dofile
- set system mode atpg
  typically use defaults
- create patterns -auto
- save patterns -verilog

FastScan Commands (generate patterns)

Source: DFTadvisor Reference
Basic scan insertion flow
DFTAdvisor supported test structures

**Test Structures**

- **Full Scan**
- **Partial Scan**
- **Wrapper chains**
- **Test Points**

**Sequential ATPG-Based**
- **Automatic**

**SCOAP-Based**
- **Structure-Based**

**Sequential Transparent**
- **Clocked Sequential**

**Sequential ATPG-based**: choose cells with a sequential ATPG algorithm
**SCOAP**: Sandia Controllability Observability Analysis Program (#’s for each ff)
**Automatic**: combine scan selection methods using several techniques
**Structure-based**: look at loop breaking, limiting sequential depth, etc.
**Sequential Transparent**: cut all sequential loops and evaluate
**Clocked Sequential**: cut sequential loops and limit sequential depth
Example DFTadvisor session

- Invoke:
  - `dftadvisor count4.v –lib $ADK/technology/adk.atpg`

- Implement scan **with defaults** (full scan, mux-DFF elements):
  - set system mode setup
  - analyze control signals –auto
  - set system mode dft
  - set scan type mux_scan
  - run
  - insert test logic –scan on
  - write netlist count4_scan.v
  - write atpg setup count4_scan

- Options:
  - insert test logic –scan on –number 3
  - insert test logic –scan on –max_length 20
DFT options

- set scan type mux_scan
  - Others: lssd, clocked_scan
  - Find indicated scan flip flop type in the ATPG library
- setup scan identification “type”, where “type” =
  - full_scan (default)
  - sequential atpg –percent 50
  - clock_sequential [-depth integer]
  - etc.
- insert test logic
  - -scan on/off (insert scan elements; default=on)
  - -test_point on/off (insert test points; default=on)
  - -maxlength n (max scan chain length = n)
  - -number n (divide ffs into n scan chains)
Binary counter
(4-bit)

Synthesized by
Leonardo
count4 – without scan design
Binary counter (4-bit)

Synthesized by Leonardo

DFTAdvisor

Changed to Scan Design

```verbatim
/*
 * DESC: Generated by DFTAdvisor at Wed Nov 30 17:01:15 2005
 */
module count4 ( clock , clear , enable , output , scan_in1 , scan_en );
input clock , clear , enable , scan_in1 , scan_en ;
output [ 3 : 0 ] output ;
wire nx139 , nx30 , nx109 , nx24 , nx99 , nx127 , nx129 , nx87 , nx89 , nx121 , nx79 ;
wire [ 3 : 0 ] $dummy ;
sfr_n1 output_0__rename_rename (.D ( nx79 ) , .SI ( scan_in1 ) , .SE ( scan_en ) , .CLK ( clock ) , .R ( clear ) , .Q ( output [ 0 ] ) , .QB ( $dummy [ 0 ] ));
inw02 ix122 .A ( enable ) , .Y ( nx121 ));
sfr_n1 output_1__rename_rename (.D ( nx89 ) , .SI ( $dummy [ 0 ] ) , .SE ( scan_en ) , .CLK ( clock ) , .R ( clear ) , .Q ( output [ 1 ] ) , .QB ( $dummy [ 1 ] ));
nand02_2x ix130 .AO ( output [ 1 ] ) , .A1 ( output [ 0 ] ) , .Y ( nx129 ));
xor2 ix80 .AO ( output [ 0 ] ) , .A1 ( enable ) , .Y ( nx79 ));
nor02ii ix88 .AO ( nx127 ) , .A1 ( enable ) , .Y ( nx87 ));
endmodule
```
count4 – scan inserted by DFTadvisor
FastScan ATPG session for a circuit containing scan chains

- **Invoke:**
  ```
  fastscan count4_scan.v --lib $ADK/technology/adk.atpg
  ```

- **Generate test pattern file:**
  - `dofile count4_scan.dofile` (defines scan path & procedure)
  - `set system mode atpg`
  - `create patterns -auto`
  - `save patterns -verilog` (write patterns & verilog test bench)
  - `write faults count4_faults.txt` (write fault information to file)
  - `write procfile count4.proc` (write test procedure & timing data)
count4_scan.dofile

// Generated by DFTAdvisor at Wed Nov 30 17:01:33 2014
//
// define group “grp1” of scan chains and their test procedure
add scan groups grp1 count4_scan.do.testproc
// define sc_in and sc_out of scan “chain1” in group “grp1”
add scan chains chain1 grp1 scan_in1 output[3]
// define “clocks” controlling the scan chain
add clocks 0 clear
add clocks 0 clock

Notes:
• Can have multiple scan chains in a group – with a common test procedure
• Can have multiple groups – each with its own test procedure
Test file: scan chain definition and load/unload procedures

scan_group "grp1" =
    scan_chain "chain1" =
        scan_in = "/scan_in1";
        scan_out = "/output[3]";
        length = 4;
    end;

procedure shift "grp1_load_shift" =
    force_sci "chain1" 0;
    force "/clock" 1 20;
    force "/clock" 0 30;
    period 40;
end;

procedure load "grp1_load" =
    force "/clear" 0 0;
    force "/clock" 0 0;
    force "/scan_en" 1 0;
    apply "grp1_load_shift" 4 40;
end;

procedure shift "grp1_unload_shift" =
    measure_sco "chain1" 10;
    force "/clock" 1 20;
    force "/clock" 0 30;
    period 40;
end;

procedure unload "grp1_unload" =
    force "/clear" 0 0;
    force "/clock" 0 0;
    force "/scan_en" 1 0;
    apply "grp1_unload_shift" 4 40;
end;
Test file: scan chain test

// send one pattern through the scan chain

CHAIN_TEST =

    pattern = 0;  (pattern #)
    apply "grp1_load" 0 =  (use grp1_load proc.)
        chain "chain1" = "0011";  (pattern to scan in)
    end;

    apply "grp1_unload" 1 =  (use grp1_unload proc.)
        chain "chain1" = "1100";  (expected pattern scanned out)
    end;

end;
Test file: sample test pattern

// one of 14 patterns for the counter circuit

pattern = 0;
apply "grp1_load" 0 =
    chain "chain1" = "1000";
end;
force "PI" "00110" 1;
measure "PO" "0010" 2;
pulse "/clock" 3;
apply "grp1_unload" 4 =
    chain "chain1" = "0110";
end;
Alternate format

set time scale 1.000000 ns;

timeplate gen_tp1 =
  force_pi 0; (1)
  measure_po 10; (2)
  pulse clock 20 10; (3)
  period 40; (4)
end;

procedure shift =
  scan_group grp1;
  timeplate gen_tp1;
  cycle =
    force_sci;
    measure_sco;
    pulse clock;
  end;
end;

procedure load_unload =
  scan_group grp1;
  timeplate gen_tp1;
  cycle =
    force clear 0;
    force clock 0;
    force scan_en 1;
  end;
apply shift 4;
end;

Timing of op’s within each cycle

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>10</th>
<th>20</th>
<th>30</th>
<th>40</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(2)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(3)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(4)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>clock</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Initial values

- Execute shift proc. 4 times
DFTAdvisor example (Chao Han)

//dofile for dftadvisor
analyze control signals -auto_fix
set scan type mux_scan
set system mode dft
setup scan identification full_scan
run

//specify # scan chains to create
insert test logic -scan on -number 3
//alternative: specify maximum scan chain length
//insert test logic -scan on -max_length 30
write netlist s1423_scan.v -verilog -replace
//write dofile and procedure file for fastscan
write atpg setup s1423_scan -procfile -replace
exit