Design for Test
Scan Test

Smith Text: Chapter 14.6

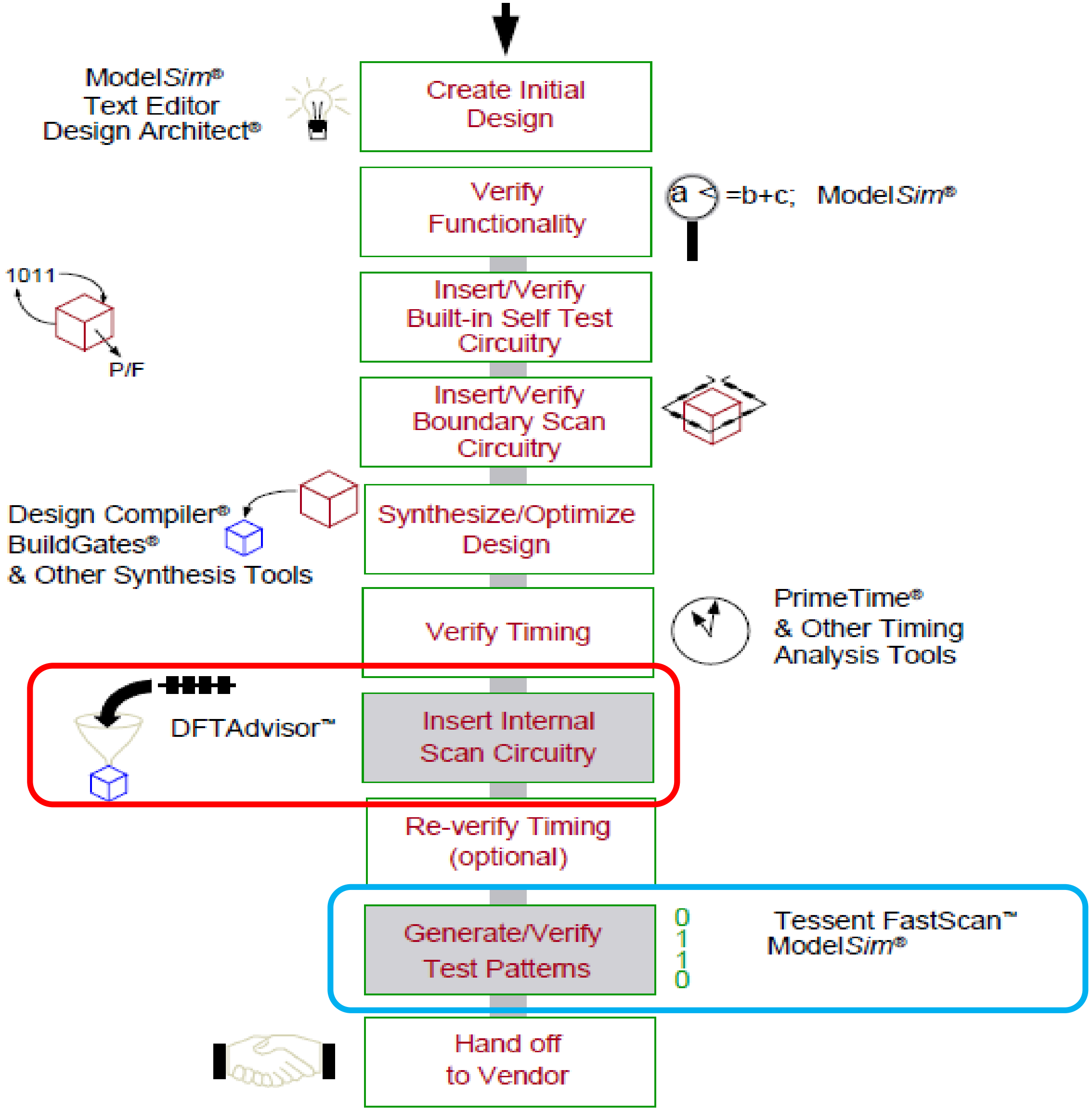
Mentor Graphics Documents:

“Scan and ATPG Process Guide”

“DFTAdvisor Reference Manual”

“Tessent Common Resources Manual for ATPG Products

Top-down test design flow



Create Initial Design

Verify Functionality

Insert/Verify Built-in Self Test Circuitry

Insert/Verify Boundary Scan Circuitry

Synthesize/Optimize Design

Verify Timing

Insert Internal Scan Circuitry

Re-verify Timing (optional)

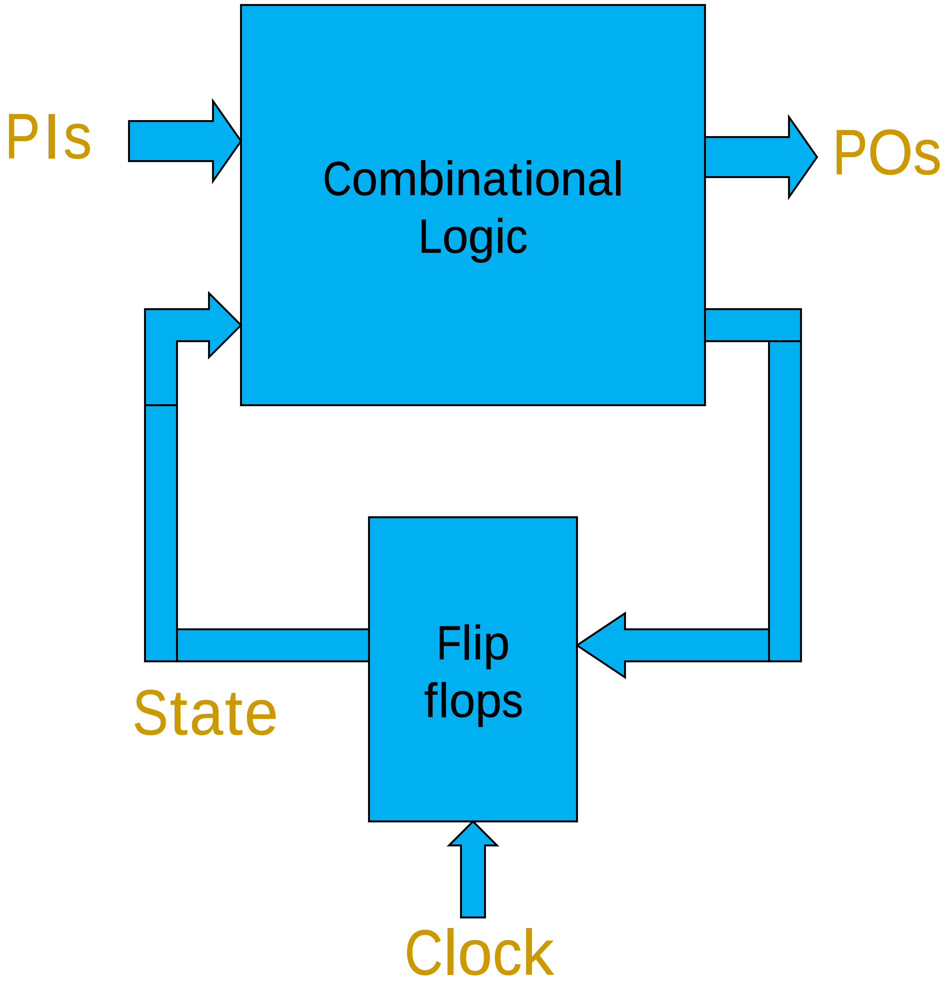
Generate/Verify Test Patterns

Hand off to Vendor

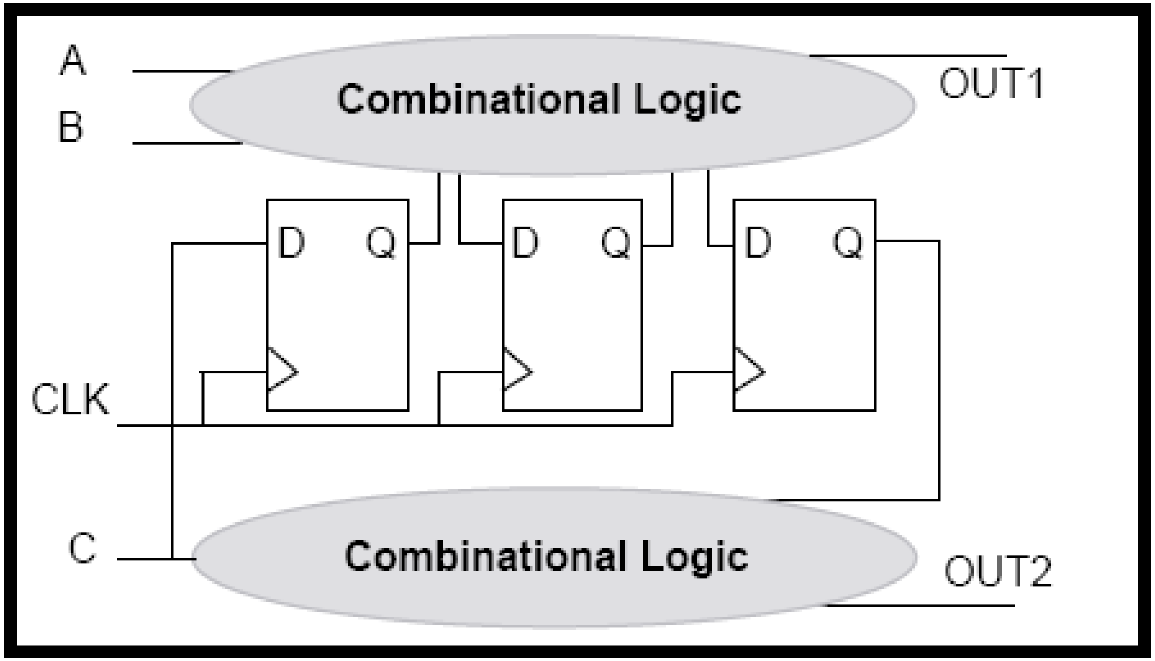
Source: Scan and APTG Process Guide

Sequential circuit testing problem

- Access limited to PIs/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

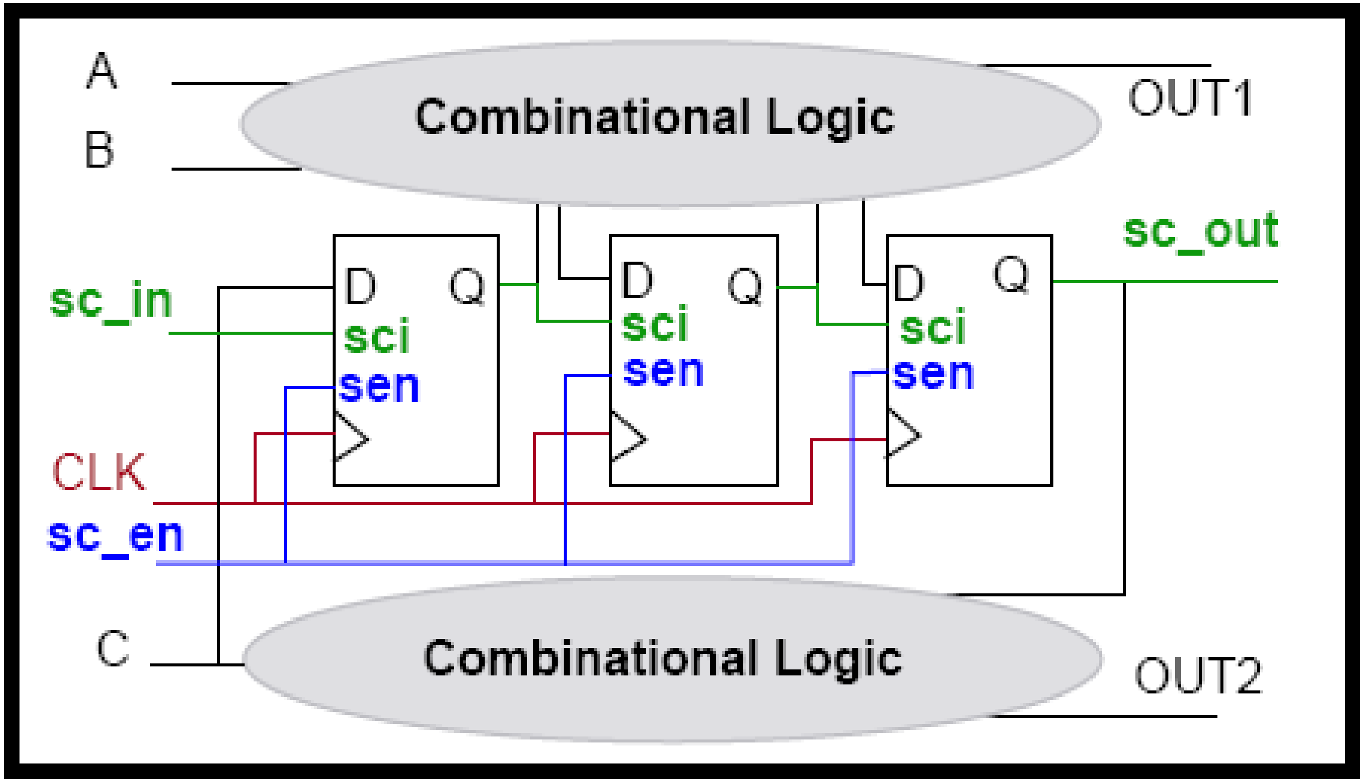


Design for Test (DFT)



Flip flop states 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
- Flip flop states examined via “scan output” sc_out

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:

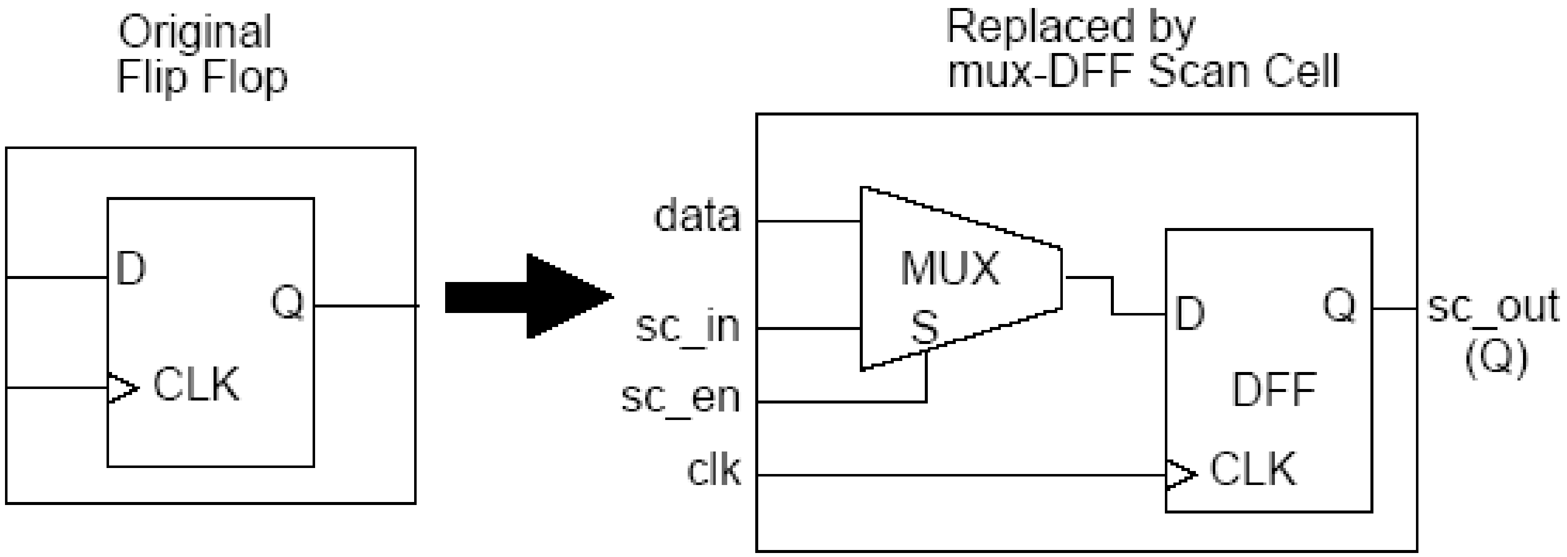
a) Check PO’s $Z_1 \ldots Z_m$

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



BICMOS8HP library “mux_scan” components:

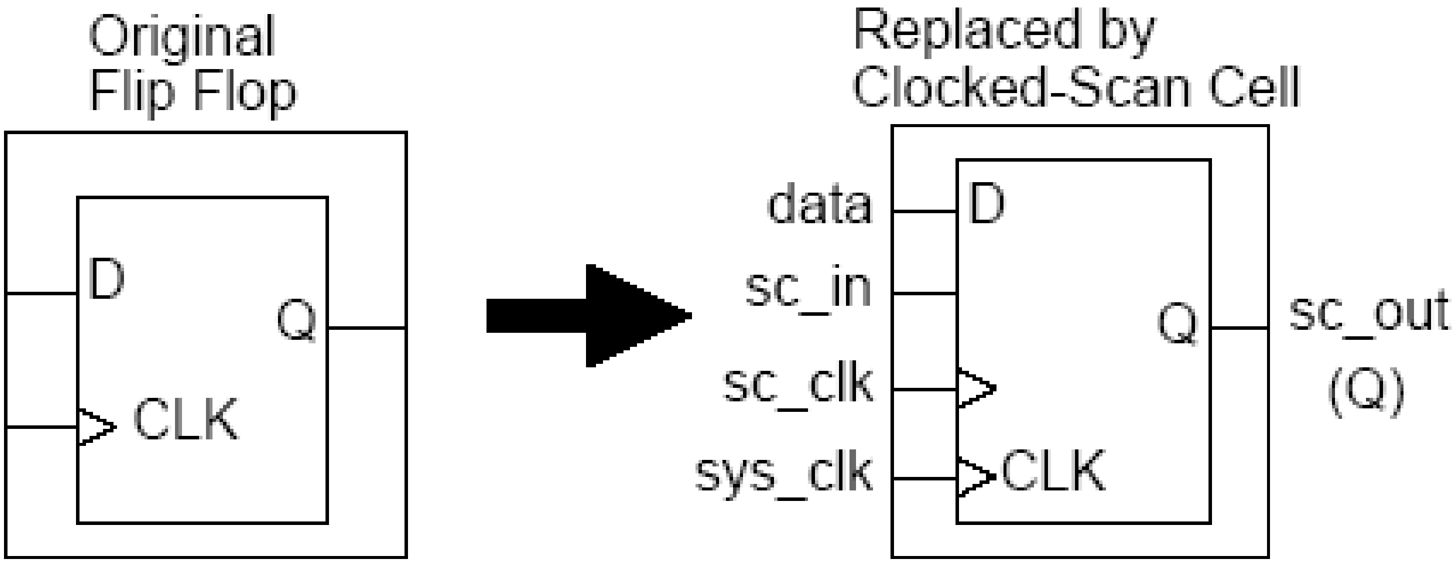
SDFF_x, SDFFR_x, SDFFS_r, SDFFSR_x, SLATSRLV_x

Replacements for:

DFF_x, DFFR_x, DFFS_x, DFFSR_x, LATSRLV_x

Scan type: clocked_scan

Separate clocks to load system data and scan data

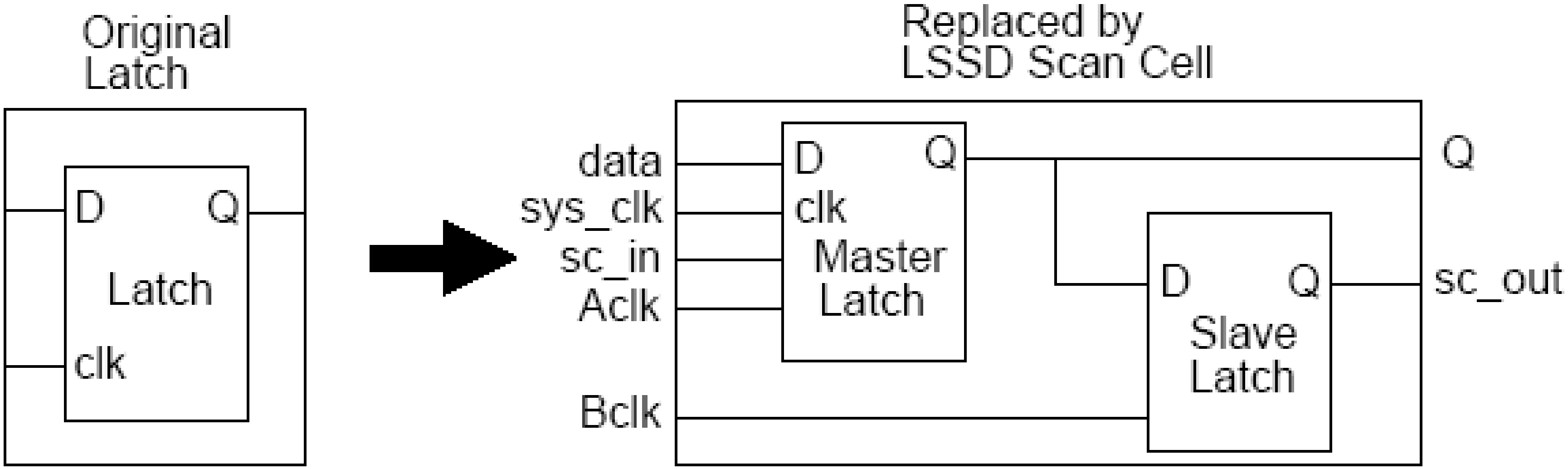


BICMOS8HP & ADK libraries: no “clocked_scan” components

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



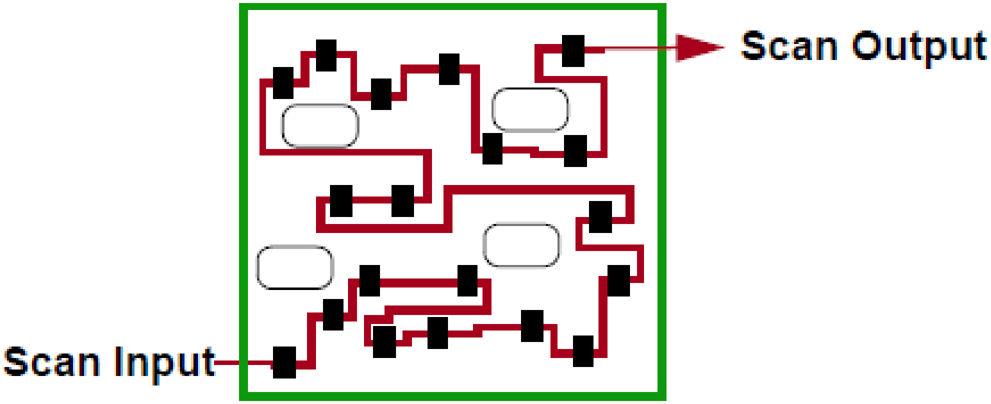
BICMOS8HP library: no “lssd” components

ADK library “lssd” components:

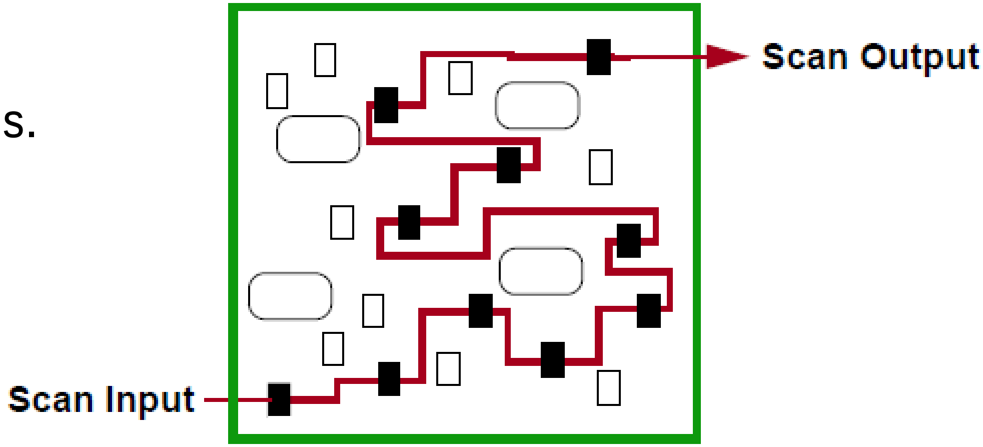
lssd_latch/latchsr/latchr/latchs/latchs_ni/latchsr_ni

Full vs. partial scan

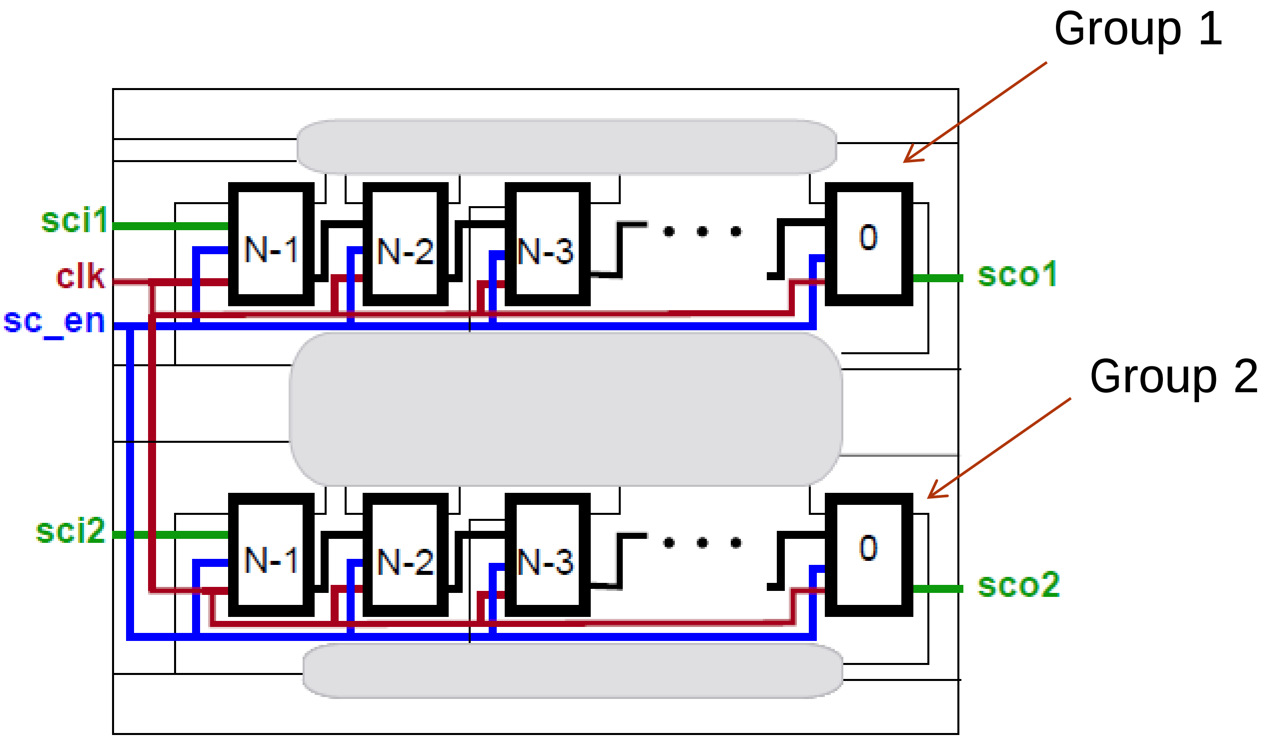
Full Scan:
All FFs in scan chains.



Partial Scan:
Some FFs not in scan chains.
Increase testability,
without affecting critical
timing/areas

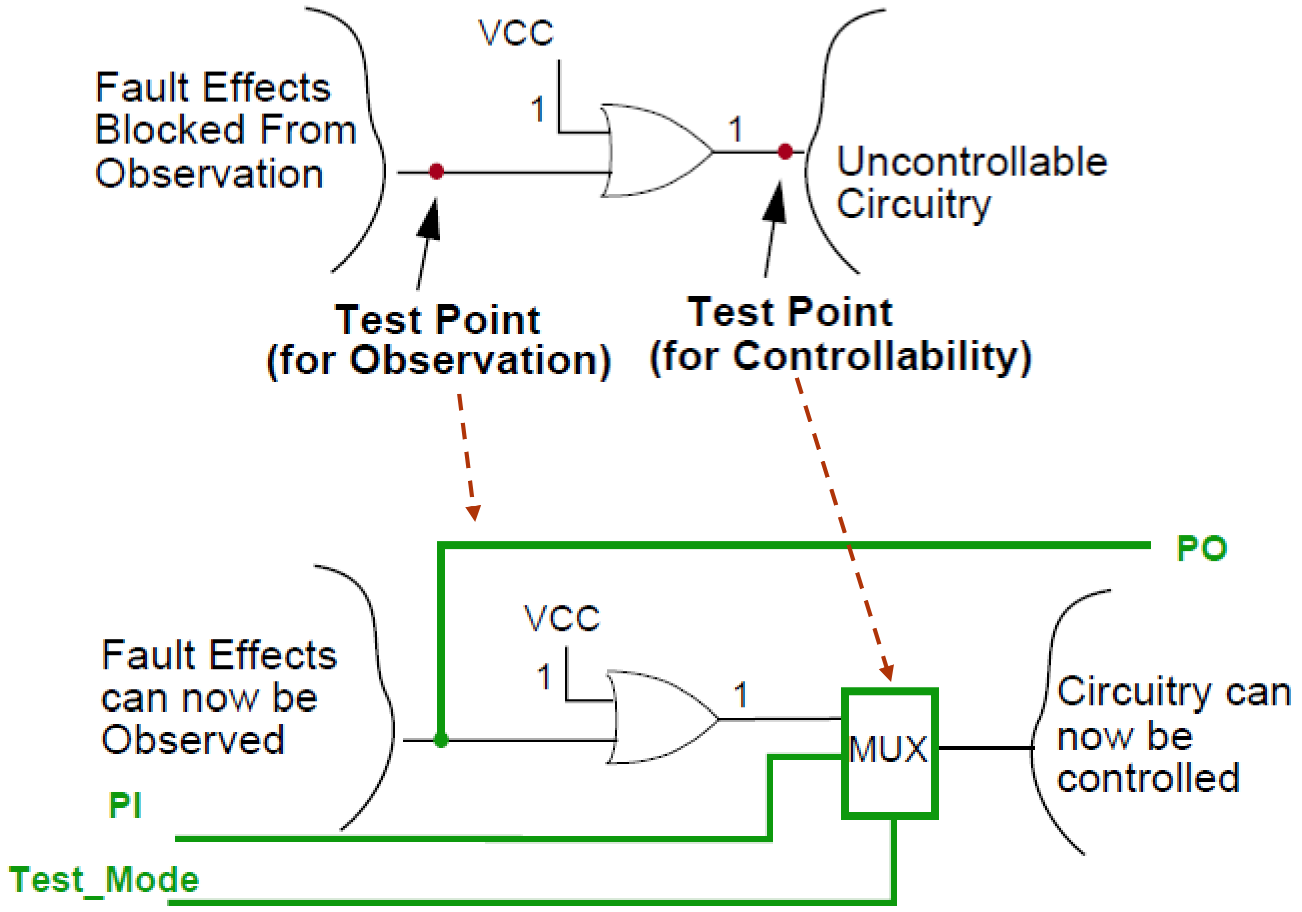


Scan chain groups

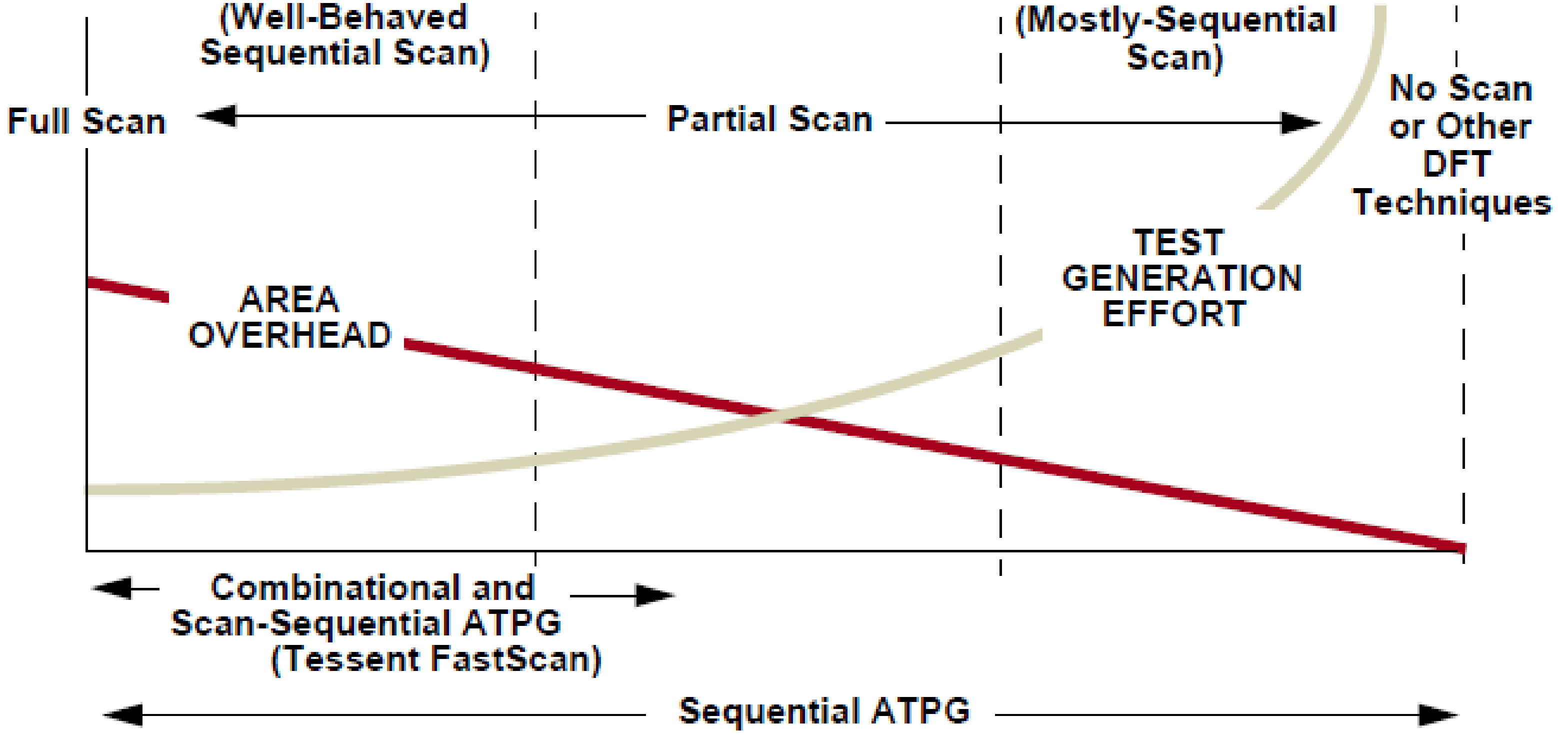


- Scan chains operate in parallel from separate scan inputs
- Reduces number of clock cycles to load/unload the chain
- Control from one procedure file
- Can use separate clocks or a common clock

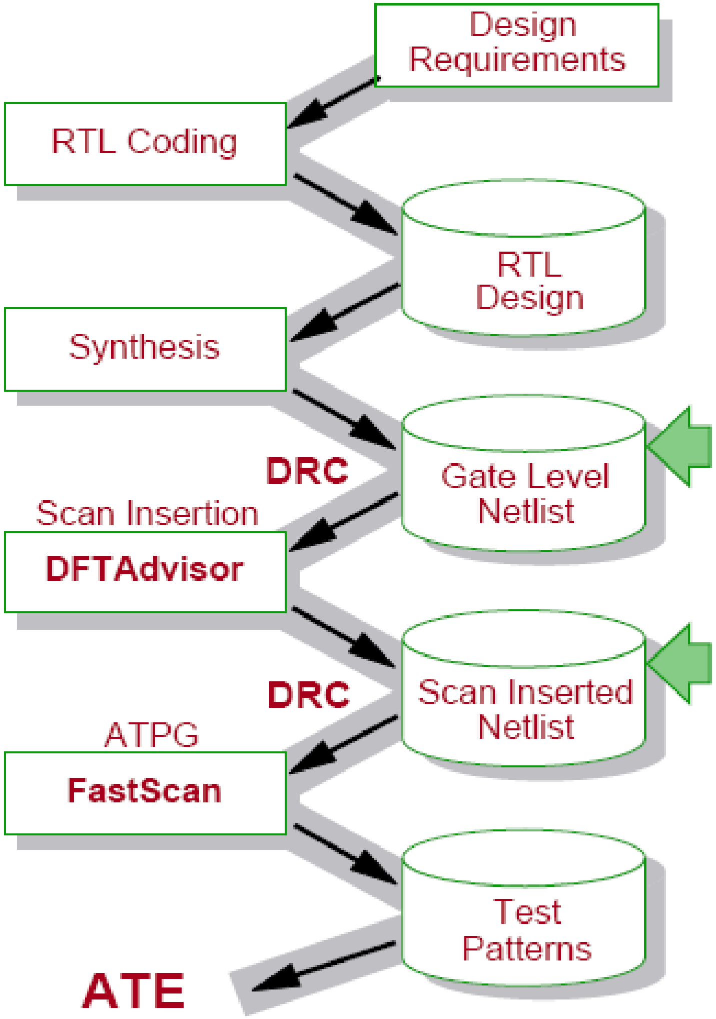
DFT test point insertion


Fault Effects Blocked From Observation
VCC
1
1
Uncontrollable Circuitry
Test Point (for Observation)
Test Point (for Controllability)
PO
VCC
1
1
Fault Effects can now be Observed
MUX
Circuitry can now be controlled
PI
Test_Mode

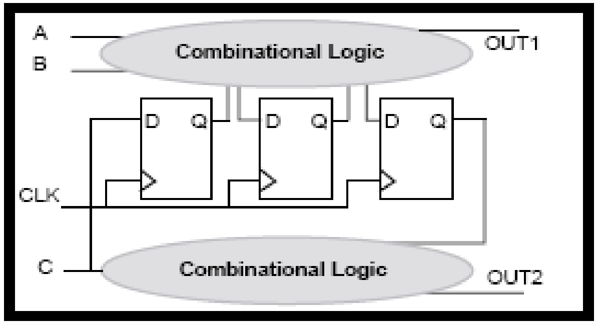
Choosing a DFT solution



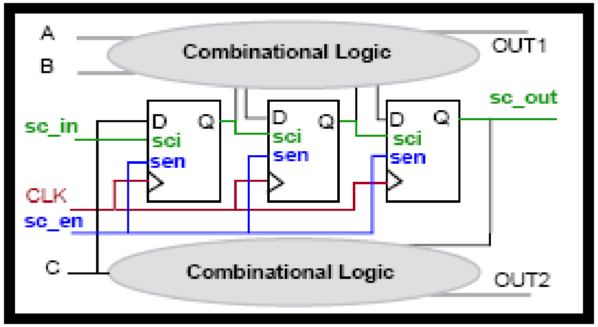
DFTadvisor/FastScan Design Flow



Before Scan

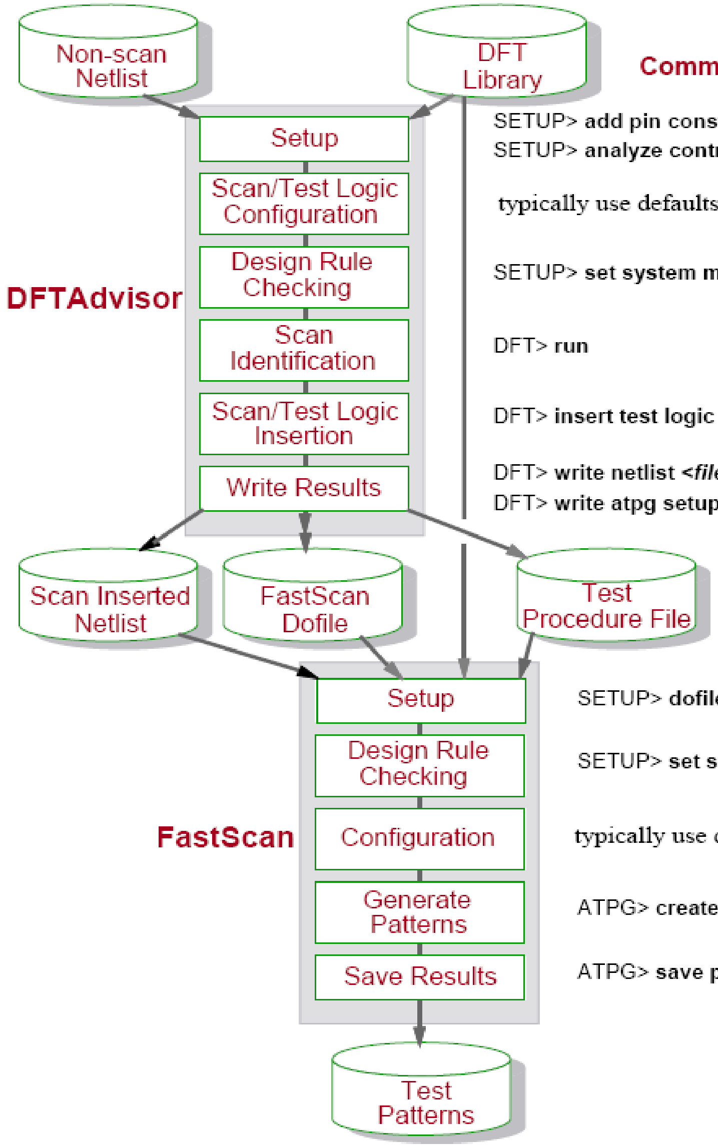


After Scan



Source: ATPG Manual

DFT test flow and commands



Commands:

```
SETUP> add pin constraints
SETUP> analyze control signals -auto_fix

typically use defaults

SETUP> set system mode dft

DFT> run

DFT> insert test logic -number 8

DFT> write netlist <file_name> -verilog
DFT> write atpg setup <file_name>
```

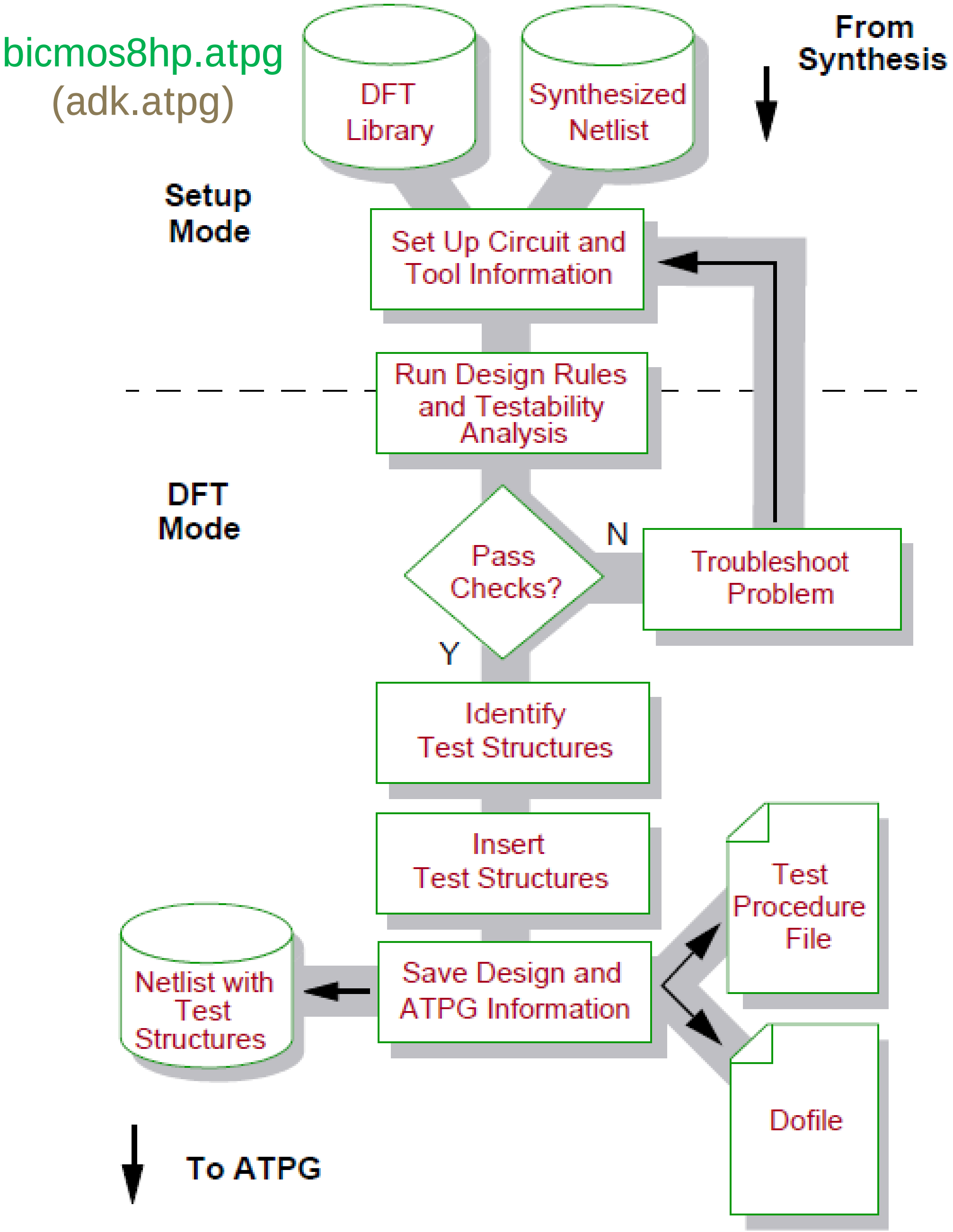
DFTAdvisor Commands (insert test logic)

```
SETUP> dofile <file_name>.dofile

SETUP> set system mode atpg

typically use defaults

ATPG> create patterns -auto

ATPG> save patterns  -verilog
```

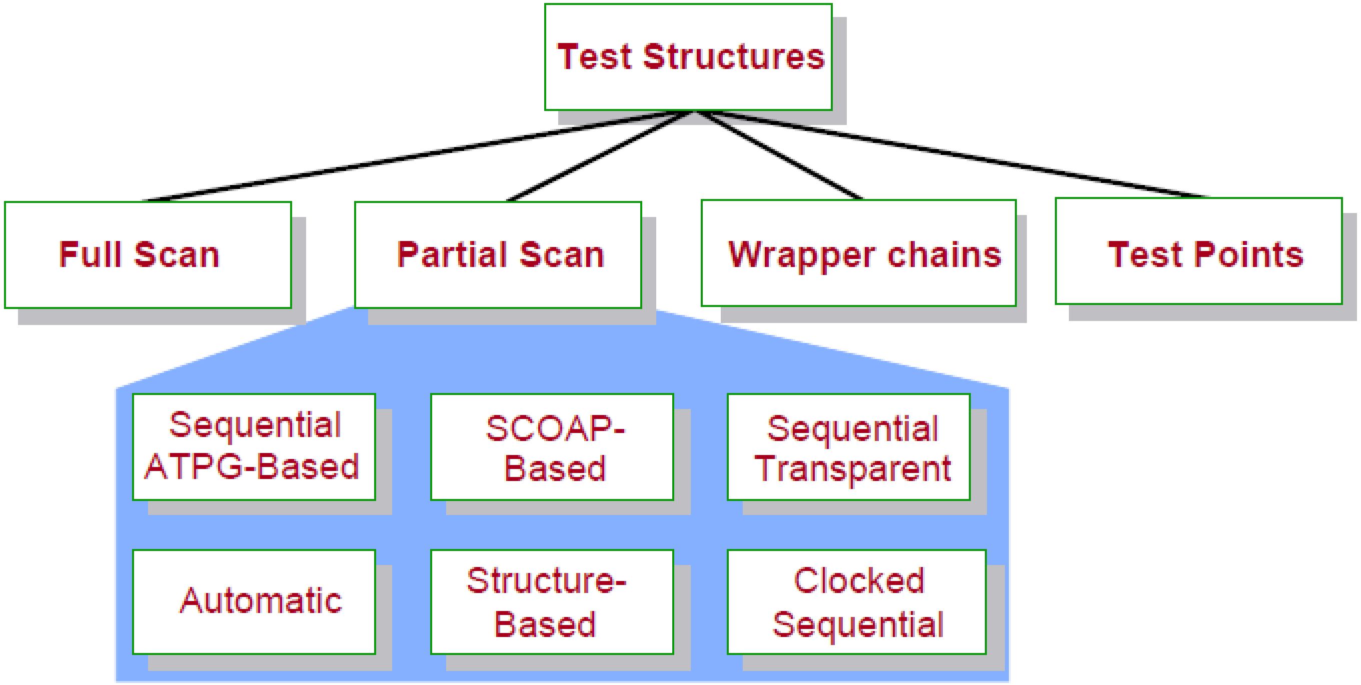
FastScan Commands (generate patterns)

Source: DFTadvisor Reference

Basic scan insertion flow



DFTAdvisor supported test structures



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

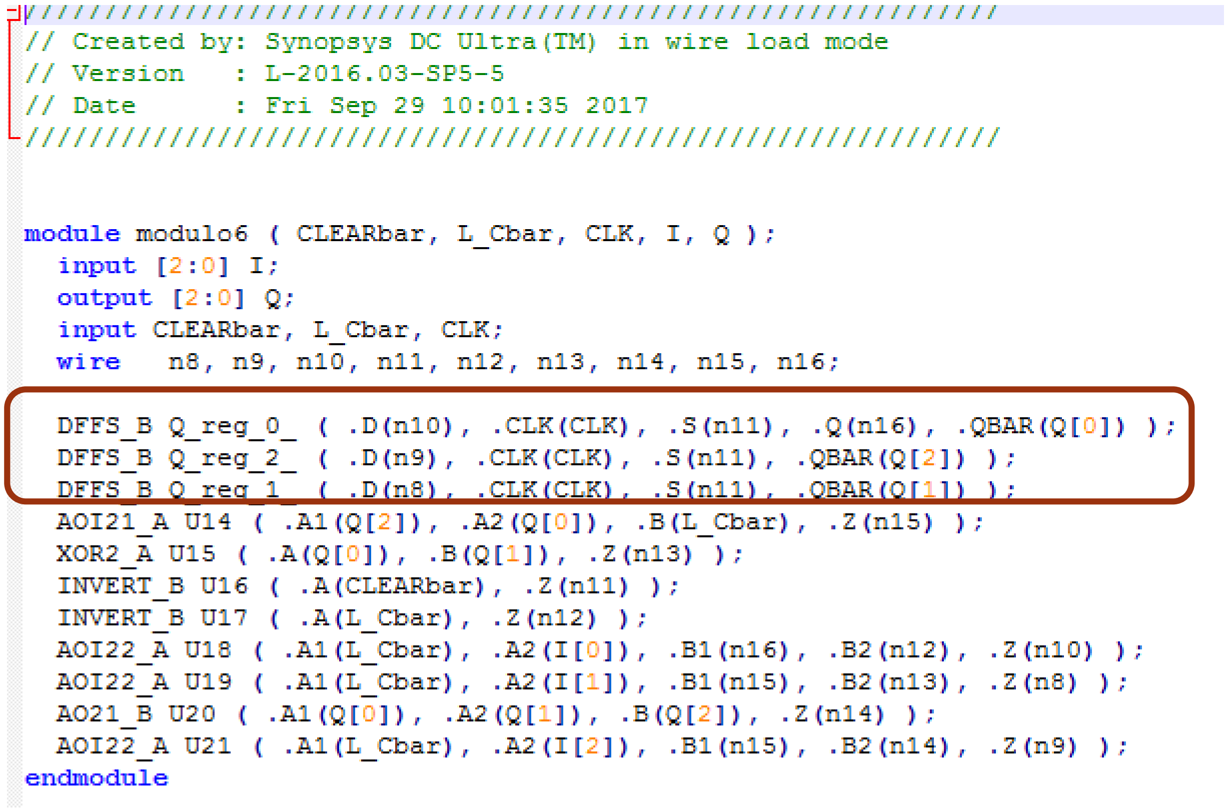
- Invoke:
  - dftadvisor modulo6_1.v –lib bicmos8hp.atpg
- Implement scan with defaults (full scan, mux-DFF elements):
  - set system mode setup (analyze the circuit)
  - analyze control signals (find clocks, resets, etc.)
  - add clocks 0 CLK (identify CLK off state)
  - add clocks 1 CLEARbar (likewise async set/reset)
  - set scan type mux_scan (use scan ffs with mux inputs)
  - set system mode dft (design for testability)
  - run (identify where to insert scan/test pts)
  - insert test logic –scan on (insert scan/tp's into netlist)
  - write netlist mod6_scan.v -replace (Verilog netlist of modified ckt)
  - write atpg setup mod6_scan -replace (dofile & test procedure for FastScan)

  Options:
  - insert test logic –scan on –number 3 (create 3 scan chains)
  - insert test logic –scan on –max_length 20 (no scan chain > 20 ffs)

DFT options

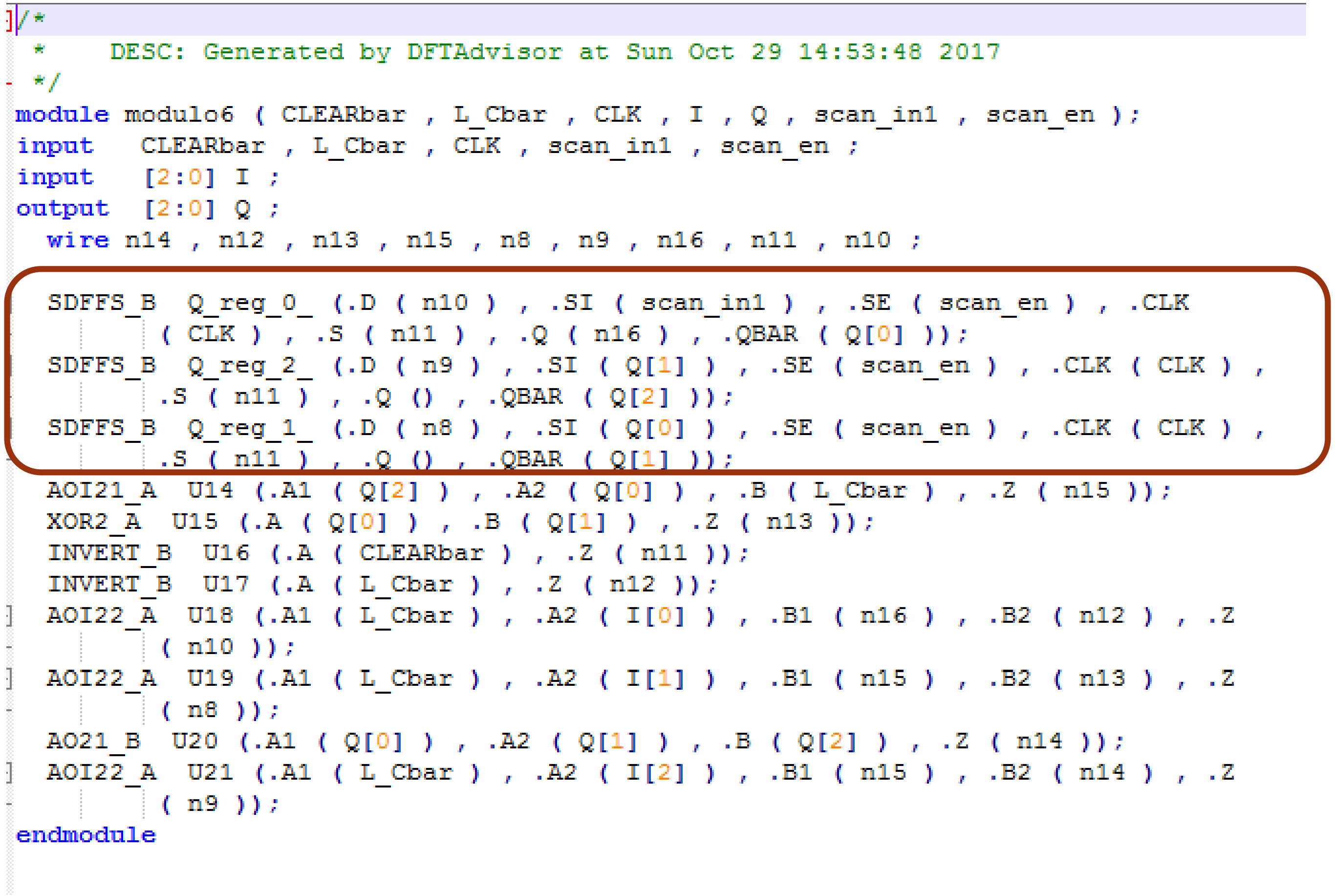
- ***set scan type mux_scan***
  - Others: lssd, clocked_scan
  - Find indicated scan flip flop type in the ATPG library
- ***setup scan identification* “type”, where “type” =**
  - full_scan (default)
  - sequential atpg –percent 50
  - clock_sequential [-depth integer]
  - etc.
- ***insert test logic***
  - -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)

Modulo-6 counter: Synthesized by Synopsys DC



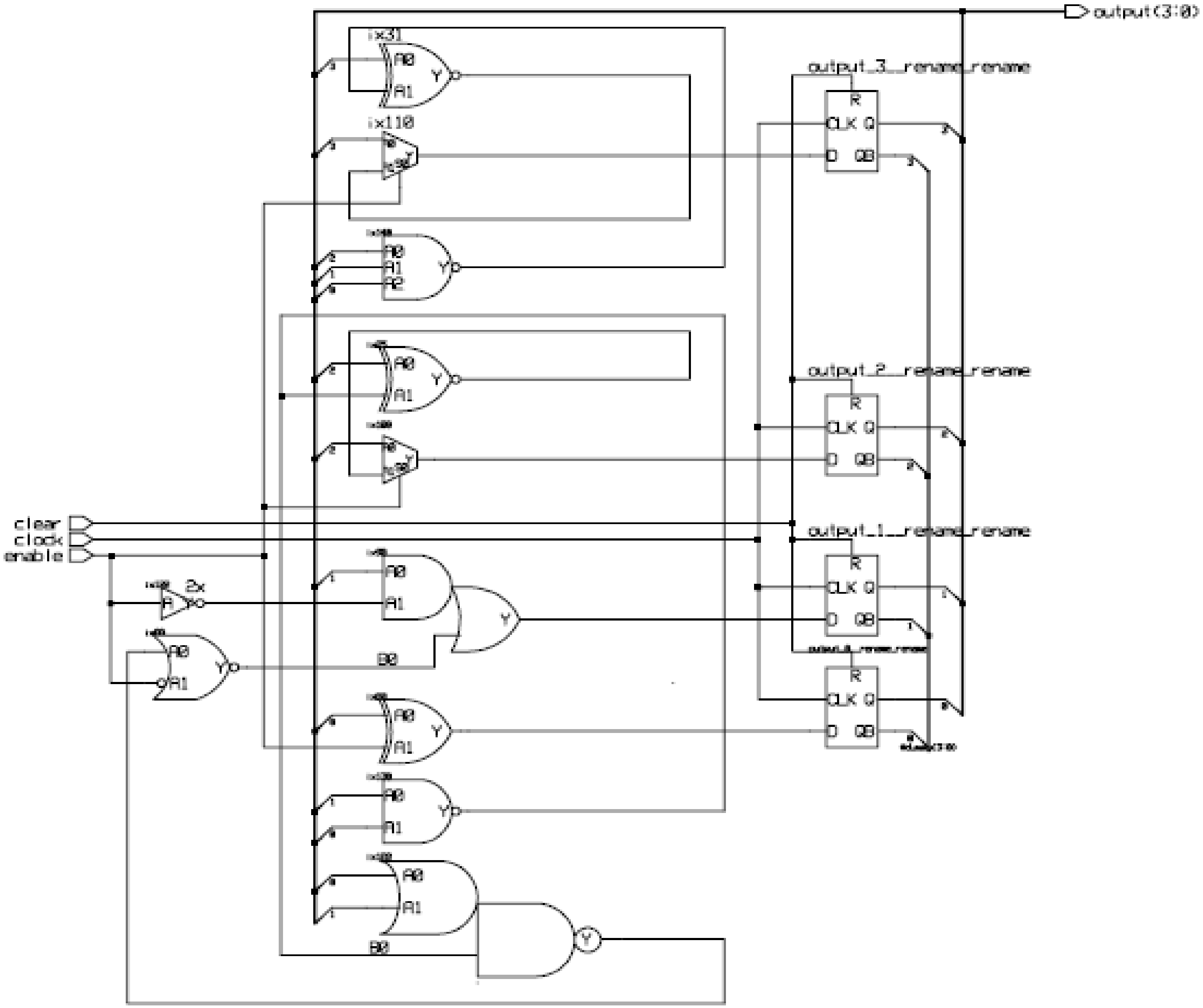
```
/////////////////////////////////////////////////////////////
// Created by: Synopsys DC Ultra(TM) in wire load mode
// Version   : L-2016.03-SP5-5
// Date      : Fri Sep 29 10:01:35 2017
/////////////////////////////////////////////////////////////


module modulo6 ( CLEARbar, L_Cbar, CLK, I, Q );
  input [2:0] I;
  output [2:0] Q;
  input CLEARbar, L_Cbar, CLK;
  wire   n8, n9, n10, n11, n12, n13, n14, n15, n16;

  DFFS_B Q_reg_0_ ( .D(n10), .CLK(CLK), .S(n11), .Q(n16), .QBAR(Q[0]) );
  DFFS_B Q_reg_2_ ( .D(n9), .CLK(CLK), .S(n11), .QBAR(Q[2]) );
  DFFS_B Q_reg_1_ ( .D(n8), .CLK(CLK), .S(n11), .QBAR(Q[1]) );
  AOI21_A U14 ( .A1(Q[2]), .A2(Q[0]), .B(L_Cbar), .Z(n15) );
  XOR2_A U15 ( .A(Q[0]), .B(Q[1]), .Z(n13) );
  INVERT_B U16 ( .A(CLEARbar), .Z(n11) );
  INVERT_B U17 ( .A(L_Cbar), .Z(n12) );
  AOI22_A U18 ( .A1(L_Cbar), .A2(I[0]), .B1(n16), .B2(n12), .Z(n10) );
  AOI22_A U19 ( .A1(L_Cbar), .A2(I[1]), .B1(n15), .B2(n13), .Z(n8) );
  AO21_B U20 ( .A1(Q[0]), .A2(Q[1]), .B(Q[2]), .Z(n14) );
  AOI22_A U21 ( .A1(L_Cbar), .A2(I[2]), .B1(n15), .B2(n14), .Z(n9) );
endmodule
```

Modulo-6 counter: Converted to full-scan (BICMOS8HP)



```
/*
 *    DESC: Generated by DFTAdvisor at Sun Oct 29 14:53:48 2017
 */
module modulo6 ( CLEARbar , L_Cbar , CLK , I , Q , scan_in1 , scan_en );
input   CLEARbar , L_Cbar , CLK , scan_in1 , scan_en ;
input   [2:0] I ;
output  [2:0] Q ;
  wire n14 , n12 , n13 , n15 , n8 , n9 , n16 , n11 , n10 ;

  SDFFS_B  Q_reg_0_ (.D ( n10 ) , .SI ( scan_in1 ) , .SE ( scan_en ) , .CLK
         ( CLK ) , .S ( n11 ) , .Q ( n16 ) , .QBAR ( Q[0] ));
  SDFFS_B  Q_reg_2_ (.D ( n9 ) , .SI ( Q[1] ) , .SE ( scan_en ) , .CLK ( CLK ) ,
         .S ( n11 ) , .Q () , .QBAR ( Q[2] ));
  SDFFS_B  Q_reg_1_ (.D ( n8 ) , .SI ( Q[0] ) , .SE ( scan_en ) , .CLK ( CLK ) ,
         .S ( n11 ) , .Q () , .QBAR ( Q[1] ));
  AOI21_A  U14 (.A1 ( Q[2] ) , .A2 ( Q[0] ) , .B ( L_Cbar ) , .Z ( n15 ));
  XOR2_A  U15 (.A ( Q[0] ) , .B ( Q[1] ) , .Z ( n13 ));
  INVERT_B  U16 (.A ( CLEARbar ) , .Z ( n11 ));
  INVERT_B  U17 (.A ( L_Cbar ) , .Z ( n12 ));
  AOI22_A  U18 (.A1 ( L_Cbar ) , .A2 ( I[0] ) , .B1 ( n16 ) , .B2 ( n12 ) , .Z
         ( n10 ));
  AOI22_A  U19 (.A1 ( L_Cbar ) , .A2 ( I[1] ) , .B1 ( n15 ) , .B2 ( n13 ) , .Z
         ( n8 ));
  AO21_B  U20 (.A1 ( Q[0] ) , .A2 ( Q[1] ) , .B ( Q[2] ) , .Z ( n14 ));
  AOI22_A  U21 (.A1 ( L_Cbar ) , .A2 ( I[2] ) , .B1 ( n15 ) , .B2 ( n14 ) , .Z
         ( n9 ));
endmodule
```

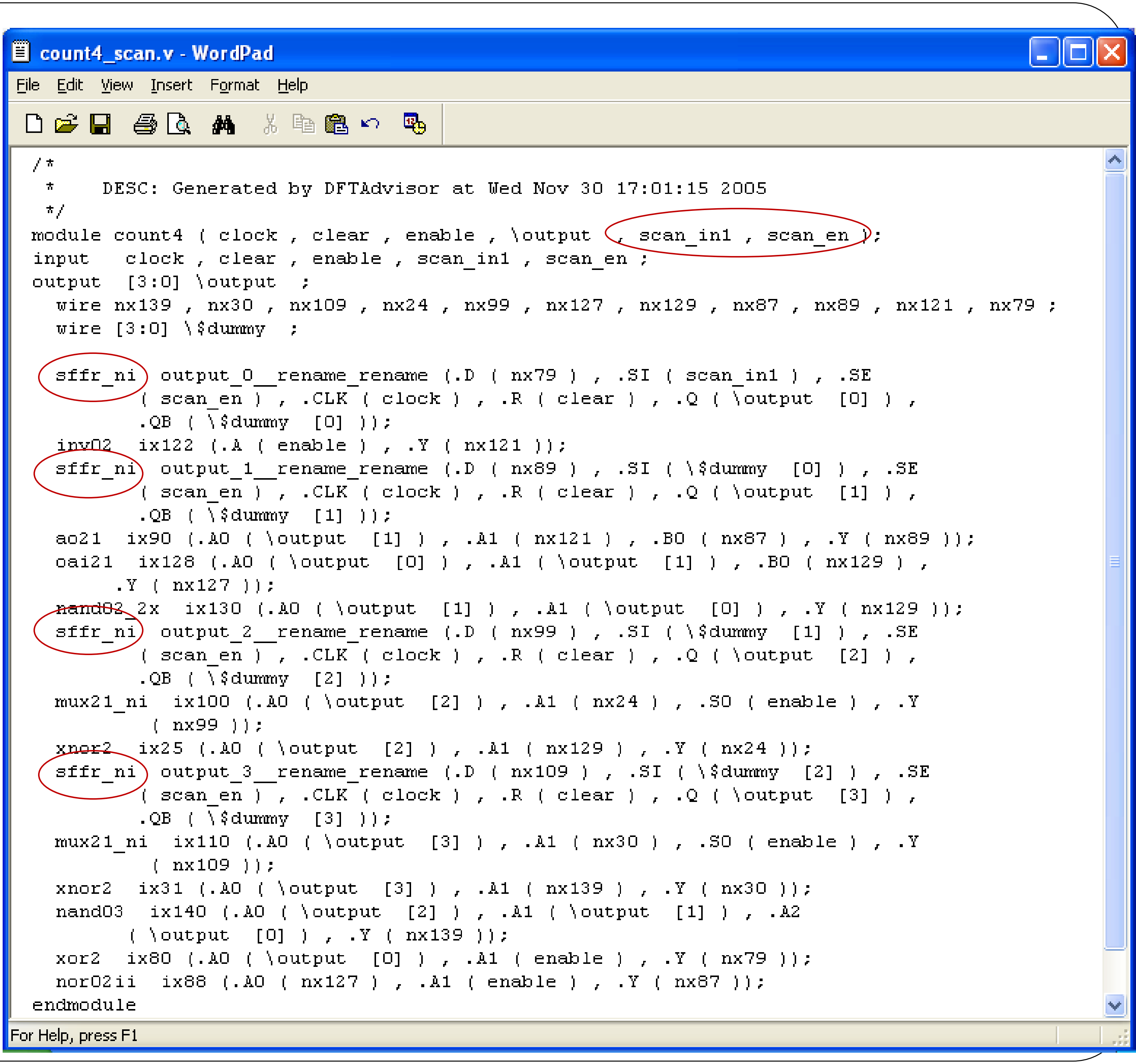
count4 – without scan design (TSMC 180nm)



Binary counter (4-bit)

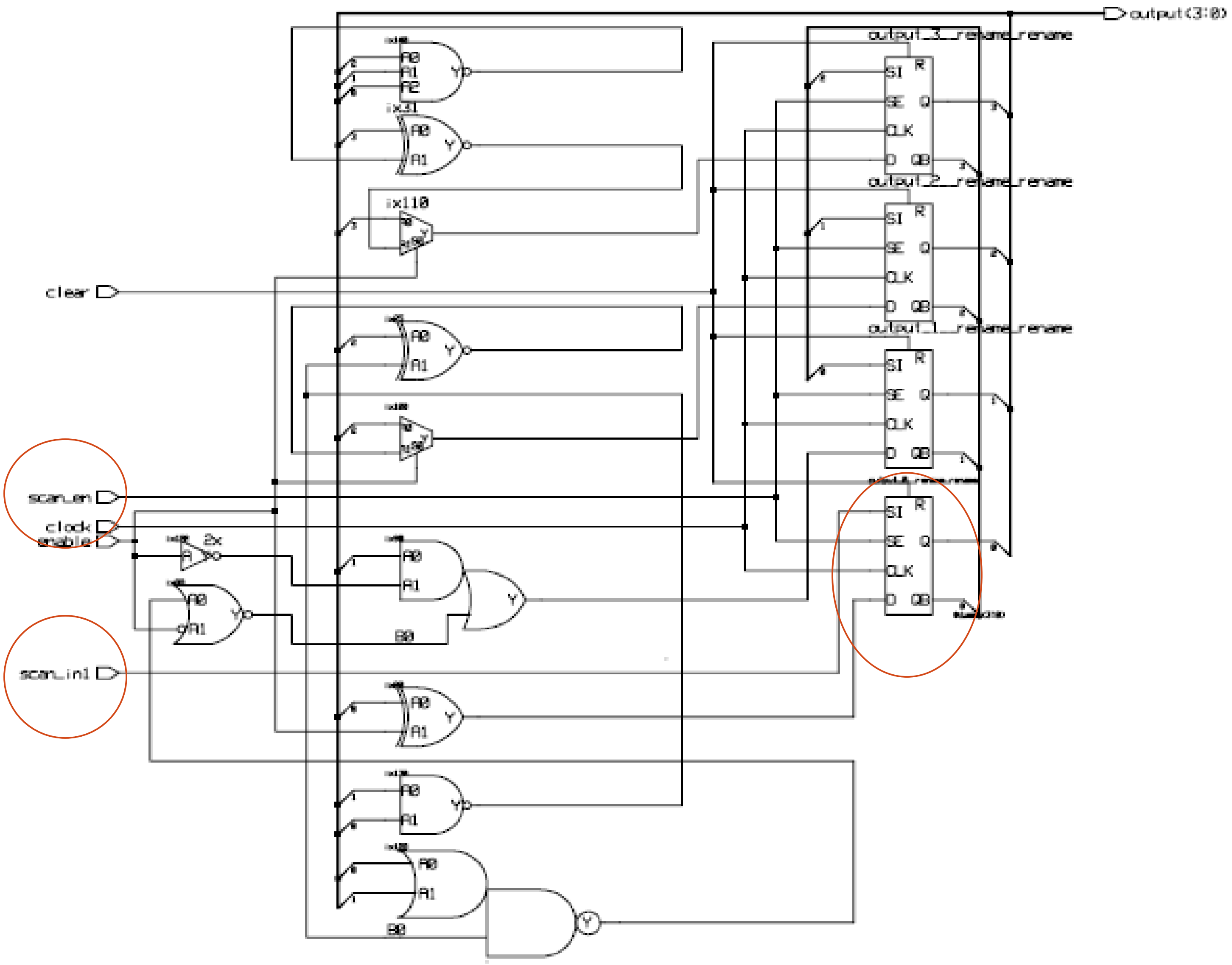
Synthesized by Leonardo

DFTAdvisor Changed to Scan Design



```
/*
 *    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  ;

  sffr_ni  output_0__rename_rename (.D ( nx79 ) , .SI ( scan_in1 ) , .SE
           ( scan_en ) , .CLK ( clock ) , .R ( clear ) , .Q ( \output  [0] ) ,
           .QB ( \$dummy  [0] ));
  inv02  ix122 (.A ( enable ) , .Y ( nx121 ));
  sffr_ni  output_1__rename_rename (.D ( nx89 ) , .SI ( \$dummy  [0] ) , .SE
           ( scan_en ) , .CLK ( clock ) , .R ( clear ) , .Q ( \output  [1] ) ,
           .QB ( \$dummy  [1] ));
  ao21  ix90 (.A0 ( \output  [1] ) , .A1 ( nx121 ) , .B0 ( nx87 ) , .Y ( nx89 ));
  oai21  ix128 (.A0 ( \output  [0] ) , .A1 ( \output  [1] ) , .B0 ( nx129 ) ,
       .Y ( nx127 ));
  nand02_2x  ix130 (.A0 ( \output  [1] ) , .A1 ( \output  [0] ) , .Y ( nx129 ));
  sffr_ni  output_2__rename_rename (.D ( nx99 ) , .SI ( \$dummy  [1] ) , .SE
           ( scan_en ) , .CLK ( clock ) , .R ( clear ) , .Q ( \output  [2] ) ,
           .QB ( \$dummy  [2] ));
  mux21_ni  ix100 (.A0 ( \output  [2] ) , .A1 ( nx24 ) , .S0 ( enable ) , .Y
            ( nx99 ));
  xnor2  ix25 (.A0 ( \output  [2] ) , .A1 ( nx129 ) , .Y ( nx24 ));
  sffr_ni  output_3__rename_rename (.D ( nx109 ) , .SI ( \$dummy  [2] ) , .SE
           ( scan_en ) , .CLK ( clock ) , .R ( clear ) , .Q ( \output  [3] ) ,
           .QB ( \$dummy  [3] ));
  mux21_ni  ix110 (.A0 ( \output  [3] ) , .A1 ( nx30 ) , .S0 ( enable ) , .Y
            ( nx109 ));
  xnor2  ix31 (.A0 ( \output  [3] ) , .A1 ( nx139 ) , .Y ( nx30 ));
  nand03  ix140 (.A0 ( \output  [2] ) , .A1 ( \output  [1] ) , .A2
         ( \output  [0] ) , .Y ( nx139 ));
  xor2  ix80 (.A0 ( \output  [0] ) , .A1 ( enable ) , .Y ( nx79 ));
  nor02ii  ix88 (.A0 ( 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 (generate the test patterns)
  - save patterns count4_patterns.v –verilog (write patterns & 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 shift "grp1_unload_shift" =
         measure_sco "chain1" 10;
         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 unload "grp1_unload" =
            force "/clear" 0 0;
            force "/clock" 0 0;
            force "/scan_en" 1 0;
            apply "grp1_unload_shift" 4 40;
         end;
      end;
```

(each shift)

(each shift)

shifts

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;
```

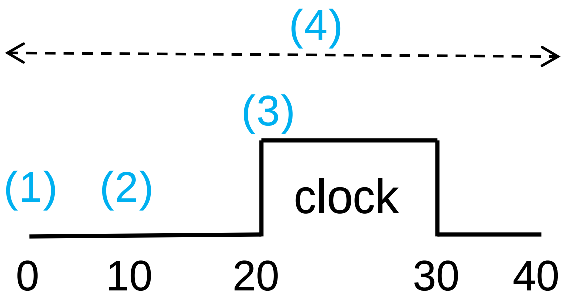
Test file: sample test pattern

```
// one of 14 patterns for the counter circuit
pattern = 0;                                (pattern #)
   apply "grp1_load" 0 =                    (load scan chain)
      chain "chain1" = "1000";              (scan-in pattern)
   end;
   force   "PI" "00110" 1;                  (apply PI pattern)
   measure "PO" "0010" 2;                   (expected POs)
   pulse "/clock" 3;                        (one normal op. cycle)
   apply "grp1_unload" 4 =                  (read scan chain)
      chain "chain1" = "0110";              (expected pattern)
   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;
```

Timing of op's within each cycle



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

Each shift cycle

```
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;
```

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
```