Power Compiler and DFT Compiler
Making them work together -- lessons learned

Henry George Berkley

ASIC WIZARD GROUP
TURNING PRODUCT IDEAS INTO SILICON

hgb@asicwizard.com

ABSTRACT

Power Compiler and DFT Compiler are two powerful tools but when used together they can fight each other if not used properly. This paper describes a consulting assignment to troubleshoot a client's attempt to save power with gated-clocks inserted by Power Compiler and scan insertion with Test-Ready Compile and DFT Compiler scan synthesis. It explains how to make DFT Compiler work with Power Compiler, what is documented by Synopsys and where it is documented. It shows troubleshooting techniques that go under-the-hood to discover how Test-Ready Compile and DFT Compiler work and what they need that is not documented by Synopsys. Lessons learned include understanding some bizaar legacy scripts, how to troubleshoot them, and how they work with a robustness challenged legacy design. The legacy design needed a change and a Synopsys bug caused `hookup_testports` and `set_scan_signal -hookup` to fight requiring a work-around script.

Table of Contents

1.0 Introduction....................4
2.0 Consulting Assignment Goals....................4
3.0 Power Compiler RTL gated clock insertion....................5
3.1 Read Chapter 7....................5
3.2 What is RTL gated clock insertion?....................6
4.0 Module level issues....................8
4.1 Original RTL gated clock insertion script....................8
4.2 DFT Compiler balks....................8
4.3 A testable gated clock circuit....................9
4.4 Using the `set_clock_gating_style` command....................9
4.5 Using the `hookup_testports` command....................10
4.6 Adding testability and `hookup_testports` helps....................11
4.7 DFT Compiler has trouble with one module....................11
4.8 Creating an RTL test case....................12
4.9 Gated clock insertion for negative edge register banks....................13
4.10 Lesson learned from the test case....................14
4.11 Change done to the design....................15
4.12 Capture violations....................15
5.0 Top level issues....................16
5.1 DFT Compiler balks again....................16
5.2 Legacy `set_dont_touch` everything script....................16
5.3 Diagnosing what it does....................17
5.4 Using the `group` command to avoid the problem....................18
5.5 Demonstrating clock polarity issues....................18
5.6 Custom pad cells lacks functionality....................19
5.7 Fixing the library cell....................20
5.8 The `hookup_testports` command fights `set_scan_signal`...20
5.9 The work-around script....................21
6.0 Test-Ready Compile....................22
6.1 What is Test-Ready Compile?....................22
6.2 How it works under-the-hood....................23
6.3 Avoid Verilog netlists....................23
7.0 Recommendations and Conclusions....................24
7.1 Synopsys Recommendations....................24
7.2 Management Recommendations....................24
7.3 Technical Conclusions....................24
Appendix A. Help Pages....................25
A.1 LIBG-16 Help Page....................25
A.2 TEST-169 Help Page....................26
A.3 TEST-451 Help Page....................27
A.4 LINT-0 Help Page....................27

List of Figures

Figure 1 Synchronous Load-Enable Register Bank 6
Figure 2 Gated Clock Register Bank 6
Figure 3 Control Point in Gated Clock Circuitry 9
Figure 4 Module with new test_se port 10
Figure 5 Negative Edge Triggered Register Banks 13
Figure 6 Tester Cycle Clock Waveforms 14
Figure 7 Test clocking circuitry 15
Figure 8 What set_dont_touch everything does 17
Figure 9 Using set_scan_signal before hookup_testports 20
Figure 10 Using hookup_testports before set_scan_signal 21
Figure 11 Modules with and without compile -scan 22
Figure 12 Modules After Scan Synthesis 22

1.0 Introduction

CAE tools have become so complex that they require dedicated specialists to operate them. This paper studies the use of two CAE tools operated by two specialists and the difficulties of getting them to work together. Included are not just the answers on how to get Power Compiler to work with DFT Compiler, but also the methods used to find the answers.

2.0 Consulting Assignment Goals

- Re-use legacy design to reduce design schedule
- Use Power Compiler to insert gated clocks to save power
- Use DFT Compiler to insert test circuitry with scan synthesis

This is a classic example of what happens in our industry. The team wants to modernize their design flow but they are required to work with a legacy design. Change the flow but don't change the design.

The task of RTL gated clock insertion was given to the in-house synthesis specialist. The task of scan insertion was given to the in-house DFT[1] specialist. Both specialists were restricted from changing the legacy design except by using their CAE tools, Power Compiler and DFT Compiler respectively. When the synthesis specialist added RTL gated clock insertion to the synthesis script, the scan insertion scripts written by the DFT specialist stopped working. The consulting assignment goal from the ASIC Design Manager was to get things working to stop further schedule slippage.

Power optimization, Test-Ready Compile, DFT, and automatic test pattern generation (ATPG) were new additions to the ASIC development flow. The new tools included Power Compiler for power optimization, Prime Power for power analysis, DFT Compiler for scan synthesis, and TetraMAX for ATPG. The consulting assignment goal from the Director of ASIC Development was to help the team learn the new tools.

In summary, the assignment goals were to change the flow but don't change the design and teach the new tools but don't impact the schedule.

1. DFT stands for Design For Test. Ironically even though the the DFT specialist has the word design in his acronym, his role was considered to be "back-end" and not part of the design effort.

3.0 Power Compiler RTL gated clock insertion

3.1 Read Chapter 7[1]

The key to knowing how to make Power Compiler RTL gated clock insertion work with DFT Compiler is to read chapter 7 of the Power Compiler User Guide. It is beyond the scope of this paper to explain all of the features of RTL gated clock insertion but to highlight how to make the two tools work together and show things that are not explained in the manual. DFT specialists who are getting designs created by Power Compiler need to read chapter seven especially the section titled “Improving Testability”. Synthesis specialists wanting to do RTL gated clock insertion and intend handing the design to a DFT specialist for scan insertion need to avoid skipping the section on “Improving Testability”.

I have a lot to say about reading manuals but it would probably be censored by the technical committee. To fulfill this paper’s promise to show troubleshooting techniques, I will say RTFM - Read The Friendly Manual. Alot of the answers are in there.

---

1. The title of this section was originally called “Read Chapter 9” because Version 2000.05 of the Power Compiler Reference Manual described RTL gated clock insertion in chapter 9. Since then the Power Compiler manual changed its name to Power Compiler User Guide and moved the chapter on RTL clock gating to chapter 7.

3.2 What is RTL gated clock insertion?

Figure 1 Synchronous Load-Enable Register Bank

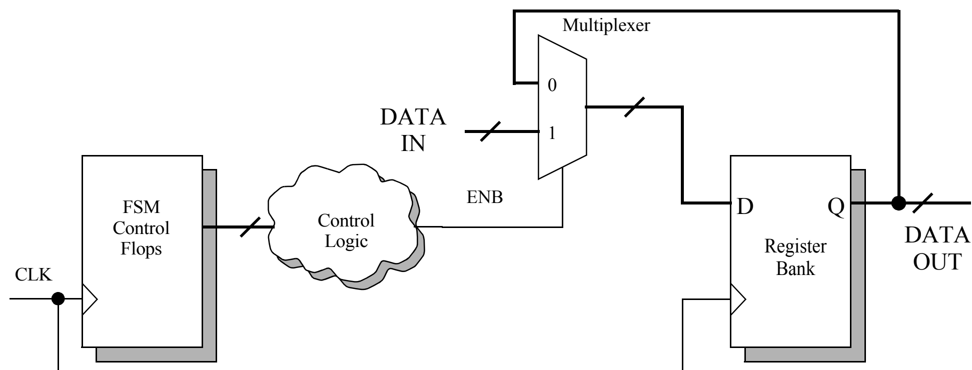


Figure 2 Gated Clock Register Bank

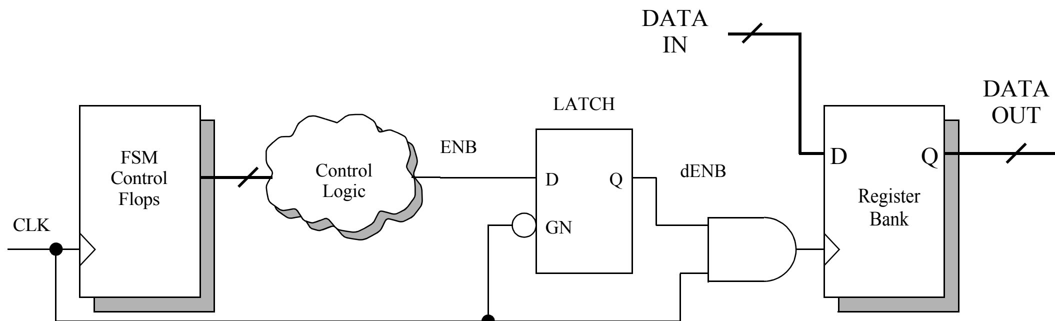


Figure 1 shows a synchronously loaded register bank. Figure 2 shows a register bank that is loaded with a gated clock. What Power Compiler RTL gated clock insertion does is find register banks that are synchronously loaded and converts them to register banks that are loaded with a gated clock. It does this with the GTECH netlist before it is mapped to flops and gates.

The Power Compiler manual describes both Figure 1[1] and Figure 2 [2]and includes timing diagrams for Figure 2 so I am not going to include that here. RTFM. When you have the manual open to those figures, take a red pen and strike FLIP-FLOP and replace it with FSM CONTROL FLOPS. This makes the need for the LATCH clearer. The cloud of logic has multiple inputs which emerge as a single signal to gate the clock of the REGISTER BANK. On the rising edge of the clock, multiple flops in the FSM may change. There is a possibility that the signal emerging from the cloud of logic may have glitchs. The LATCH prevents those glitches from getting to the clock gate when the clock signal is high. When the clock signal is low, the clock gate is closed and glitches are masked. The LATCH also delays changes of the dENB signal until the falling edge of the clock. This prevents the clock signal coming out of the AND gate from becoming a runt pulse or disappearing.

The Power Compiler manual says that gated clock insertion reduces power consumption by reducing the switching activity of the REGISTER BANK. That is incorrect. The switching activity of the REGISTER BANK is the same before and after gated clock insertion. Power consumption is reduced by removing the multiplexer (MUX) circuitry and by reducing the switching activity on the clock for the REGISTER BANK.

A historical note on gated clocks. In the days when we placed gates by hand we used gated clocks to load register banks because this saved circuitry. When we adopted placement tools that were timing unaware, gated clocks increased the risk of introducing hold time violations. When time delay is added to the clock path and delay is removed from the data path, a flop gets closer to violating its hold time. Synchrononously loaded registers also were easier for static timing analysis tools and scan synthesis tools. Now that we have timing-aware placement tools and timing-aware clock tree synthesis tools it is possible to go back to a gated clock methodology. The problems with scan synthesis remain and this paper shows how to deal with that.

1. Figure 1 is Figure 9-1 in the Version 2000.05 Power Compiler Reference Manual or Figure 7-1 in the Version 2003.12 Power Compiler User Guide
2. Figure 2 is Figure 9-2 in the Version 2000.05 Power Compiler Reference Manual or Figure 7-2 in the Version 2003.12 Power Compiler User Guide

4.0 Module level issues

4.1 Original RTL gated clock insertion script

Example of original module-level script

dc_shell> **analyze -f verilog A.v**
dc_shell> ~~**set_clock_gating_style**~~
dc_shell> **elaborate -gate_clock**[1] **A**
dc_shell> ~~**hookup_testports**~~
dc_shell> **create_clock -period 10 find(port CLK)**
dc_shell> **set_clock_skew ideal CLK**
dc_shell> **propogate_constraints -gate_clock**
dc_shell> **compile -scan**

This was an example of a module-level script from the in-house synthesis specialist. The commands that have strikethroughs were not used. Without the `set_clock_gating_style` command Power Compiler implements the default clock gating style. The `hookup_testports` command was also omitted.

4.2 DFT Compiler balks

How did DFT Compiler react to a module using the default clock gating style?

```
Warning: Normal mode clock pin %s of %s (%s) is
uncontrollable. (TEST-169)
```

When DFT Compiler is unhappy with a design, it drops the flops that it doesn't like from the scan chain and issues a message that fails to explain the solution but hints at the problem. In this case the DFT specialist's scripts which had worked in the past stopped working.

Look back at Figure 2. In scan mode, data shifting through the FSM CONTROL FLOPS cause the clock of the REGISTER BANK to drop out. Data can not be shifted through the REGISTER BANK. Controllability and observability of the REGISTER BANK is lost. DFT Compiler correctly decides to drop the REGISTER BANK from its scan chain.

The problem is caused by the in-house synthesis specialist's script but only appears when the design gets to the in-house DFT specialist. The solution to the problem is found in the Power Compiler manual which was not part of the DFT specialist's training.

---

1. The `elaborate -gate_clock` command was the original was of doing RTL gated clock insertion. Synopsys introduced the `insert_clock_gating` command in Version 2000.11 to support designs elaborated by Module Compiler. Synopsys recommends new scripts use `insert_clock_gating`.

4.3 A testable gated clock circuit

Figure 3 Control Point in Gated Clock Circuitry

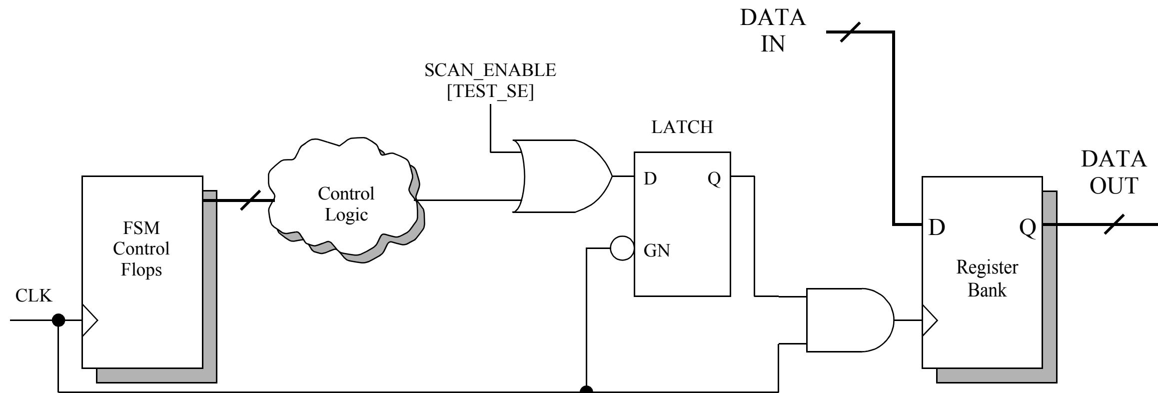


Figure 3[1] shows how to make the REGISTER BANK testable for DFT Compiler. In scan mode the SCAN_ENABLE is ORed with the signal from the FSM CONTROL FLOPS forcing the clock to the REGISTER BANK to be constantly active during scan. The Power Compiler manual has a timing diagram and a more detailed explaination.

4.4 Using the `set_clock_gating_style` command

```
dc_shell> set_clock_gating_style \
            -control_point before \
            -control_signal scan_enable
```

The default value for the `-control_point` switch is `none`. To make the clock gating work with DFT Compiler the `-control_point` switch should be set to either `before` or `after`. Our choice was the `before` option because the Power Compiler manual had a cautionary note about the transition point for the `scan_enable` signal when using the `after` option. Use of the `after` option needs further investigation.

The default value for the `-control_signal` switch is `scan_enable`. The other option is `test_mode`. RTFM[2]. (Actually the manual is not that friendly but start there anyway.) Our choice was the `scan_enable` option because this creates a DFT circuit that allows scan testing of the functional circuit. Scan test vectors that work before gated clock insertion also work for the circuit after gated clock insertion.

1. Figure 3 is Figure 9-7 of the Version 2000.05 Power Compiler Reference Manual or Figure 7-7 of the Version 2003.12 Power Compiler User Guide.
2. Read “Scan Enable Versus Test Mode” in chapter 9 of the Version 2000.05 Power Compiler Reference Manual or chapter 7 of the Version 2003.12 Power Compiler User Guide.

4.5 Using the `hookup_testports` command

```
dc_shell> hookup_testports
```

Elaborating a module with a clock gating style which adds test control points for DFT Compiler creates unconnected pins in the module. In this case it creates unconnected `scan_enable` nets. See Figure 3. These unconnected nets need to be connected to a module port before `compile` or `compile -scan` or there will be problems[1]. The `hookup_testports` command creates a `test_se` port in the module and wires that port to all the unconnected `scan_enable` nets. Power Compiler then uses auto-magic[2]to signal DFT compiler about the usage of the `test_se` port.

Figure 4 Module with new `test_se` port

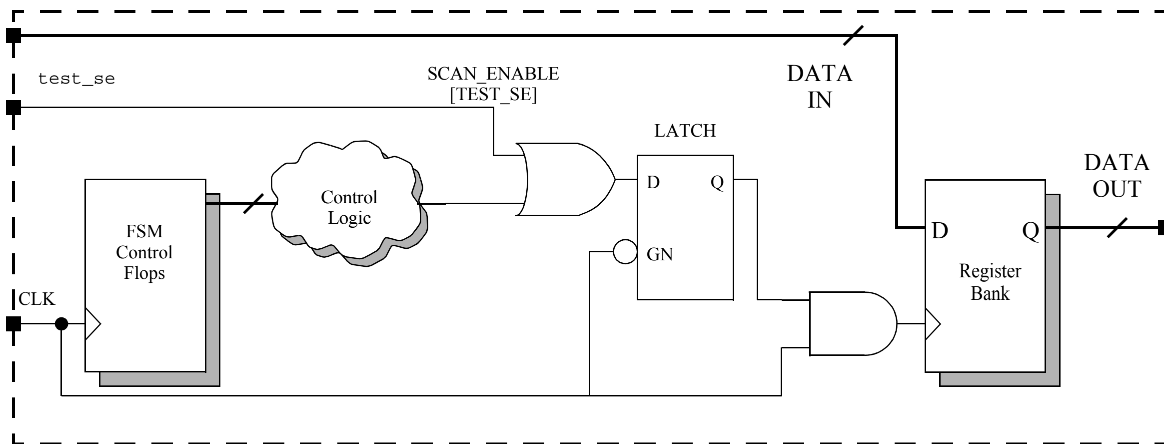


Figure 4 shows the module after using `hookup_testports`. A new port `test_se` has been added and connected to all the control points added by Power Compiler.

At the time of this consulting assignment, the `hookup_testports` command was only described in the chapter 9 of the Power Compiler Reference Manual (now chapter 7 of the Power Compiler User Guide). There was no `man` page for this command and it was not listed in the Synthesis Quick Reference. The Power Compiler manual describes how to use `hookup_testports` with a top-down compile methodology. This client used a bottom-up compile methodology. This paper describes the use of `hookup_testports` with that methodology.

---

1. Those problems are discussed in “Legacy set_dont_touch everything script” on page 16
2. Auto-magic is discussed in “How it works under-the-hood” on page 23

4.6 Adding testability and `hookup_testports` helps

The methodology used by this client was to have each module designer synthesize his own module under the guidance of the in-house synthesis specialist. Each designer's synthesis script was modified to include gated clock insertion. The compiled DB files from each designer were given to the DFT specialist for scan synthesis. Top level integration will be discussed in "Top level issues." on page 16.

As stated earlier, modules synthesized using the default clock gating style caused the DFT specialist's scripts to stop working. Each designer was asked to change their clock gating style to `-control_point before` and add the `hookup_testports` command before compiling their module. The compiled DB files were given to the DFT specialist. The DFT scripts were modified to account for the addition of a `test_se` port to each module. This solved the problem for the most part. DFT Compiler started to work with Power Compiler RTL gated clock insertion.

4.7 DFT Compiler has trouble with one module

```
Warning: Normal mode clock pin cn of cell rd_data_reg_7_
(ffs240b0) is
uncontrollable. (TEST-169)
```
[1]

After getting all of the module-level synthesis scripts modified to insert gated clocks compatible with DFT Compiler, one of the modules still had a problem. DFT Compiler declared the negative edge flops uncontrollable. The positive edge flops were added to the scan chain but DFT Compiler refused to add the negative edge flops to the chain. The DFT Compiler script worked before gated clock insertion but not after gated clock insertion. Changing the clock gating style from the default to `-control_point before` fixed the controllability problem with the positive edge flops but not with the negative edge flops.

This is where the Power Compiler Reference Manual and the Scan Synthesis Reference Manual stopped giving me answers. I emailed a DFT specialist that I know. He wrote back saying he didn't know the solution. I telephoned the Power Products manager. He told me he didn't know. It should work. That advice turned out to be very helpful. By quoting Synopsys as saying it should work, I bought more time to look for the solution.

It was time to create a test case.

1. This message comes from SolvNet article 001017. I solved this problem differently than any of the solutions suggested in that article.

4.8 Creating an RTL test case

```
module test_case(clk,d,qa,qb);
    input        clk;
    input  [7:0] d;
    output [7:0] qa;
    output [7:0] qb;

    always @(posedge clk)
        qa <= d;

    always @(negedge clk)
        qb <= d;
endmodule;
```

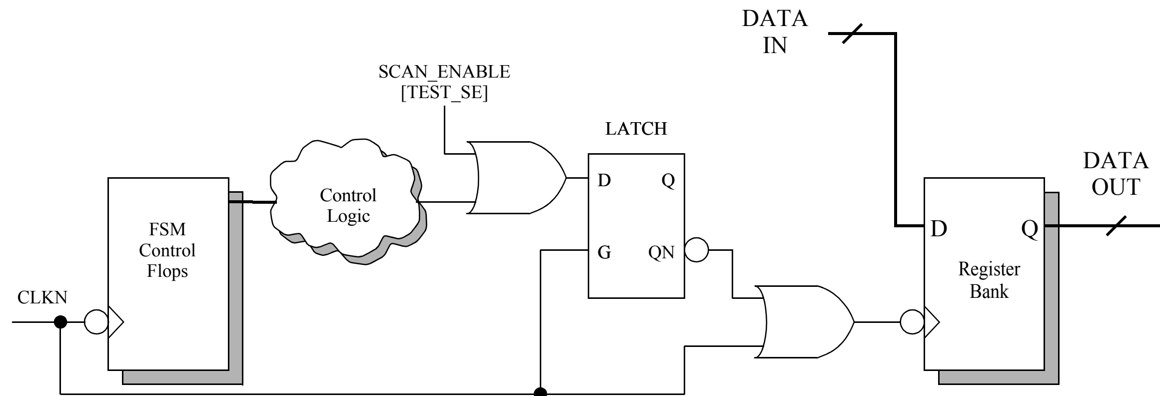
When I have management and the entire ASIC team breathing down my neck, creating an RTL test case can be a relief. It takes my mind off the problem and gives time for ideas to surface from my subconsious. I often come up with ideas when I am away from the client's site. Writing an RTL test case is a way to escape mentally and still physically be there.

A test case is also useful as a troubleshooting tool. I can email it to my peers and to Synopsys. A problem with a test case gets more attention from both my peers and Synopsys. Reducing the problem from a huge module to a small test case gives everyone something with which to experiment. Netlists produced are small and easily read with a text editer. Run times are short allowing different approaches to be analyzed.

After I emailed out the test case and the management and the ASIC team left for the night, I relaxed and the question came to me. What would DFT Compiler do if I split the clocks? I modified the RTL to have two clock ports: one clock going to the positive edge flops and another clock going to the negative edge flops. The result: DFT Compiler inferred a return-to-zero (RZ) test clock for the positive edge register bank and a return-to-one (RO) test clock for the negative edge register bank. It declared both register banks controllable and put them on the scan chain.

4.9 Gated clock insertion for negative edge register banks

Figure 5 Negative Edge Triggered Register Banks



Before explaining the lessons learned from the test case, let's look at how Power Compiler implements RTL gated clock insertion for negative edge triggered register banks. Figure 5 shows the circuit. Power Compiler inserts an OR gate to gate the clock and a transparent high LATCH to de-glitch the enable signal and de-runt the gated clock signal.

Figure 7-5[1] of the Power Compiler User Guide is wrong on how and when Power Compiler inserts an OR clock gate. Power Compiler uses an OR gate for negative edge triggered register banks and inserts a transparent high latch. So get out your red pen again and cross out the bubble in front of the gate pin of the LATCH and put a bubble in front of the clock pin for the FLOP.

1. Figure 7-5 of the Version 2003.12 Power Compiler User Guide is Figure 9-5 of the Version 2000.05 Power Compiler Reference Manual.

4.10 Lesson learned from the test case

```
dc_shell> create_test_clock -w {50 60} CLK
dc_shell> create_test_clock -w {60 50} CLKN
```

Figure 6 Tester Cycle Clock Waveforms

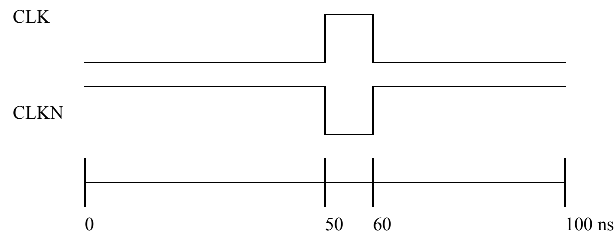


Figure 6 shows the tester cycle waveforms with the return-to-zero (RZ) clock for the positive edge register bank and the return-to-one (RO) waveform for the negative edge register bank.

When Power Compiler inserts a gated clock with a `scan_enable` control point *`before`* the `LATCH`, DFT Compiler needs the `LATCH` to be transparent at the beginning of the tester clock cycle. This is accomplished by using an RZ test clock for register banks with positive edge flops and an RO test clock for register banks with negative edge flops. When an RZ test clock was used on register banks with negative edge flops, the `LATCH` was latched (not transparent). That is why DFT Compiler declared those flops to be uncontrollable and dropped them from the scan chain.

While writing this paper I started to wonder if DFT Compiler's requirement that the LATCH be transparent at the beginning of the tester clock cycle was necessary. In the general case where DFT compiler can not make a latch transparent, controllability is an issue. The latch in a clock gating circuit is a special case. The latch feeds the gating circuit of flops that use the same clock as the gate of the latch. In the case when the latch is latched (not tranparent), the register bank will have flops that clock on the second edge of the test clock. The latch will open (become transparent) on the first edge of the test clock.

I also started to wonder what would happen if the control point was moved to *`after`* the latch. It seems to me DFT Compiler's controllability issue would go away but the timing implications need review.

4.11 Change done to the design

Figure 7 Test clocking circuitry

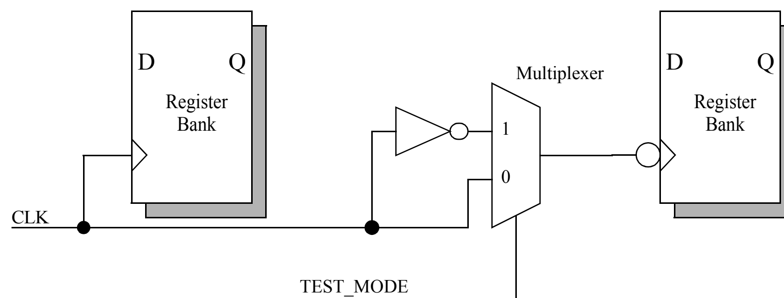


Figure 7 shows the test clocking circuitry added to the problem module. When the `TEST_MODE` signal is low, the clocks for both the negative edge and positive edge register banks are identical. When the `TEST_MODE` signal is high, the clock signal is inverted for the negative edge register banks. An RZ test clock stays RZ for the positive edge banks and becomes RO for the negative edge banks.

This fixes the controllability issue for DFT Compiler and it removes capture violations as well.

4.12 Capture violations

When the same test clock goes to both positive edge register banks and negative edge register banks the DFT Compiler `check_test` command will find capture violations. Capture violations were not an issue for this group. They intended to use the sequential ATPG feature of TetraMAX to work around capture violations.

As stated before in “Consulting Assignment Goals” on page 4, they wanted no changes to the legacy design. They were not happy about having to add the test clocking circuitry. The fact that it removed capture violations did not impress them.

5.0 Top level issues.

5.1 DFT Compiler balks again

```
Warning: Clock/enable pin %s of cell %s (%s) tied
constant. (TEST-125)
```

Now it is time to discuss top level integration. The module level DB files that were scan synthesized by the in-house DFT specialist were given to the in-house synthesis specialist for integration. The integrated DB file was given back to the DFT specialist for top level scan chain stitching. Again a DFT script that worked before gated clock insertion stopped working.

```
dc_shell> xpin = find(pin, I123/test_se)
dc_shell> xnet = all_connected(xpin)
dc_shell> report_net -connections xnet
```

I probed the top level DB file that came from the in-house synthesis specialist and found the module level `test_se` ports created by `hookup_testports` were connected to a tie-down cell at the top level. When we (I and the DFT specialist) pointed this out to the synthesis specialist, he told us that it was OK because DFT Compiler will auto-magically[1] decide to remove the ground connection.

It was time to look at the in-house synthesis specialist's top level integration script.

5.2 Legacy `set_dont_touch` everything script

```
dc_shell> set_dont_touch find(cell, I*)
dc_shell> compile
```

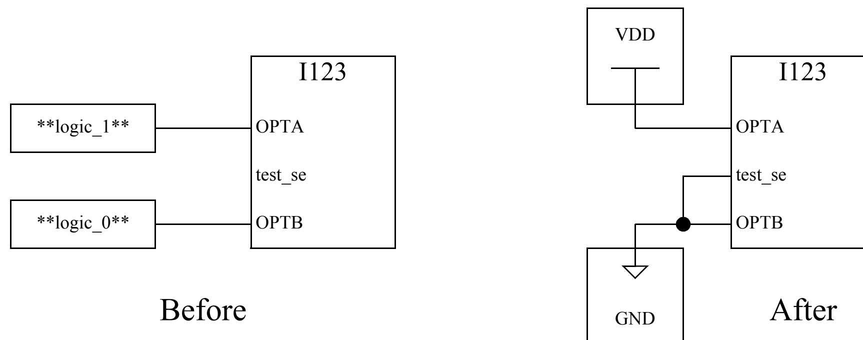
One of the things that I spotted in the top level integration script was that it was telling Design Compiler to compile the design after telling it to not touch all of the module instances. I asked the synthesis specialist what it was doing. He told me that it was "linking" the design. That didn't make sense to me. If the goal was to link the design why not just use the `link` command? Something else had to be going on[2].

1. Auto-magic is explained in "How it works under-the-hood" on page 23.
2. This was a legacy script which was not completely understood by the in-house synthesis specialist.

5.3 Diagnosing what it does

```
dc_shell> write -format verilog -out before.v
dc_shell> set_dont_touch find(cell, I*)
dc_shell> compile
dc_shell> write -format verilog -out after.v
```

Figure 8 What `set_dont_touch` everything does



One way to find what a script fragmant does is to write out a Verilog netlist before and after the script fragment and then examine it with a text editor like `emacs`. What I found that it was doing two things. It was mapping the generic tie-high and tie-low cells to technology specific tie-high and tie-low cells. It was also connecting all unconnected input ports to ground as well.

```
dc_shell> check_design

Warning: In design '%s', input pin '%s' of cell '%s' was
left unconnected. logic zero assumed. (LINT-0)
```

This was something new for me. Since I never leave unconnected input pins in my designs, I didn't know that Design Compiler does more than assume a logic zero. It connects the pins to ground. There were more unconnected input pins than just the test ports. I did a `grep -C LINT-0` on the `check_design` file and gave the results to the lead design engineer for review. He found some unconnected ports which would have caused problems in the final design.

5.4 Using the `group` command to avoid the problem

```
dc_shell> group -design_name GLUE -cell_name Iglue \
            find(cell, **logic_*)
dc_shell> current_design GLUE
dc_shell> compile
dc_shell> current_design TOP
dc_shell> ungroup Iglue
```

To map the generic tie-high and tie-low cells to technology specific cells without grounding unconnected input pins, `group` the generic cells and compile only those cells.

This solved the problem with TEST-125 warnings.

5.5 Demonstrating clock polarity issues

```
Warning: Normal mode clock pin %s of %s (%s) is
uncontrollable. (TEST-169)
```

Will this message ever go away?

```
dc_shell> xpin = find(pin, I1/clk)
dc_shell> xnet = find(net, all_connected(xpin))
dc_shell> disconnect_net xnet xpin
```

One of the modules that worked at the module level stopped working when integrated at the top level. Since TEST-169 warnings previously happened with clock polarity problems, I suspected that the module's clock was being fed the wrong polarity from the clock module. I disconnected the module's clock pin at the top level and re-ran the DFT Compiler script. DFT Compiler inferred the correct polarity test clock for the disconnected pin and the TEST-169 warnings disappeared for that module.

This is something to know about DFT Compiler. If a script omits a `set_test_clock` command, DFT Compiler will automatically infer it. In the case where it can't bring all the flops on a clock into a scan chain, it will choose a clock polarity which makes the most flops controllable.

5.6 Custom pad cells lacks functionality

```
Warning: Cell %s (%s) is unknown (black box)
because functionality for output pin %s is bad or incomplete.
(TEST-451)
```

After getting DFT Compiler to work at the top level, the in-house synthesis specialist added `insert_pads` to his integration script. Yet again the DFT specialist's script stopped working. This time they wouldn't let me into the DFT specialist's cubicle to help him. They asked me what "black-box" meant and I told them that Design Compiler black-boxes instances where there is a port mismatch. They had several design engineers attempt to help the DFT specialist and then they finally let me look at it.

This was a new error message for me. So what does one do when one encounters a new error message?

```
dc_shell> help TEST-451
```

```
TEST-451 (warning) Cell %s (%s) is unknown (black box)
because functionality for output pin %s is bad or incomplete.
DESCRIPTION
Test design rule checking has identified that this output or three-state pin does
not have a function described in the library. Test design rule checking requires
that all outputs/three-states have functions defined in order to recognize the
functionality of the cell. If any output/three- state does not have a function
specified, the entire cell will be black-boxed as is the case here.
check_test, create_test_patterns, and write_test will treat this cell as a black
box.
WHAT NEXT
You should either use a different library cell or contact your vendor/library
developer to fix this cell.
```

They had three pad cells that were custom made for them by their foundery. They lacked a functional definition. This was not a problem for synthesis because the timing arcs were defined, but it was a problem for DFT Compiler. DFT Compiler has a circuit simulator under-the-hood which it uses to check its scan insertion.

The custom pad cells were defective and would have given LIBG-16 warning messages when compiled with Library Compiler. This is another example of a person under so much schedule pressure that he ignores warning messages. I really recommend that users of these tools take the time to understand the warning messages. It will save time in the long run.

5.7 Fixing the library cell

This was a client that had such highASIC volume that they had a technical representative from their ASIC foundary permanently in residence. I went to him and got the LIB files for the custom pads. The next step was to fix them.

Where are the Library Compiler manuals? Half the time I go to a client's site the online manuals don't work and the Online Manual CD ROM is locked in a cabinet that only a SysAdmin has a key. That was indeed the case for this client which is why I pack a laptop with my own Online Manual CD ROM.

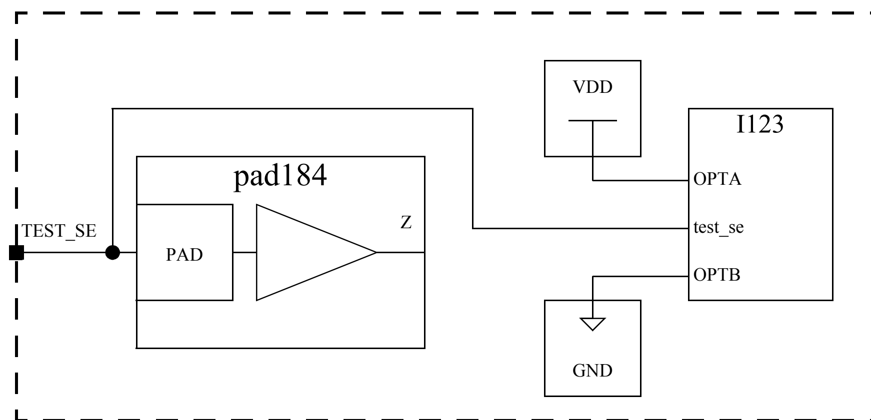
The other work-around for a lack of manuals is to access them by the web at Synopsys SolvNet (http://solvnet.synopsys.com). SolvNet will put the manual in a tiny frame in your browser. Save the PDF to disk and call it up with Acrobat Reader. Then you can see it on a full screen and you will have it when you are offline[1].

The Library Compiler has numerous examples and it was relatively easy to fix the defective LIB files.

5.8 The `hookup_testports` command fights `set_scan_signal`.

```
dc_shell> set_scan_signal test_scan_enable \
              -port TEST_SE -hookup pad184/Z
dc_shell> hookup_testports
```

Figure 9 Using `set_scan_signal` before `hookup_testports`



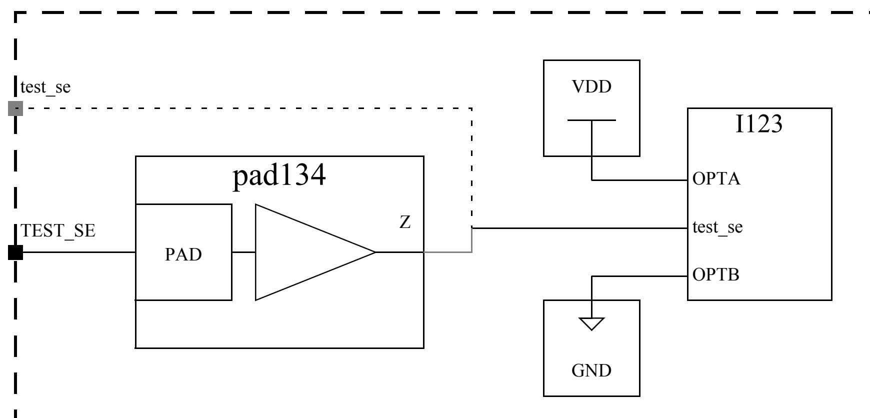
The `hookup_testports` command didn't understand the `set_scan_signal -hookup` option. It wired the scan enable pins to the wrong side of the scan enable pad.

1. The PDF Plug-in frame can also be enlarged by grabbing its top border.

5.9 The work-around script

```
dc_shell> hookup_testports
dc_shell> test_se_port = find(port, test_se)
dc_shell> test_se_net = all_connected(test_se_port)
dc_shell> connect_net test_se_net pad184/Z
dc_shell> remove_port test_se_port
dc_shell> set_scan_signal test_scan_enable \
              -port TEST_SE -hookup pad184/Z
```

Figure 10 Using `hookup_testports` before `set_scan_signal`



The work-around is to use `hookup_testports` before `set_scan_signal -hookup`. It will create a lowercase `test_se` port. Use the above script to rewire the line to the correct side of the scan enable pad buffer and then remove the port.

6.0 Test-Ready Compile

6.1 What is Test-Ready Compile?

```
dc_shell> current_design A
dc_shell> compile -scan
dc_shell> current_design B
dc_shell> compile
```

Figure 11 Modules with and without `compile -scan`

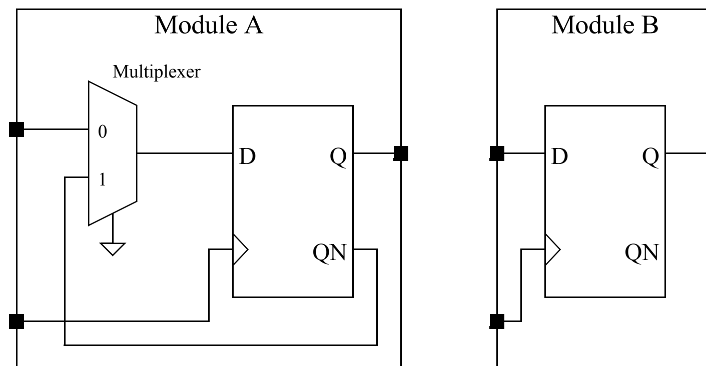


Figure 11 show Module A which has been compiled with Test-Ready compile and Module B which has been compiled without Test-Ready compile. Test-Ready compile adds a multiplexer which has its control grounded and one of the inputs of the multiplexer wired to either Q or QN of the flop.

Figure 12 Modules After Scan Synthesis

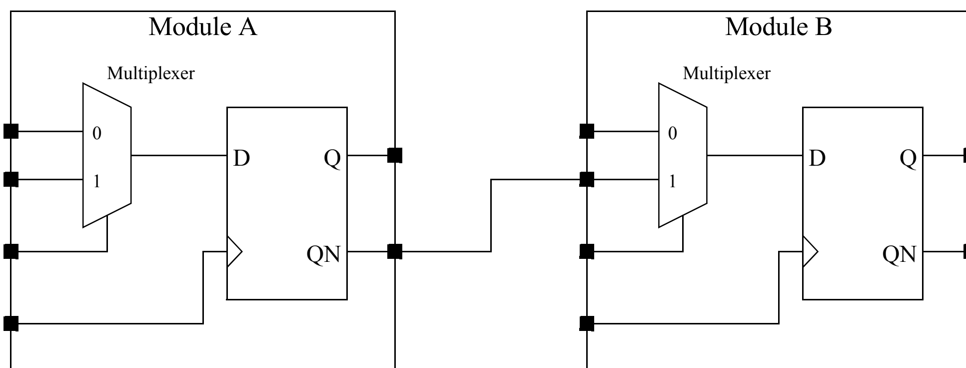


Figure 12 shows the modules after scan synthesis. For module B, DFT Compiler adds a multiplexer and three test ports. For module A, DFT Compiler also adds three test ports but instead of adding a MUX it uses the multiplexer created by Test-Ready compile. It removes the wire from the QN to input "1" of the multiplexer and removes the wire to ground for the control to the MUX.

6.2 How it works under-the-hood

```
dc_shell> report_attribute
dc_shell> get_attribute
dc_shell> set_attribute find(cell,{...})
dc_shell> set_attribute find(pin,{...})
dc_shell> set_attribute find(port,{...})
dc_shell> set_attribute find(net,{...})
dc_shell> set_attribute find(design,{...})
dc_shell> remove_attribute
```

One of the sources of confusion for the team was if DFT Compiler is smart enough to auto-magically remove grounds inserted by Test-Ready compile, why did it not remove grounds inserted by the `set_dont_touch` everything compile? The answer is that the Test-Ready compile uses the Synopsys tool attribute facility to mark its grounds for removal.

When something goes wrong with DFT Compiler, it is wise to check the attributes set by the tools. The attribute facility can also be used to work around problems.

6.3 Avoid Verilog netlists

The Synopsys tools (Design Compiler, DFT Compiler, Physical Compiler, etc.) use the attribute facility to hold design information. When writing a Verilog netlist, the attribute information is lost. Avoid using Verilog netlists to exchange information between the tools. Use Synopsys DB files instead.

I would like Synopsys to add an option that would write attribute information to Verilog netlists as pragmas. Besides information that the tools need I would like to add my own attributes such as author, version, source files, source scripts, etc. to Verilog netlists that Synopsys produces.

The SNUG Technical Committee would like Synopsys to add a `write_attributes` command similar to the `write_script` command. There comment was that keeping everything in DB files is OK but there always seems to be a need to modify netlists and compiled PERL or C is faster than an interpretive language like DC-SHELL of DC-TKL.

7.0 Recommendations and Conclusions

7.1 Synopsys Recommendations

- Enhance DFT Compiler to recognize the latch of a clock gating circuit as a special `case` that does not have a controllability problem.
- Make `hookup_testports` understand the `set_scan_signal -hookup` option.
- Add an option for writing Verilog netlists with pragmas that contain attribute information.
- Add a `write_attributes command`.

7.2 Management Recommendations

- Don't change the tool flow in the middle of an ASIC design cycle.
- Review all Synopsys warning messages before final sign-off.
- Get the online manuals loaded.

7.3 Technical Conclusions

- The default Power Compiler RTL clock gating style does not work with DFT Compiler. A control point must be added either *before* or *after* the `LATCH.`
- When the control point is added *before* the `LATCH`, feed an RZ test clock to positive edge register banks and feed an RO test clock to negative edge register banks.
- Use of the control point *after* the LATCH needs further investigation.

Appendix A. Help Pages

A.1 LIBG-16 Help Page

```
dc_shell> help LIBG-16
```

```
LIBG-16 (warning) The '%s' Pin/bus on the '%s' cell has no
'function' attribute.
The cell becomes a black box.
DESCRIPTION
This warning is issued when any output of a given cell does not have a valid
function(missing or wrong information associated with the function of the pin). A
function can be defined either by a function attribute, a three_state attribute, a
memory_read group, or an internal_node attribute.
WHAT NEXT
Check your library for missing or wrong attributes.
EXAMPLES
cell(libg16) {
area : 1 ;
pin (I0) {
direction : input ;
capacitance : 0 ;
}
pin ("Y") {
direction : output ;
/* function : "I0" ; */
timing() {
related_pin: "I0" ;
intrinsic_rise: 1.0 ;
rise_resistance : 0.0 ;
intrinsic_fall : 1.0 ;
fall_resistance : 0.0 ;
}
}
EXAMPLE MESSAGE
Warning: Line 39, The 'Y' Pin/bus on the 'libg16' cell has no 'function' attribute.
The cell becomes a black box. (LIBG-16)
```

A.2 TEST-169 Help Page

```
dc_shell> help TEST-169
```

```
TEST-169 (warning) Normal mode clock pin %s of cell %s (%s)
is uncontrollable.
DESCRIPTION
A normal mode clock pin is a clock pin that is active during normal operation of the
design. Depending on the scan style, the clock pin might be active or inactive
during scan shift. A typical example is an LSSD C Clock. In normal operation this
captures data into the LSSD cell's master latch, but is held inactive during a scan
shift while the A and B clocks are active. Another example is the clock pin of a
multiplexed flip-flop scan cell, which is active in both normal operation and in
scan shift.
The check_test command issues the TEST-169 warning when it finds a normal mode clock
pin that is in an unknown state when all clock ports of the design are held
inactive.
The most frequent cause of this violation is invalid clock gating, where the value
on the clock pin is determined by the value in another sequential element when the
clock port is held inactive.
When check_test can identify the source of this violation it also issues a TEST-281
to identify the network that causes the clock pin to be in an unknown state.
For prescan designs, it is important that you review this warning and correct it,
otherwise the cell will not be scan replaced by insert_scan and so your fault
coverage will be reduced.
WHAT NEXT
Review the accompanying TEST-281 message to identify the source of the
uncontrollable clock. If the source is a clock gating circuit, consider adding testmode
gating to ensure that the clock pin can be controlled from an external clock
port for test purposes. For example, add a multiplexer to the clock gating circuit
so when a test_mode port is held to logic 1 a clock port drives the clock pin. Now
apply a set_test_hold 1 command to the test_mode port and rerun check_test.
If you think that the clock gating is valid and should be accepted by check_test,
review the sense of the clock signals inferred by check_test. (Check the TEST-169
messages to see if Test Compiler inferred a return-to-zero or a return-to-one
clock.)
Consider a clock gating circuit that ANDs together an internal signal and a signal
from a clock port, CLK. If CLK is inferred as a return to one clock, check_test
reports a TEST-169 violation because the clock gating's output is controlled by the
internally generated signal when CLK is in its off-state of logic one. However, if
you explicitly instruct Test Compiler to use a return-to-zero clock, no TEST-169
violation is issued because the clock gating's output is at logic 0 when CLK is in
its inactive state of logic zero. You can instruct Test Compiler to use a return-to-zero
clock with the create_test_clock command. For example,
create_test_clock CLK -period 100.0 -waveform {30.0 40.0}
```

A.3 TEST-451 Help Page

dc_shell> **help TEST-451**

```
TEST-451 (warning) Cell %s (%s) is unknown (black box)
because functionality for output pin %s is bad or incomplete.
DESCRIPTION
Test design rule checking has identified that this output or three-state pin does
not have a function described in the library. Test design rule checking requires
that all outputs/three-states have functions defined in order to recognize the
functionality of the cell. If any output/three- state does not have a function
specified, the entire cell will be black-boxed as is the case here.
check_test, create_test_patterns, and write_test will treat this cell as a black
box.
WHAT NEXT
You should either use a different library cell or contact your vendor/library
developer to fix this cell.
```

A.4 LINT-0 Help Page

dc_shell> **help LINT-0**

```
LINT-0 (warning) In design '%s', input pin '%s' of cell '%s' was
left unconnected. %s assumed.
DESCRIPTION
This message appears when a component instance has an unconnected input pin.
Synopsys tools assume a logical value (for example, logic zero or logic one) for
such unconnected signals, based on the type of technology library you are using.
This warning message is a notification of the assumption that is being made for the
named input pin in the named design.
WHAT_NEXT
Verify that the assumption made by Synopsys regarding the logical value for the
given signal is correct. If the assumption is not correct, then connect the floating
input pin to the correct logical value in your design.
```