How To Successfully Use Gated Clocking in an ASIC Design

Darren Jones

MIPS Technologies, Inc.

dj@mips.com

ABSTRACT

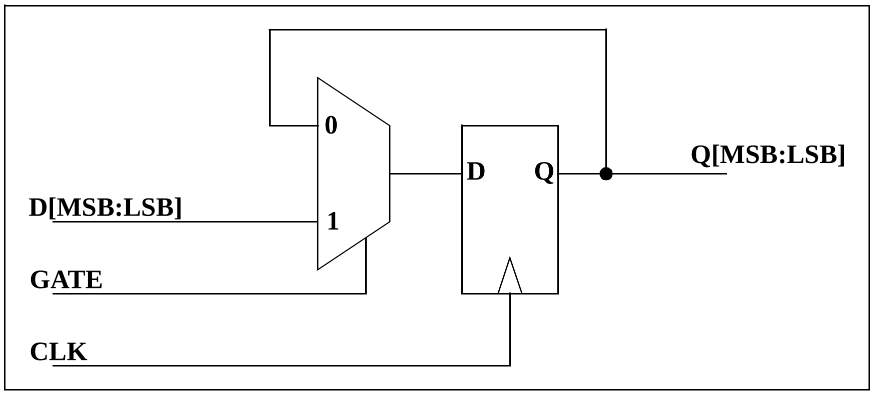
Gated clocking is a true silver bullet for hardware designers. Using this technique, engineers can improve all three major performance metrics of a circuit: speed, area, and power. Unfortunately, EDA tools have traditionally lacked support for gated clocks. These limitations have relegated clock gating to the full-custom design community. However, new features in synthesis and static timing analysis tools have brought gated clocking to mainstream ASIC designers.

This paper discusses the pitfalls that still exist to using gated clocks in an ASIC design. Furthermore, it suggests methodologies and workarounds that can be used to avoid these problems so that gated clocking can be used successfully. The paper explains how gated clocking impacts the following areas: logic synthesis, static timing analysis (STA), automatic test-pattern generation (ATPG), clock tree synthesis, and standard-cell library design.

1.0 Description of Problem

ASIC designers primarily use positive edge-triggered D flip flops to generate registers and/or storage elements. These flip flops are clocked every cycle; if they need to hold their previous value, a recirculating MUX circuit is typically used. Figure 1 shows this circuit.

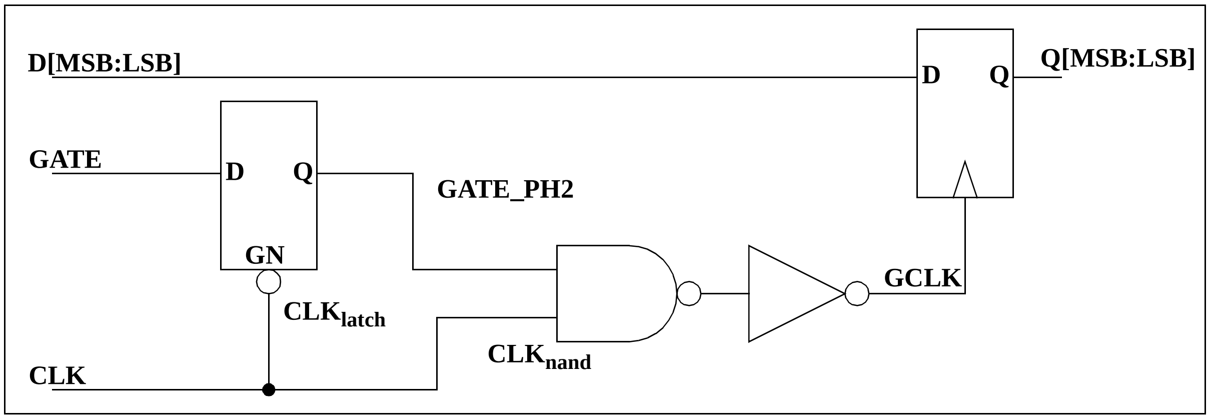
Figure 1. Recirculating MUX Schematic



While this circuit is conceptually simple, it can be improved upon in several ways. Figure 2 shows a functionally equivalent circuit using a gated clock. This circuit is higher performance because it removes the MUX from the timing-critical data input to the flops. Removing these MUXes also saves area. Finally, this circuit is lower power since the flops are not clocked in cycles they do not need to be clocked.

NOTE: There are many possible ways of implementing gated clocking. However, most can be generalized to the circuit in Figure 2.

Figure 2. Gated Clocking Schematic



The remainder of this paper uses this circuit as a basis for discussion, so a detailed explanation of the logic is needed. In this circuit, the positive pulse of the clock signal is either enabled or disabled by the gate signal. Thus, in any cycle when **GATE** is deasserted low, **GCLK** will remain low and no positive edge will be propagated to the downstream flip flops. The transparent-low latch is used to hold the gate stable over the positive pulse of **CLK** in order to prevent clock glitching. In order to be glitch free, there is a setup and a hold requirement at the NAND gate.

Figure 3. Clock Glitch Setup Check

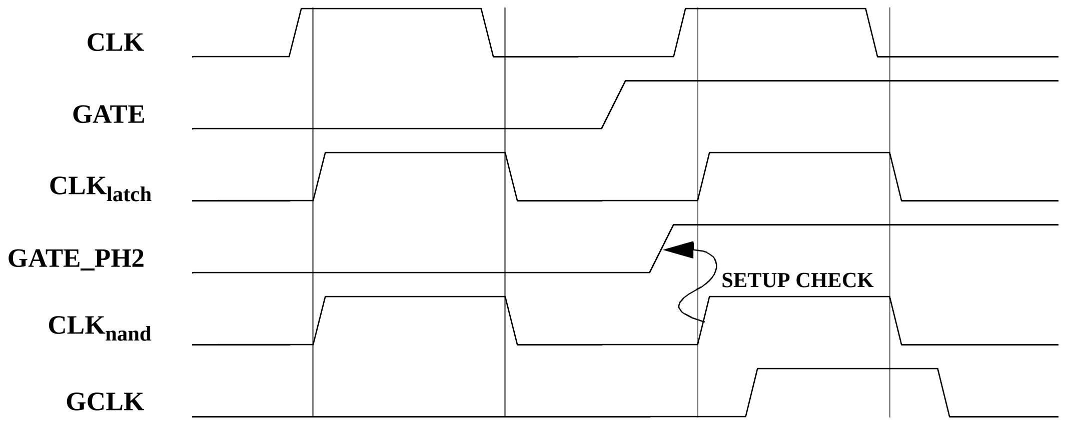


Figure 3 shows the waveform for analyzing setup time checks. In this case, **GATE** changes late in the cycle. The setup requirement occurs at the positive edge of the clock at the NAND gate, $\mathbf{CLK_{nand}}$. The setup check analysis must use the following timing:

- The clock path starts at **CLK** and ends at the NAND gate. The timing must be for delays of the *positive edge* of the clock.
- The data path starts before **GATE**, which is itself generated from flip-flops and logic clocked by the positive edge of **CLK**, goes through the latch (D->Q), and ends at the NAND gate. The timing must be for *either positive edge or negative edge* of **GATE**, whichever is longer.

Note that in this analysis, the latch is open and therefore, can be thought of as simply a delay element in the path. Ideally, this path would not be analyzed as a latch path.

Figure 4. Clock Glitch Hold Check

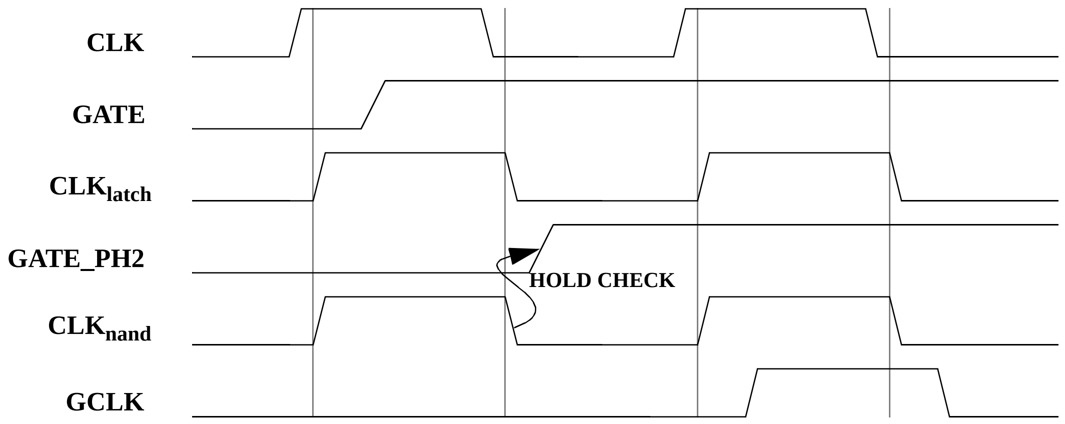


Figure 4 shows the waveform for analyzing hold time checks. In this case, **GATE** changes early in the cycle, during the positive pulse of the clock. Since **GATE_PH2** must be stable over the entire positive pulse of the clock, it must have net positive hold time against **CLK$_{nand}$**. There is a race here because **GATE_PH2** is also launched from the negative edge of the clock, which opens the latch. The hold check analysis must therefore use the following timing:

- The clock timing path starts at **CLK** and ends at the NAND gate. The timing must be for delays of the *negative edge* of the clock.
- The data timing path starts at **CLK**, goes through the latch (GN->Q), and ends at the NAND gate. The timing must be for the negative edge of the clock to the latch and then for *either positive edge or negative edge* of **GATE_PH2**, whichever is shorter.

It is advantageous if **CLK$_{latch}$** is a little later than **CLK$_{nand}$** since this would give additional hold time margin without impacting the setup check.

2.0 Gated Clocks and Synthesis

Synopsis synthesis tools do have clock glitch checks built in. But, they sometimes do not do the correct analysis. However, the optimization algorithm is robust enough to overcome this problem and still achieve optimal results. This section explains how all this works.

2.1 Enabling Clock Glitch Checking

First of all, you must enable clock glitch checks since they are disabled by default. To enable clock glitch checks, issue the following command (TCL):

```
set_clock_gating_check -setup $setup_val -hold $hold_val
```

Where:
`$setup_val` is the amount of setup needed at the NAND gate on the data input to allow the output to be glitch free.
`$hold_val` is the amount of hold time needed at the NAND gate to allow the output to be glitch free.

NOTE: The above values should be proven using SPICE simulations.

2.2 Setup Time Checks

Synopsys tools (Design Compiler, Physical Compiler, and Primetime) recognize the latch in the gating circuit and assume this is a latch path. The latch analysis algorithm divides the setup analysis into two parts, which I call frontside and backside paths:

Frontside Path. This is the path in front of the latch. It includes all of the gating logic and ends at the D input of the latch. It is a normal logic path included in the **CLK** path group.

Backside Path. This is the path in back of the latch. It starts at the latch output and ends at the NAND gate. This path is a clock gating check path.

Synopsys' latch analysis algorithm will first analyze the frontside path. It will borrow time across the latch so that the frontside path can meet timing. Then, it analyzes the backside path given the amount of borrowing it did on the frontside. What this tends to do is leave violations on the backside path, even though the frontside path appears to be meeting timing.

While this algorithm does work well for generic latch-based designs, it is exactly the opposite of what is needed for the clock gating circuit. Since the backside path is actually a clock glitch check, it should take priority over the frontside path, which is a normal logic path. The analysis should first time the backside path, then set the borrowed time to the maximum available time such that there are no clock glitches. Then, the frontside path will correctly reflect the slack on the entire path. This is exactly what you want, since the synthesis tool cannot synthesize any logic in the backside path anyway.

In spite of the above analysis problems, the synthesis tools will try to improve the frontside path, even if it appears to meet timing. Thanks to the latch optimization algorithm, the tool recognizes that a failure in the backside path can be improved by optimizing the frontside path. However, even this feature may not be enough to get correct results. In the cases where there is a failure on both the frontside and backside paths, the total path failure (the sum of frontside failure and backside failure), will be split into two parts. This will mask what may be one of the most critical paths in the design and the tool may not select these paths for optimization since separately, they do not appear to be the most critical paths.

You could use `set_critical_range` if your worst violation is small. However, this solution can significantly increase synthesis runtimes. The recommended solution is to use `set_max_time_borrow` to limit the amount of borrowing so that the backside path just meets timing and all of the violation is seen on the frontside path. The exact command to use is given in the next section.

2.3 Impact of Ideal Clocks

As the reader may recall, when an ideal clock is used, all clock endpoints are assumed to have perfect skew, but the clock uncertainty is subtracted to account for skew. With the gated clock circuit, you can see in Figure 2 that the clock to the latch and the NAND gate will be early when compared to the clock of the flip-flops. Thus, paths ending at these latches will have a reduced cycle time and require tighter timing than normal paths do.

The above problem does not impact the clock glitch hold check, since this path is a 0-cycle path. However, it does impact the setup time optimization since by default, the tool will optimize for a full cycle, not a reduced cycle.

The recommended workaround for this problem again uses `set_max_time_borrow` to limit the amount of time that can be shifted to the frontside path to account for this reduced cycle time. Combining this solution with the previous solution for the split paths gives the following command:

```
set_max_time_borrow [expr $phase - $unc - $clkbufdelay - $dqdelay] \
  [get_clock $clk]
```

Where:

`$phase` is the phase time of the negative phase of the clock.
`$unc` is the clock uncertainty at the latches
`$clkbufdelay` is the propagation delay for the posedge of the clock through clock buffering that is downstream from the gating element, (including delay through the gating element)
`$dqdelay` is the propagation delay through the latch.

The phase time available for borrowing is reduced by three factors. First, clock uncertainty: there is still uncertainty about when the clock will arrive at the NAND gate, although this uncertainty may be less than the full-chip uncertainty. Second, the clock buffer delay: this factor is applied due to the early clock effect described in this section. Third, the latch D->Q propagation delay: this factor must be included because the delay through the latch itself is included in the backside path and is not available for borrowing into the frontside path.

The above command forces synthesis to see the backside path just meet timing. At the same time, it forces the frontside path to use all the available time in the reduced cycle.

2.4 Setup Time Check Reporting

One further ramification of the splitting of the gating path into frontside and backside paths is that the setup time check appears at first glance to be incorrect. The following shows a report from DesignCompiler for a sample backside setup clock gating check.

```
Startpoint: U_gate_ph2 (negative level-sensitive latch clocked by clk)
Endpoint: U_gclk_n (gating element for clock clk)
Path Group: clk
Path Type: max

Point                                    Incr       Path
-----------------------------------------------------------
clock clk (fall edge)                   50.00      50.00
clock network delay (propagated)        12.00      62.00
time given to startpoint                21.00      83.00
U_gate_ph2/D (LATCHN)                    0.00      83.00 r
U_gate_ph2/Q (LATCHN)                   20.00 *   103.00 r
U_gclk_n/A (NAND2)                       0.00 *   103.00 r
data arrival time                                 103.00

clock clk (rise edge)                  100.00     100.00
clock network delay (propagated)        10.00     110.00
clock uncertainty                       -0.20     109.80
U_gclk_n/B (NAND2)                       0.00     109.80 r
clock gating setup time                  0.00     109.80
data required time                                109.80
-----------------------------------------------------------
data required time                                109.80
data arrival time                                -103.00
-----------------------------------------------------------
slack (MET)                                         6.80
```

This report shows the data path beginning at the falling edge of the latch clock. As was described earlier, the correct setup analysis must start from the positive edge of the clock and propagate through the gating logic and then through the latch is if it were transparent. In fact, this is what the tool is doing. The following listing shows the report for the associated frontside latch setup path.

```
   Startpoint: U_internal_sig1(rising edge-triggered flip-flop clocked by clk)
   Endpoint: U_gate_ph2 (negative level-sensitive latch clocked by clk)
   Path Group: clk
   Path Type: max

   Point                                    Incr       Path
   -----------------------------------------------------------
   clock clk (rise edge)                    0.00       0.00
   clock network delay (propagated)        10.00      10.00
   U_internal_sig1/CLK (DFF)                0.00      10.00 r
   U_internal_sig1/Q (DFF)                  3.00 *    13.00 f
   U_internal_sig3/Y (INV)                 70.00 *    83.00 r
   U_gate_ph2/D (LATCHN)                    0.00 *    83.00 r
   data arrival time                                  83.00

   clock clk (fall edge)                   50.00      50.00
   clock network delay (propagated)        12.00      62.00
   U_gate_ph2/GN (LATCHN)                   0.00      62.00 f
   time borrowed from endpoint             21.00      83.00
   data required time                                 83.00
   -----------------------------------------------------------
   data required time                                 83.00
   data arrival time                                 -83.00
   -----------------------------------------------------------
   slack (MET)                                         0.00
```

As can be seen, this path does in fact start from the positive edge of the clock and arrives at the input to the latch at time T=83. The backside analysis reproduces this path timing and shows time T=83 at the latch input. It then propagates timing through the open latch correctly to the NAND gate. The tool does a good job of obfuscating the true analysis, but it does in fact time the correct path.

3.0 Gated Clocks and Primetime STA

Primetime uses similar algorithms to its synthesis brethren for analyzing clock glitches. Again, these checks have to be enabled, since they are disabled by default:

```
      set_clock_gating_check -setup $setup_val -hold $hold_val
```

More importantly, the problems addressed during synthesis must also be addressed during STA:

- STA does not have the analysis problem associated with ideal clocks as long as STA is run with detailed actual clock network delays.
- STA does incorrectly divide the clock gate path into two separate paths, thus obscuring potential critical paths.
- STA has a similar reporting deficiency for clock_glitch setup checks. However, as was true for

synthesis, STA does time the correct paths.

The problem in the second bullet must be addressed differently for STA than it was for synthesis. In synthesis, we used estimated for clock timing. With STA, we have actual net delays, and so we can correct the borrowing for each latch individually to be 100% accurate. However, the spirit of the workaround is the same: use `set_max_time_borrow` to allow just enough time on the backside path for glitch-free operation while forcing all available slack to the frontside path. Here is the TCL code to use: (line numbering added for clarity)

```
1: set timing_include_available_borrow_in_slack true
2: set paths [get_timing_paths -group **clock_gating_default** \
     -max_paths 10000]
3: foreach_in_collection path $paths {
4:       set slack [get_attribute $path slack]
5:       set tb [get_attribute $path time_lent_to_startpoint]
6:       set stpt [get_attribute $path startpoint]
7:       set_max_time_borrow [expr $tb + $slack] $stpt
8: }
```

As the reader may recall, by default, Primetime will only borrow enough time to exactly meet timing. Thus, while a latch path may have lots of slack on the backside, the frontside is normally reported as having 0 slack. From the timing reports, this makes it difficult to see if the path actually has 0 slack, or if it has positive slack that is just not being reported. The command on line1 allows Primetime to report positive slack for those paths that could borrow more time. This is done because we are going to push slack to the frontside and we want it to report positive slack instead of 0 slack.

After line2, `$paths` will contain a collection of all the clock gating paths. We will `set_max_time_borrow` on each of these paths individually.

We want to push all slack from the backside clock_gating check to the frontside logic path. The commands inside the loop get the backside slack and the time that was borrowed in order to get this result. Then, it adjusts the maximum borrow time of the frontside path according to the slack. Thus, if the clock_gating check was failing, this will reduce the available borrow time so that it will pass. If the backside path was passing, it makes all of this slack available to the frontside path. Accurate failures will now be lumped onto the frontside path.

4.0 Gated clocks and ATPG

ATPG is almost always one of the biggest concerns for ASIC designers when using gated clocks. In reality, it is not a difficult problem to solve. It breaks down into two parts- enabling scan testing, and optimizing ATPG results.

4.1 Enabling Scan Testing

The most important thing is to be able to run scan tests. For this, the scan chain must be clocked when it is being shifted (when `scan_enable` is asserted). The easiest way to assure this is to disable all gating elements when `scan_enable` is asserted. This amounts to an OR gate on the **GATE** input, basically forcing **GATE** on when `scan_enable` is asserted.

This OR gate can be placed in front of the latch or behind the latch. If it is in front, then it looks just like any other logic in front of the latch. If it is behind the latch, then there is one benefit and one penalty. The benefit is that it adds extra hold time margin for the clock glitch hold check. It does this by delaying the data signal, which helps the clock win the race.

The penalty for putting the OR gate behind the latch is that the scan_enable signal must then be stable over phase1 of the clock, since it does not go through the latch. In practice this is not difficult to guarantee. This should not impact performance, since scan is only run at relatively slow speed.

NOTE: The OR gate itself does impact the setup path, but this is unavoidable. The choice of its location in front of or behind the latch has no effect on the setup path, since during setup analysis the latch is open and the OR gate will be in the path no matter which location is chosen.

4.2 Optimizing ATPG Results

Once you have forced clocks to run during scan shifting, the degree of coverage achieved is largely dependent on your ATPG tool. Popular ATPG tools can handle gated clock circuits. This means that they can properly model an unclocked flop and generate patterns which can detect faults in the gating logic.

Using gated clocks does introduce one ATPG untestable fault: stuck-at-0 on the latch enable input, **GN**. Since the latch is included purely for timing reasons, if it is stuck open, then the circuit may still functionally pass, depending on timing. This is inherently untestable by scan test patterns. However, since the latch is needed to guarantee hold time, it is a zero-cycle path which may fail when the gate signal changes values early in a cycle. This makes faults of this nature more likely to be detectable during slower-speed operation. Timing analysis can be run to prove which faults will be detected and which will not affect proper operation of the circuit.

Even if your scan tool does not support gated clocking, there is a workaround. You can replace all registers that use gated clocking with the recirculating MUX circuit just for ATPG. This workaround relies on the fact that the circuit shown in Figure 1 is functionally equivalent to the gated clocking circuit. Patterns which run on the recirculating MUX will also run on the gated clock circuit. This solution is not as accurate as using a more advanced ATPG tool which directly supports gated clocking, but it will generate legal patterns with decent coverage, if not perfect coverage.

5.0 Gated Clocks & CTS tools

For some reason, most mainstream clock tree synthesis tools are relatively primitive and have trouble achieving good skew even with vanilla non-gated clock trees. As one might imagine, there will also be limitations when using a gated clock tree. Nonetheless, this section describes several guidelines to follow. If your CTS tool can support all of these guidelines, you are home free. If it cannot follow any of them, give up and do not use gated clocks. For most design teams, their CTS tool will be somewhere in between these two extremes. Success has been achieved using popular CTS tools and a little intelligent scripting.

Do Not Skew-match the Latch or NAND gate clock endpoints with the Flops. Most CTS tools will automatically assume all clock endpoints should be skew-matched against all others. This is not true for the clock gating circuit. The latch clock and the NAND gate clock will necessarily be earlier than the flip-flop clocks and should not be skew-matched together.

Do Skew-match Latch and NAND gate clock endpoints against each other. The previous rule is not meant to allow poor skew on the gating element clocks. You must still achieve good skew on these endpoints, when taken as their own group.

Do Skew-match the gated flops with un-gated flops. Most designs using gated clocks will have some logic which is not gated. Make sure that the CTS tool skew matches all of the flops: gated and non-gated.

Place the Latch and the NAND gate close together. These two cells need to have the same clock and have well-controlled delays on the nets between them. Placing them close together helps to achieve these goals.

Route the Latch and the NAND gate on the same clock Subnet. There needs to be very tightly controlled skew between these two cells. Achieving this is most easily accomplished by connecting these two cells to the same physical wire.

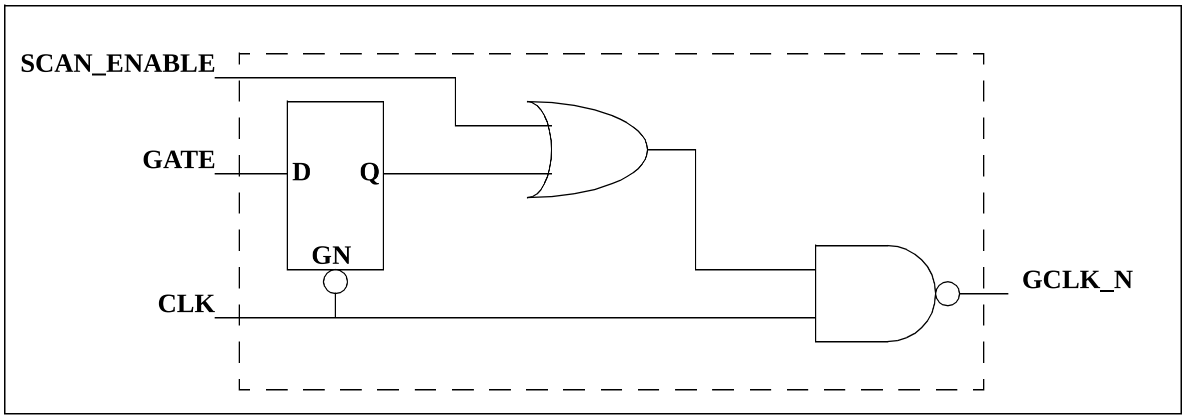
Make the Clock Gating circuit the leaf level Clock Driver. As stated previously, the latch clock is earlier than the clock to the flip-flops. How early it is may directly impact the maximum frequency of the design. Thus, the clock gating circuit wants to be at the leaf level of the clock tree. In other words, there should be an absolute minimum of clock buffering after the NAND gate. This rule implies that the entire clock gating circuit should be duplicated, not just the downstream buffers.

6.0 Gated Clocks & Cell Libraries

Now that you have read and understood the previous pitfalls, workarounds, and suggestions, there is one final suggestion which can solve many of the above problems all at the same time. If your standard cell library has a few strategic cells designed for clock gating, many of the previously documented problems go away. The cells required can be divided into two groups- the front-end gating cell and the backend clock driver.

The front-end gating cell is comprised of the latch and the NAND gate. In addition, to help ATPG, we include an OR gate for enabling scan shifting. Figure 5 below shows the schematic for this, the GCK cell:

Figure 5. GCK Cell



This cell must be designed such that it never generates clock glitches. It would have the following timing arcs:

- Setup and hold time requirements at the **GATE** input relative to the positive edge of **CLK**.
- Setup time requirement at the **SCAN_ENABLE** input relative to the positive edge of **CLK**.
- Hold time requirement at the **SCAN_ENABLE** input relative to the negative edge of **CLK**.
- Delay from positive edge of **CLK** to the negative edge of **GCLK_N**.
- Delay from negative edge of **CLK** to the positive edge of **GCLK_N**.

The backend cell is a plain inverter used to invert **GCLK_N** to **GCLK** and drive downstream flip-flops. This cell should have a large variety of drive strengths, since the drive strength selections will largely determine how well skew can be matched. Furthermore, all of the backend cells should be the same size and have the same IO pin locations. This effectively means that all cells will be the size of the largest cell. The advantage of this is that it makes for easy swapping of these cells for improving clock skew. Since there are relatively few of these cells in the design, their size has little effect on overall chip size.

By having these two types of cells, you will see the following benefits:

- Clock gating checks are not necessary in synthesis and STA, since the clock is guaranteed to be glitch free.
- The frontside path is seen as a regular cycle path by synthesis and STA due to the posedge requirements on the gate signal.
- The ideal clock problem in synthesis can be addressed through a semi-customizeable synthesis model for the front-end gating cell. You can artificially increase the setup time requirement by the amount of clock insertion delay, thus squeezing the frontside path into a reduced cycle.
- Scan testing is enabled because the gating cell is always enabled when **SCAN_ENABLE** is asserted.

As you can see, these simple cells greatly simplify many of the problems inherent to clock gating.

7.0 Conclusions and Recommendations

Ten years ago, virtually no mainstream ASIC design tools supported gated clocking. Now, Synopsys and other EDA companies have begun to address this design style. This paper showed how to successfully navigate various stages of chip design to successfully integrate gated clocks with your design.

While most tools do not by default do the correct thing, they can be directed to the correct operation by a few intelligent scripts. Furthermore, a few strategic standard cells can also go a long way toward alleviating the remaining tool limitations. Even though manual work is still required, the benefits of gated clocking to performance, area, and power are well worth the effort needed to implement them successfully.

8.0 Acknowledgments

It was Soumya Banerjee who suggested that my work on clock gating might be useful to SNUG members.