

RTL Low Power Techniques for System-On-Chip Designs

Mike Gladden

Motorola, Inc.

Austin, TX

rwdb80@email.sps.mot.com

Indraneel Das

Synopsys, Inc.

Austin, TX

ineel@synopsys.com

ABSTRACT

Low power design remains a complex and critical challenge for System-On-Chip (SOC) designs which often involve the reuse of existing internal and/or external Intellectual Property (IP), while often incorporating new IP as well. This approach to IC design might involve multiple, unrelated clocks, convergent clocks, clocks being used as enables, and vice versa. SOC Design Houses often integrate designs originally developed by remote, or even international teams. This paper addresses issues that might come up while defining a low power design methodology for real-world SOC designs. The paper also addresses issues involving automated clock-gating techniques applied to legacy designs. The applicability of various techniques are discussed using an SOC design test case at Motorola. Various styles of RTL are contrasted to compare their applicability to automated clock-gating. Implications to the back-end flow are discussed.

1.0 Introduction

Power consumption and its control used to be almost an afterthought in the methodology for Integrated Circuit(IC) Design. As transistor counts continue to increase, coupled with reducing feature sizes, system complexities have increased as well: thereby increasing clock frequencies and an increased market focus on portable application platforms. In non-portable applications, it has become more expensive to provide adequate cooling to semiconductor devices, which may impact system costs and how much system integration may be provided on Silicon. Other concerns such as electro-migration and self-heating (Ohmic heating) directly affect system reliability and the Mean Time Between Failures(MTBF). Today, power consumption often ranks in the top four or five areas of concern for Design Managers: sometimes right after functionality and speed. Design for low power, however, has collateral costs associated with it. Additional steps and checkpoints need to be inserted into the design project's life cycle: thereby increasing training requirements and project schedule. In the end, however, the resultant product would stand a better chance of meeting its power and reliability goals.

Techniques for low power may be applied at various levels during a product development life cycle. An SOC design may typically comprise a Processor core to serve it's computation needs, perhaps an additional DSP core for specific applications, assorted memories such as Flash and EEPROM, and various peripherals interconnected with a Bridge or a bus controller. Clearly, such systems require the use of diverse techniques for design, and for implementing a low-power methodology. Power consumption is heavily application-dependant. The same processor would have different power characteristics when used in a printer application and in a cellular phone.

Low power design techniques may be applied at various domain levels during a design cycle, from specification to physical integration and tapeout.

2.0 Domains of Low Power Design

Minimizing power consumption reduces to the twin problems of reducing the supply voltage, and reducing the effective capacitance seen by nodes that may switch during operation. The following well-known equation demonstrates the quadratic dependence of power consumption on the supply voltage:

$$P_{dyn} = C_l V_{dd}^2 f P$$

Where P_{dyn} is the dynamic power dissipation, C_l is the capacitive load seen by the device, V_{dd} is the supply voltage, f is the frequency of 0->1 transitions, and P is the probability of the occurrence of a 0->1 transition (i.e., a power-consuming transition)[1]. Static power dissipation is assumed to be significantly smaller than dynamic dissipation. Low-power design techniques may be applied to an SOC design during the four design domains of Architecture definition, RTL definition and coding, Gate level implementation, and Transistor Level implementation (usually for custom blocks and analog circuits). Low power techniques have a greater impact at higher levels of abstractions during a design cycle. As the domain shifts from a high-level definition (such as architecture definition) to low-level implementation (such as transistor level designs), the incremental improvement 'per unit design time' reduces drastically. Memory and other large structures that cannot be synthesized, are, of course, exceptions.

3.0 Architecture and Specification

As mentioned before, though a reduction in supply voltage results in a quadratic reduction in power consumption, it also results in loss of performance. Smaller geometries, and micro-architectural changes usually compensate for this loss in performance. Analyses would be needed to examine the impact (on power consumption) of larger effective capacitances of the faster structures being used, in relation to the lower supply voltages they can run at. Some semiconductor vendors are also implementing designs with multiple supply voltages on the same die: a technique that allows designers to explore tradeoffs between low power and operational speeds.

Using parallel architectures[2] and pipelining are some of the techniques used to maintain datapath throughputs while reducing the supply voltage. Power-area tradeoffs need to be carefully examined in such cases.

Though decisions made in the architectural domain would have significant impact on power consumed by the target system, there are no commercially available tools specifically for low power design exploration at the architectural level. The task is generally completed by senior designers based on their experience and by experimental implementation of specific blocks.

4.0 Low Power Design at RTL and Gate Levels

For low power design, SOCs today are most amenable to techniques developed for the RTL and Gate level domains. The techniques may be developed uniformly for the entire design team, ensuring consistency and predictability in the design strategy. Synopsys' strategy provides SOC designers with a set of tools to address power analysis and optimization issues, which address low power design issues pertaining to accurate modeling of libraries, analysis of results at all levels of abstraction, and gate level optimization based on those results.

4.1 Modeling

Accurate modeling forms the foundation for automatic power management techniques at the RTL and Gate levels. Library models used in the SOC designs need to be prepared to be able to analyze power data, and transistor level models need to be prepared for custom blocks.

4.2 Estimation

Design Power™ is the gate level average power analysis tool from Synopsys[3], which can help SOC designers manage power consumption by providing quick estimates of power consumption early in the design cycle by using switching activity information from RTL simulation data, and synthesis library information. Later on in the design phase, it can use gate level simulation data to come up with more accurate estimates. Figure 4.1 describes the flow for power estimation. Gate level analysis is more accurate than RTL level analysis, since it accounts for glitches and has support for state and path-dependencies. PowerGate™ from Synopsys is another tool that can be used for time-based analysis of power supply specifications, hot spots and peak power.

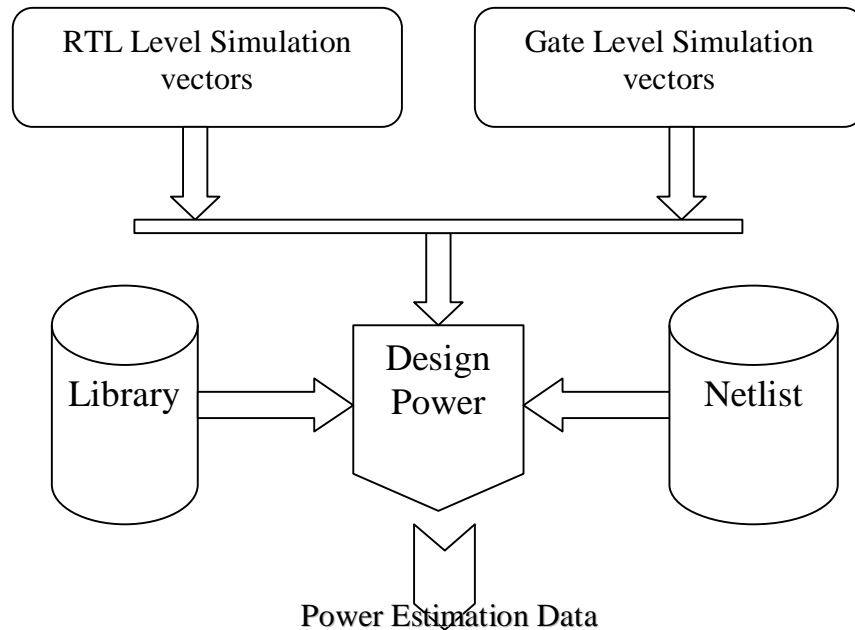


Figure 4.1 Power Estimation using Design Power™

4.3 Implementation and Optimization

Power optimization can be applied to the RTL implementation phase for SOCs using Power Compiler™. Power Compiler works at two levels. In the RTL domain, Power Compiler performs automated clock-gating by identifying synchronous load-enable register banks, and implementing them by gating the clock with the enable instead of recirculating the data when the enable is inactive. This technique saves power, and in many cases, saves area as well. At the gate level, Power Compiler optimizes delay, area and static and dynamic power dissipation simultaneously, to meet user-defined constraints. The implementation works on an unmapped, or Generic Technology (GTECH™) implementation created by elaborating the RTL source with the ‘*-clock_gate*’ switch. Figure 4.2 demonstrates the flow:

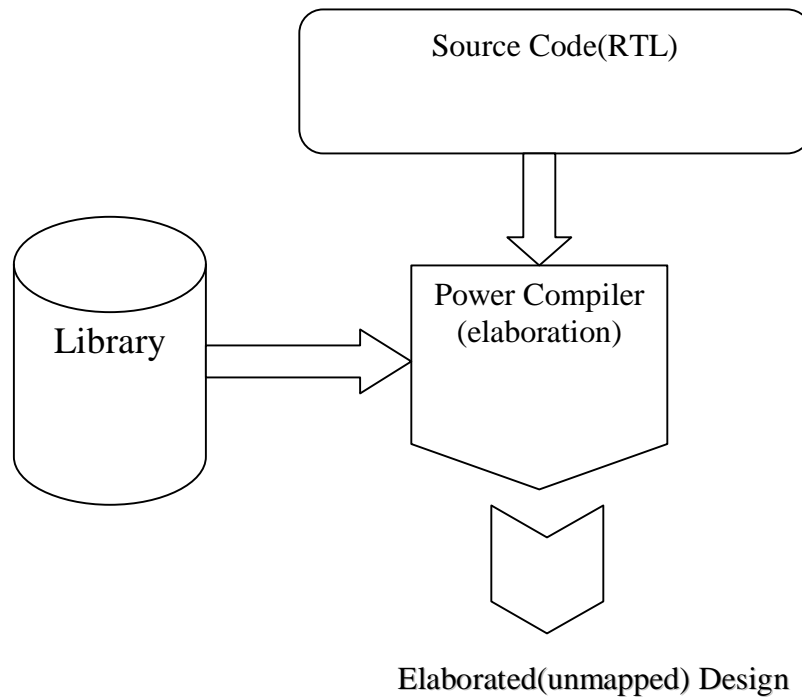


Figure 4.2 Gated Clock Implementation using Power Compiler

Figure 4.3 demonstrates the conversion of synchronous load-enable implementation of muxes into a gated clock implementation, by Power Compiler. The benefits of this conversion are: a reduced internal power consumption at the clock-gated flip-flops, elimination of the muxes (since there is now no need to recirculate data, resulting in area and power savings), and a reduced power consumption by the clock network. There are implications on testability, which are discussed in Section 5.5 .

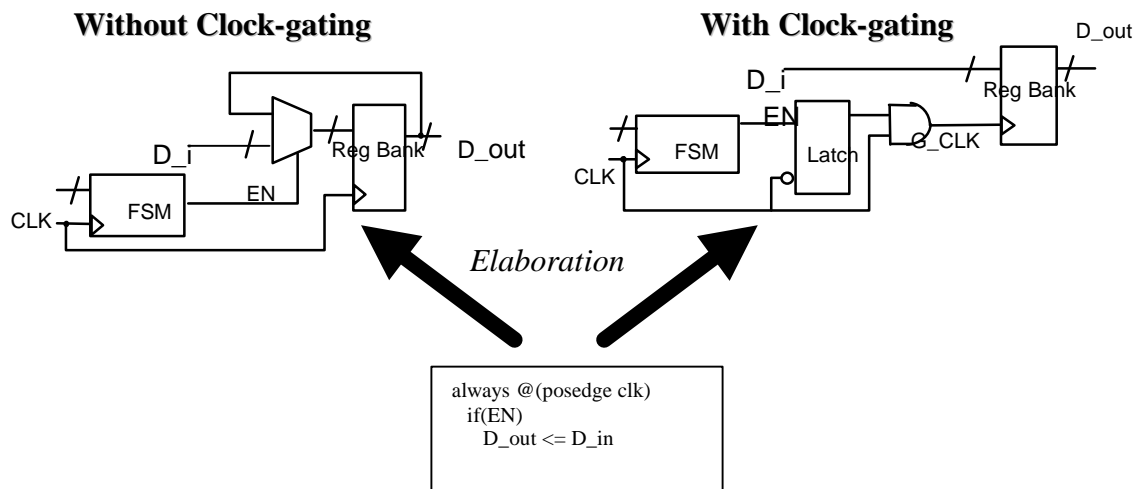


Figure 4.3 Automated Clock-gating in RTL

The elaborated design is mapped by Design Compiler™ onto target libraries. By inserting gates into the clock network, clock pulses are prevented from propagating to the register's clock pin when the enable is inactive. Average power consumed by the registers is

thereby reduced. Furthermore, overall power consumption drops because the multiplexers and associated routing are eliminated. Though Power Compiler's clock-gating works on register banks, it may be complemented by manually inserted module-level clock-gating. Power Compiler allows users to choose between a combinational-only, and a latch based clock-gating style (Figure 4.4). Users may choose which register banks to gate or exclude from gating, whether they want Positive (AND) or negative (OR) gating logic, and what the minimum bit-width of gated registers should be.

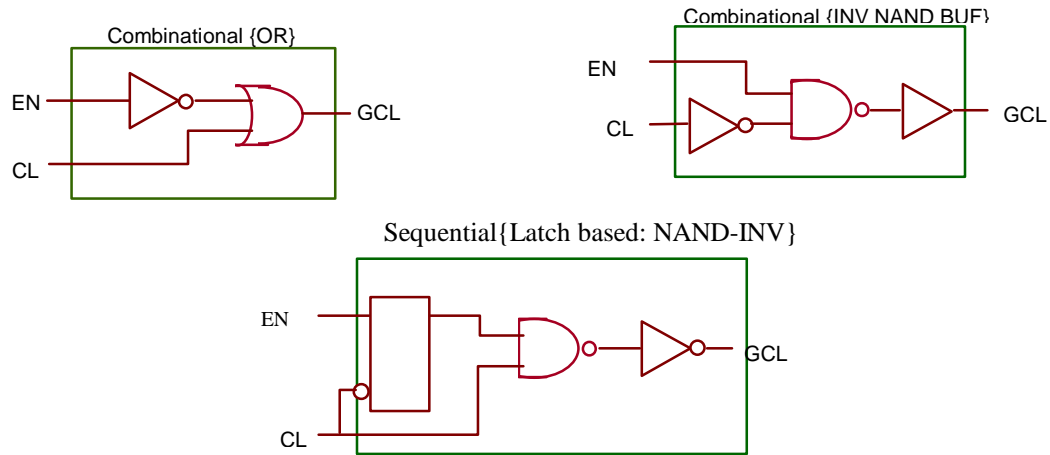


Figure 4.4 RTL Clock-gating Styles using Power Compiler

The latch based clock-gating technique, while requiring more area than the combinational scheme, is protected from glitches in the 'active' phase of the clock. When the clock input (CL) is at '1', any glitch on EN could, in the absence of the latch, potentially corrupt a register bank clocked by GCL. The latch prevents glitches on the enable signal (EN) from propagating to the gated clock net (GCL). Power Compiler allows the user to specify setup and hold time requirements for the enable, with reference to the input clock.

Power Compiler can also share the same gating logic across multiple register banks, if the enable and the clock are common.

4.4 Custom Techniques

Custom circuit structures, including non-synthesized modules need to have manually implemented strategies for low-power design. Transistor level power analysis, optimization, and characterization tools such as PowerMill, RailMill, PowerGate, PowerArc and AMPS are available for the purpose. Transistor-level low-power design techniques are outside the scope of this paper, and deal with properly sizing gates to sharpen transition times (thereby reducing 'through' or short-circuit currents), and ensuring lower capacitance on high-activity nets such as clock pins. Library cell development is a critical area as well.

5.0 RTL Clock-gating Methodology: Motorola SOC Case Study

This section describes techniques used in the development of a 16-bit Micro-Controller Unit (MCU) developed by Motorola (block diagram in Figure 5.1). The device is an SOC developed for automotive applications, and incorporates processor cores, embedded volatile and non-volatile memories, and a set of peripheral modules typically found in a majority of automotive applications.

On this project, Motorola designers and Consultants from the Synopsys Professional Services Group worked jointly as a team, for an extended period, to develop and implement the synthesis methodology and scripts used during the design.

The design methodology used a bus-macro-based design flow, which also allowed easy partitioning of the design from a Power Compiler based clock-gating perspective. A multi-site design team executed the project. This physical separation placed special demands on the design process, in terms of managing design data, formulating a set

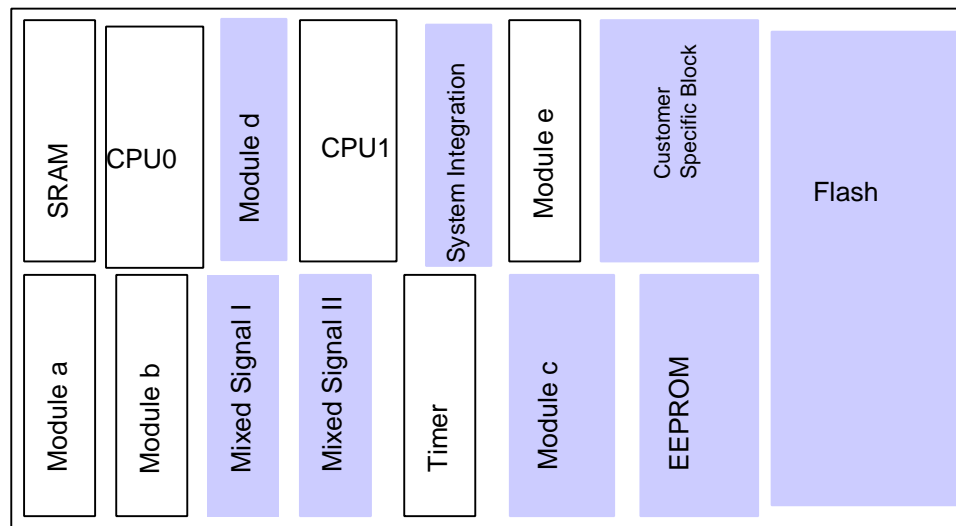


Figure 5.1. Block Diagram of Motorola's Automotive Application SOC

of common goals and tracking progress for the entire set of project teams. Furthermore, the modules that make up the SOC presented a variety of design and implementation challenges. For instance, the EEPROM has internally generated clocks, and the Flash Memory module's memory core has a custom clocking structure, which had to be excluded from the generic clock-gating and clock tree synthesis methodology used for other modules.

5.1 Methodology in Practice

Prior to the use of automated clock-gating using Power Compiler, designers in our group used instantiated clock-gating techniques: using continuous assignment statements outside of procedural blocks. As an option, manual instantiations of clock-gating structures were also used. Depending on the experience of the designer, this method can add complexity to the implementation and analysis, leading to a longer design cycle. This

increase in design cycle time was caused by the need to partition RTL with the enable logic in separate modules and procedural blocks and using labels which were easily identifiable as inputs to the combinational gate. *set_max_delay* constraints were set from the raw(ungated) clock to the output of the Nand-Inverter pair (Figure 4.4). The review process was tedious, since each clock-gating scenario had to be reviewed individually, particularly for setup and hold violations by the enable being gated. Using Power Compiler automated the process of ensuring compliance with setup and hold time specifications for the clock-gating element. Since the SOC used a ‘bus-macro’ approach, all designers were simultaneously informed of I/O timing and electrical requirements for their respective modules. Designers more or less proceeded in lock step, through the various stages in design: from RTL through Place and Route. This *Concurrent Design* effort necessitated some sort of centralized mechanism of archiving, changing and broadcasting synthesis and methodology directives. Accordingly, a generic set of scripts was developed that could be used by all module designers (Figure 5.2). This set of scripts would contain all ‘basic’ constraints, from a bus-macro perspective. Global scripts and templates were provided for setting up synthesis variables, clocks, the project’s default compile strategy, and generating reports. These global scripts had the ‘hooks’ to incorporate module-specific constraints, exceptions, and compile strategies, if necessary.

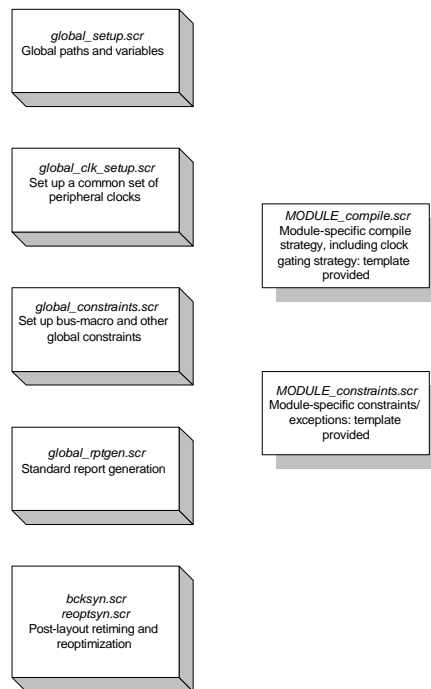


Figure 5.2. Scripts Used for SOC Concurrent Design Flow

Periodic sync-ups were performed to ensure all team-members were using current revisions of scripts, and had incorporated the latest changes into their strategies. User group meetings were conducted regularly to facilitate communication between team members and help them exchange information regarding problem resolutions. Reviews were conducted to verify that the low-power strategy had been correctly and consistently

applied: from library cells used to implementation issues such as clocking and reset strategies. Uniform and consistent RTL coding guidelines were followed, to accurately model the design that may be correctly interpreted by Power Compiler. Scripts and synthesis directives were placed under centralized control, with periodic updates being broadcast to the entire team simultaneously.

Having a centralized broadcast mechanism for script management and updates provided significant benefits to the team. There was a conscious attempt to provide designers with a quick, push-button approach to RTL clock-gating that was robust, consistent, and provided significant reduction in power consumption.

5.2 RTL Style and Power Compiler Command Issues

Most SOC Designs try to optimize design cost functions by leveraging existing code. The problem of ‘retrofitting’ such legacy RTL source to an automated clock-gating methodology using Power Compiler is a challenging one, mostly because the older code may not have been originally written for a synthesis-based design flow. Additionally, the legacy RTL might not follow coding guidelines for clock-gating available in the documentation. In general, the source code should follow guidelines for coding specified in the Power Compiler Reference Manual [3] and the HDL Compiler Reference Manual [4].

The following sections outline some recommendations which may be used to evaluate and implement a Power Compiler-based RTL clock-gating scheme on SOC Designs, and includes recommendations while setting command line options for Power Compiler.

1. For hierarchical modules, Power Compiler allows the inclusion and exclusion of designs from the general clock-gating strategy (with the command *set_clock_gating_signals*). Each design that needs to undergo this ‘customization’ must have a single command line for both, the inclusion list and the exclusion list. Specifying exceptions on multiple lines for the same design would only preserve the last one.
2. If specific bits out of a Register bank need to be excluded from clock-gating, they need to have different enables than the rest of the register bank. Using the same enable for all bits in a register bank will result in a common clock-gating logic on all bits. Thus, it is not possible to exclude, for instance, *data_bus[4:0]* from being clock gated, if they are part of *data_bus[15:0]*, if all bits in *data_bus* have a common clock and enable. Conversely, if separate enables *are* used for *data_bus[4:0]* and *data_bus[15:5]*, the two bit-vectors groups will be clock gated using separate logic.
3. Master-slave flip-flops should not be clock gated, since at the level of abstraction where clock-gating occurs, flip-flops are represented generically, and clock-gating logic may be implemented on the slave clock signal (provided it satisfies clock-gating conditions). The resulting circuit may not function properly. Master-slave flip-flops may be excluded from the clock-gating scheme using the *set_clock_gating_signals – exclude* command.
4. Clock-gating single or dual-bit signals has questionable benefits, since the additional logic added due to clock-gating may consume more power and need more area than the ‘feedback muxes’.

Generally, setting the minimum bit-width (*set_clock_gating_style* – *minimum_bitwidth*) threshold to five bits or more is recommended.

5. For some procedural blocks containing *case* statements that are not fully defined (i.e., those with *default* conditions), and which have embedded conditional statements, the *default* condition needed to be included within the ‘*if then else*’ construct for each case. This suggestion for change only applied to a minority of case statements in the test design, and should only be applied on an as-needed basis.

For instance:

```
case (A1)
  2'h0: if(B1) C1 <= D1 ;
  1'h1: if(B2) C1 <= D2 ;
  default: C1 <= E1;
endcase
```

needs to be converted to:

```
case (A1)
  2'h0: begin
    if (B1) C1 <= D1 ;
    else C1 <= E1 ;
  end //block 2'h0
  2'h1: begin
    if (B2) C1 <= D2 ;
    else C1 <= E1 ;
  end //block 2'h1
```

6. Power Compiler has the useful feature of being able to combine the clock-gating logic for multiple register banks if they share the same enable and clock signals. This feature helps save area as well. Thus, the same clock-gating logic would be generated for the following two procedural blocks (*proc_block_a* and *proc_block_b*), since they share the same enable(*e*) and the same clock(*clk*):

```
always @ (posedge clk or posedge reset)
begin :proc_block_a
  if (reset)
    out_a <= 'b0;
  else if (e)
    out_a <= in;
end: proc_block_a
```

```
always @ (posedge clk or posedge reset)
begin :proc_block_b
  if (reset)
    out_b <= 'b0;
  else if (e)
    out_b <= in_b;
```

end: proc_block_b

5.3 Clocking Strategy

Devising a simple global clocking strategy eases the adoption of automated clock-gating using Power Compiler. A two-level clock-gating scheme was used for this project, with the first level being instantiated at module boundaries. The second level was for internal registers, using Power-Compiler. Keeping the number of clock domains to a minimum simplified timing analysis and clock-tree synthesis issues with signals crossing over clock domains. Lower frequency modules should have enables, rather than divided-down clocks, to simplify timing analysis and ease automated clock-gating.

The *set_dont_touch_network* command should be used to prevent further compile changes on clock networks, after compilation with clock-gating. This will prevent any modification to the gating logic during a multi-step compile process, and assist an easier route through gate-level functional verification.

5.3 Sets and Resets

A simple strategy, e.g. one where a common asynchronous reset is used for all registers is recommended. Complex set and reset strategies may result in circuits that are less amenable to gate level functional debug. Some SOC designs have internal logic to generate set and reset signals. Such logic should be in a separate logical partition for synthesis, than the register banks these signals go to.

5.4 Clock Balancing

Clock-tree Synthesis (CTS) tools are commonly used to route clock trees for synthesized designs. These tools usually work by resizing, moving, adding or deleting buffers and inverters along clock tree nets to manage skew and insertion-delay requirements. Gating logic on clock tree nets makes this problem harder, since these gating elements now have logic gates associated with them, as opposed to buffers. Furthermore, a latch-based clock-gating scheme is a more complex structure than a combinational clock-gating one: some CTS tools have only recently begun supporting latch-based clock-gating schemes, in terms of being able to balancing the tree to meet target insertion delay and skew.

Back-end clock buffer insertion delays for the gated clock structure used on our project are represented in Figure 5.3 as delays $t1$, $t2$, and $t3$. Clock balancing in most commercially available CTS tools attempts to balance all sequential elements in the input netlist. Excluding the latch leaf port in the gated clock structure cannot be easily avoided. Power Compiler's combinational setup and hold directives should account for this delay in both, pre-layout (ideal clock situation), and post-layout (propagated clock situation) cases. Prior to the availability of layout data, when ideal clocks are used, the setup should be specified as the expected post-layout insertion delay $t1$. To avoid post-layout timing violations, a pessimistic value for the worst case expected insertion delay was used on the peripheral modules on this SOC.

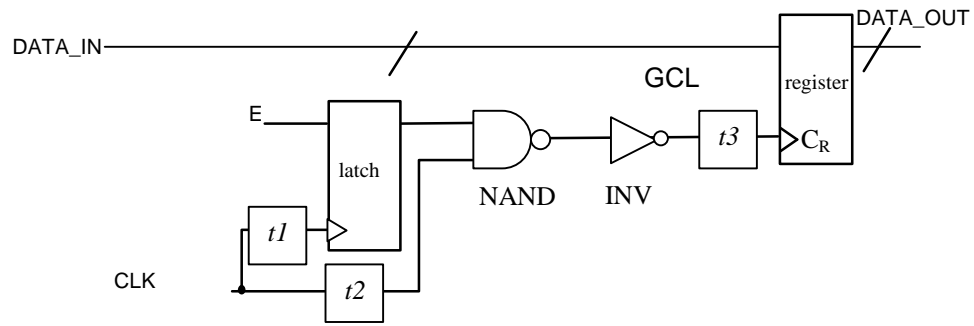


Figure 5.3. Clock Buffer Insertion

One limitation of the CTS tool used on this project was its inability to optimize the combinational devices in the gated clock path, except for the final inverter (INV, in Figure 5.3). This led to less than optimal delays from the clock tree root (CLK) to leaf ports of gated flip-flops (C_R). As shown in Figure 5.3, large t_2 insertion delays should be avoided, since this can potentially cause significant unbalancing between the enable (EN) and the combinational gate's clock input (i.e., between the two input pins of the NAND gate shown in Figure 5.3). This is especially true for any post-layout optimization, since on our project, the *set_dont_touch_network* command was used on the clock networks after buffer insertion by the CTS tool. The CTS tool then did a good job at minimizing the t_2 delay, mainly as it attempted to equalize the $t_2 + t_3 +$ combinational gate delay, to the t_1 delay.

A new feature in Power Compiler, which would allow the use of an integrated clock-gating cell, comprising the latch and the combinational gate as a single library element should address this issue (for latch-based clock-gating schemes). CTS tools can then treat the latch-gate combination as a single 'psuedo-library' element, and not be concerned with what is internal to the cells.

Even so, the back-end CTS flow using clock-gating is not fully automated for many tools available today, and it is recommended that a 'test module' be taken completely through the back-end flow to validate the process

Furthermore, correlation tests should be run between skew and insertion delay numbers reported by the CTS tools and DesignTime™ for clock nets, to ensure consistency of delay-calculation results between the two. If there is a big discrepancy, the back-end iteration loop becomes a tougher challenge.

5.5 Testability

Clock-gating requires additional area to improve controllability and observability of faults. This is because clock-gating effectively introduces multiple clock domains into the design, necessitating additional logic to increase testability. Power Compiler's *set_clock_gating_style* command has options to determine the amount and type of test logic to be added during clock-gating. The user may choose between *test_mode* or *scan_enable*, add a control point for testing (using an OR gate implementation), and add observability logic (Figure 5.4).

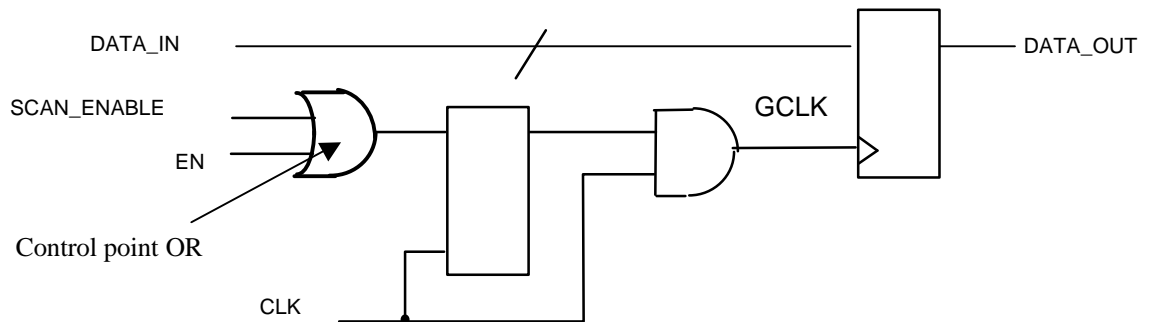


Figure 5.4 OR Gate as a Control Port

Power Compiler also supports automatic generation of XOR Trees and observability registers for the clock enables. This has the benefit of making Finite State Machine (FSM) faults testable, and also that the testability logic does not consume power during normal operation. However, it comes with the cost of a larger area (Figure 5.5).

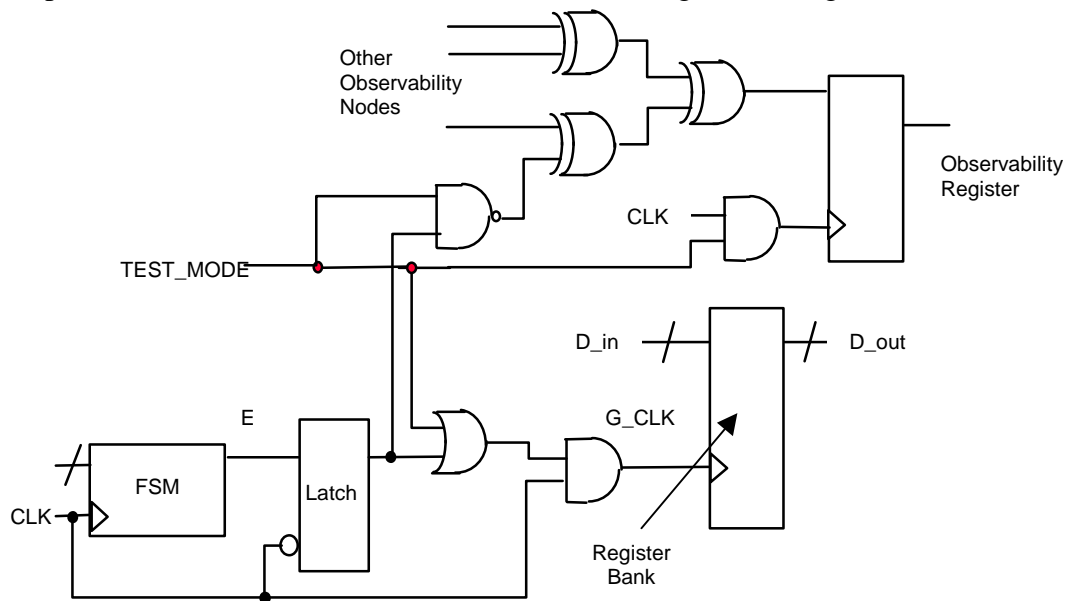


Figure 5.5 Higher Observability with Larger Area

The Test Compiler™ products from Synopsys[5] support Power Compiler clock-gating, and the Reference Manual may be referred to for more details.

5.6 Other Factors in the Equation

Designers should be made aware that RTL clock-gating is only one of many options to consider when devising a low-power strategy for SOCs. Different logic and circuit elements are amenable to different techniques. Architecture, micro-architecture, library,

device, and the software application issues all play a role in the overall power consumption of a system. A low-power design strategy, for maximum benefits, needs to address issues at all design domains.

5.7 Results

Results obtained from using Power Compiler on the Customer Specific Module (CSM) are presented in Table 1.0. The CSM was a 3,500 cell design with 40 user-software-selectable byte-wide registers. With minimum bitwidth set to 3, Power Compiler was able to take advantage of this structure to produce a smaller and more power-efficient design. The power numbers reported are for the clock network, and the use of Power Compiler resulted in a 340% power improvement accompanied by a 15.4% area improvement.

<i>Power Compiler Option</i>	<i>Area(sq. microns)</i>	<i>Power(mw)</i>
ON	1.58	1.62
OFF	1.87	5.56

Table 1.0

6.0 Conclusions

Low power design has come to be a critical factor in the definition of SOC design methodologies. Products designed using such flows go into a variety of applications, such as portable communications, computing, and automotive electronics devices. Other reasons favoring low-power techniques are increased reliability and smaller areas. With increasing complexity and integration, SOC design teams need to continually improve design cycles to counteract the engineering productivity gap. Automated tools for low-power design, optimization and analysis will therefore find increasing acceptance and use in the design community.

In this paper, we have described the relative impact of injecting low power techniques at various domains in a design. Maximum benefits can occur from intelligent choices made at the architectural domain, but there is a lack of commonly available commercial tools that allow design exploration based on power consumption considerations.

Automated RTL clock-gating offers a relatively easy and push-button way of implementing low-power design into synthesized modules. However, the technique needs to be carefully applied if it is intended for use on legacy source code. Project management for multi-site teams has its own challenges while trying to implement low power techniques in a design flow. Recommendations for coming up with a viable clock-gating strategy have been presented, including suggestions on RTL coding styles, Power Compiler issues, and the back-end flow.

7.0 References

1. Digital Integrated Circuits: A Design Perspective: Jan Rabaey, Prentice Hall, 1996.
2. Low Power CMOS Digital Design: A. Chandrakasan et.al., Proc. IEEE JSSC, pp1082-1085, 1992.

3. Power Compiler Reference Manual: Synopsys, Inc., v1998.08.
4. HDL Compiler for Verilog Reference Manual: Synopsys, Inc., v1998.08.
5. Test Compiler Reference Manual: Synopsys, Inc., v1998.08.