MINIMIZATION OF POWER DISSIPATION OF THE PROCESSOR
# CHAPTER - 4

## Contents

**CHAPTER - 4**  
MINIMIZATION OF POWER DISSIPATION OF THE PROCESSOR  

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page No.</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1</td>
<td>Applying Clock Gating To 16 Bit Processor</td>
<td>48</td>
</tr>
<tr>
<td>4.2</td>
<td>Clock Gating Implementation Using Power Compiler</td>
<td>53</td>
</tr>
<tr>
<td>4.3</td>
<td>Clock-Gating Cells</td>
<td>54</td>
</tr>
<tr>
<td>4.4</td>
<td>Clock Gating</td>
<td>56</td>
</tr>
<tr>
<td>4.4.1</td>
<td>Benefits of Clock Gating</td>
<td>57</td>
</tr>
<tr>
<td>4.5</td>
<td>Clock-Gating Conditions</td>
<td>61</td>
</tr>
<tr>
<td>4.6</td>
<td>Inserting Clock Gates</td>
<td>63</td>
</tr>
<tr>
<td>4.7</td>
<td>Clock Gating Flows in Compiler</td>
<td>65</td>
</tr>
<tr>
<td>4.7.1</td>
<td>Inserting Clock Gates in Gate-Level Design</td>
<td>66</td>
</tr>
<tr>
<td>4.8</td>
<td>Specifying Clock-Gate Latency</td>
<td>69</td>
</tr>
<tr>
<td>4.9</td>
<td>Specifying Setup And Hold</td>
<td>70</td>
</tr>
<tr>
<td>4.10</td>
<td>Choosing Gating Logic</td>
<td>72</td>
</tr>
<tr>
<td>4.11</td>
<td>Selecting Clock-Gating Style</td>
<td>73</td>
</tr>
<tr>
<td>4.12</td>
<td>Choosing A Specific Latch And Library</td>
<td>77</td>
</tr>
<tr>
<td>4.13</td>
<td>Keeping Clock-Gating Information In A Structural Net list</td>
<td>79</td>
</tr>
<tr>
<td>4.14</td>
<td>Clock-Gate Optimization Performed During Compilation</td>
<td>80</td>
</tr>
<tr>
<td>4.15</td>
<td>Multistage Clock Gating</td>
<td>81</td>
</tr>
<tr>
<td>4.16</td>
<td>Hierarchical Clock Gating</td>
<td>82</td>
</tr>
<tr>
<td>4.17</td>
<td>Reporting Command For Clock Gates and Clock Tree Power</td>
<td>84</td>
</tr>
<tr>
<td>4.18</td>
<td>Clock-Gating Report Using Default Settings</td>
<td>85</td>
</tr>
</tbody>
</table>
CHAPTER 4
MINIMIZATION OF POWER DISSIPATION OF THE PROCESSOR

In modern processors the clock distribution network is responsible for an increasing fraction of the dynamic power consumption [35]. The Figure 4.1 shows the breakdown of power consumption for a recent high-performance Microprocessor [34].

Figure 4.1 Processor Power Breakdown [34]

The clock power is expected to increase as the complexity and the operating frequency of the circuits keep growing as a result of technology scaling [28,39]. Designing the clock tree has thus become critical not only for performance, but also for power, and the development of new modeling capabilities and synthesis techniques that help in controlling the clock tree power effectively is one of the challenges that EDA engineers currently have to face[37]. Clock gating is a technique used to reduce power dissipation in clock distributed
network. This is achieved by shutting down the clock of any component whenever it is not being used or accessed. It involves inserting combinational logic along the clock path to prevent the unnecessary switching of sequential elements. By shutting down the idle units we can prevent the circuit from consuming unnecessary power. A portion of the clock tree can also be shut down by masking off the clock at the internal node of the tree using an AND gate. This prevents wasteful switching in the clock tree and saves power in the clock tree in addition to saving power in the functional units which are fed by the clock. Different solutions for minimizing the power consumed by the clock tree have been investigated in the recent past.

**4.1 APPLYING CLOCK GATING TO 16 BIT PROCESSOR**

In this section we have presented the minimization of power dissipation of 16 bit Processor by applying the clock-gating technique at fine grained level. In clock gating technique clock is disabled to a circuit to save power by eliminating power dissipation on clock network by preventing unnecessary activity in logic modules. In the processor architecture, identified the units for which the clock is to be gated and the condition for the gating is evaluated separately for each of the module. In the case of register file only source and destination are used in the execution the other registers are in idle condition hence the clock is masked with the AND gate by using an enable signal. The Figure 4.2 shows how the various modules are connected to the clock through the masking AND gate with an enable signal.
The condition for activating the enable signal for various modules is found out by carefully analyzing the functionality and timing diagram of the processor.

**Figure 4.2 Clock gating at fine grained level**

The Figure 4.3 shows how a clock enable signal is derived for flip flop, whenever there is no change between previous output and present input the clock signal is masked by the clock_enable signal which is computed by performing XOR operation between D and Q.

**Figure 4.3 Clock Enable signal for Flip Flop**

Here the cost we are paying for saving the power is extra logic circuit overhead. The carry_flag and zero_flag are implemented with this method. The group enable signal is generated for 16 bit registers.
as they consist of group of flip flops. The instruction based clock gating is also incorporated by carefully partitioning the CPU registers into blocks and a common clock enable signal is derived to turn off the register group independently [41].

The table 4.1 shows the operand registers used for the execution different instructions of the processor. It can be seen from the table that four instructions does not need any register operands. Two instructions use only one register operand and the remaining nineteen instructions use two register operand.

Table 4.1 Number of operand registers for various Instructions

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Instruction Word</th>
<th>Number of Operand registers used</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>Opcode</strong></td>
<td><strong>source</strong></td>
</tr>
<tr>
<td>NOP</td>
<td>00000</td>
<td>??</td>
</tr>
<tr>
<td>ADD</td>
<td>00001</td>
<td>Reg X</td>
</tr>
<tr>
<td>SUB</td>
<td>00010</td>
<td>Reg X</td>
</tr>
<tr>
<td>AND</td>
<td>00011</td>
<td>Reg X</td>
</tr>
<tr>
<td>NOT</td>
<td>00100</td>
<td>Reg X</td>
</tr>
<tr>
<td>RD</td>
<td>00101</td>
<td>??</td>
</tr>
<tr>
<td>WR</td>
<td>00110</td>
<td>Reg X</td>
</tr>
<tr>
<td>JMP</td>
<td>00111</td>
<td>??</td>
</tr>
<tr>
<td>JZ</td>
<td>01000</td>
<td>??</td>
</tr>
<tr>
<td>OR</td>
<td>01001</td>
<td>Reg X</td>
</tr>
<tr>
<td>Instruction</td>
<td>Code</td>
<td>Reg X</td>
</tr>
<tr>
<td>-------------</td>
<td>-------</td>
<td>-------</td>
</tr>
<tr>
<td>XOR</td>
<td>01010</td>
<td>Reg X</td>
</tr>
<tr>
<td>INC</td>
<td>01011</td>
<td>Reg X</td>
</tr>
<tr>
<td>DEC</td>
<td>01100</td>
<td>Reg X</td>
</tr>
<tr>
<td>MUL</td>
<td>01101</td>
<td>Reg X</td>
</tr>
<tr>
<td>SHL</td>
<td>01110</td>
<td>Reg X</td>
</tr>
<tr>
<td>SHR</td>
<td>01111</td>
<td>Reg X</td>
</tr>
<tr>
<td>ROL</td>
<td>10000</td>
<td>Reg X</td>
</tr>
<tr>
<td>ROR</td>
<td>10001</td>
<td>Reg X</td>
</tr>
<tr>
<td>ADC</td>
<td>10010</td>
<td>Reg X</td>
</tr>
<tr>
<td>SBC</td>
<td>10011</td>
<td>Reg X</td>
</tr>
<tr>
<td>RLC</td>
<td>10100</td>
<td>Reg X</td>
</tr>
<tr>
<td>RRC</td>
<td>10101</td>
<td>Reg X</td>
</tr>
<tr>
<td>DIV</td>
<td>10110</td>
<td>Reg X</td>
</tr>
<tr>
<td>HALT</td>
<td>11111</td>
<td>??</td>
</tr>
</tbody>
</table>

The table shows the operand register names and their operand code. As the processor consists of six operand registers, at a time either 2, or 1, or no register is used in the execution of instruction and the remaining registers are unused hence a clock gating logic is designed to disable the clock for the unused registers.
Table 4.2 Operand register names and their operand code

<table>
<thead>
<tr>
<th>Operand code</th>
<th>Register</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>R0</td>
</tr>
<tr>
<td>001</td>
<td>R1</td>
</tr>
<tr>
<td>010</td>
<td>R2</td>
</tr>
<tr>
<td>011</td>
<td>R3</td>
</tr>
<tr>
<td>100</td>
<td>R4</td>
</tr>
<tr>
<td>101</td>
<td>R5</td>
</tr>
<tr>
<td>110</td>
<td>R6</td>
</tr>
</tbody>
</table>

The figure 4.4 shows the clock gating logic used for the registers. The clock gating logic block generates two types of enable signals. Group_en is a group enable signal which is used to enable and disable the clock for the entire group of operand registers.

Figure 4.4 Clock gating logic for registers

This signal is used while executing the branching instructions as no register operand is required. The enable signals (EN0, En1... EN6) are
used for enabling and disabling the clock signal to the individual operand registers (R0,R1...R6). When enable signal is zero the corresponding AND gate will mask the clock to the corresponding operand register. In the designed processor to execute a program of 30 instructions without application of clock gating logic input clock of all the 6 operand registers will toggle, if each instruction on average take two clock cycles to complete execution then clock will toggle for 360 times (30*6*2). When clock gating technique is applied as mentioned in above section, at a time either 0, 1 or 2 registers operands are needed for execution hence if we take 2 register operands on average for a program of 30 instructions clock will toggle for 120 times (30*2*2). The power saving achieved in register access is 50%, but the clock gating logic circuit introduced will itself consumes some power hence overall power saving in register access will be less than 50%. The implementation of clock gating to the controller unit is done by using power compiler automatically as it needs a detailed step by step observation of all the states of FSM. The following sections explain the application clock gating principle by using power compiler.

4.2 CLOCK GATING IMPLEMENTATION USING POWER COMPILER

Clock gating is a dynamic power reduction method in which the clock signals are stopped for selected register banks during times when the stored logic values are not changing [50]. One possible implementation of clock gating is shown in Figure 4.5.

Clock gating is particularly useful for registers that need to maintain the same logic values over many clock cycles. Shutting off
the clocks eliminates unnecessary switching activity that would otherwise occur to reload the registers on each clock cycle. The main challenges of clock gating are finding the best places to use it and creating the logic to shut off and turn on the clock at the proper times.

Clock gating is a well known power-saving technique that has been used for years. Synthesis tools such as Power Compiler can detect low-throughput data paths where clock gating can be used with the greatest benefit, and can automatically insert clock-gating cells in the clock paths at the appropriate locations. Clock gating is relatively simple to implement because it only requires a change in the net list. No additional power supplies or power infrastructure changes are required.

4.3 CLOCK-GATING CELLS

Synthesis tools such as Power Compiler can determine where clock gating can be used to provide the greatest power-saving benefit, and can automatically insert clock-gating circuits into the
design to implement the clock-gating functions [50]. Inserting clock-gating circuitry into an existing clock network can introduce skew that adversely affects timing. To have the synthesis tool account for such effects during synthesis, we can have the tool use predefined integrated clock-gating cells, which can be provided as logic cells in the library. An integrated clock-gating cell integrates the various combinational and sequential elements of a clock gate into a single library cell. Figure 4.6 shows one possible implementation of an integrated clock-gating cell.

A clock-gating cell can incorporate any kind of logic such as multiple enable inputs, test clock input, global scan input, asynchronous reset latch, active-low enabling logic, or inverted gated clock output. Power Compiler is free to optimize the enabling logic surrounding a clock-gating cell by absorbing the surrounding logic into the logic functions available inside the cell [50].
Power optimization at high levels of abstraction has a significant impact on reduction of power in the final gate-level design. Clock gating is an important high-level technique for reducing the power consumption of a design.

4.4 CLOCK GATING

Clock Gating applies to synchronous load-enable registers, which are groups of flip-flops that share the same clock and synchronous control signals and that are inferred from the same HDL variable. Synchronous control signals include synchronous load-enable, synchronous set, synchronous reset, and synchronous toggle. Register banks disabled during some clock cycles. Typical implementation uses multiplexers. Clock gating cell replaces multiplexers.

![Figure 4.7 Activity reduction using Clock Gating](image)

The registers are implemented by Design Compiler by use of feedback loops. However, these registers maintain the same logic value through multiple cycles and unnecessarily use power. Clock gating saves power by eliminating the unnecessary activity associated with reloading register banks. The figure 4.7 shows activity reduction using Clock Gating.
4.4.1 Benefits of Clock Gating

Dynamic power saving is achieved with low toggle rate on clock pin, internal power of registers is reduced and resulting clock network has less switching activity and consumes less switching power. Area saving is achieved by eliminating multiplexers. Easy to implement as no RTL code change is required, Clock gating is automatically inserted by the tool, and it is technology independent.

Power Compiler allows performing clock gating with the following techniques [50].

- RTL-based clock gate insertion on unmapped registers. Clock gating occurs when the register bank size meets certain minimum width constraints.

- Gate-level clock gate insertion on both unmapped and previously mapped registers. In this case, clock gating is also applied to objects such as IP cores that are already mapped.

- Power-driven gate-level clock gate insertion, which allows for further power optimizations because all aspects of power savings, such as switching activity and the flip-flop types to which the registers are mapped, are considered.

- We can choose the type of clock-gating circuit inserted among the following

- Choose an integrated or nonintegrated cell with latch-based clock gating
Choose an integrated or nonintegrated cell with latch-free clock gating

Insert logic to increase testability

Specify a minimum number of bits below which clock gating is not inserted

Explicitly include signals in clock gating

Explicitly exclude signals from clock gating

Specify a maximum number for the fan-out of each clock-gating element

Move a clock-gated register to another clock-gating cell

Resize the clock-gating element

Without clock gating, Design Compiler implements register banks by using a feedback loop and a multiplexer. When such registers maintain the same value through multiple cycles, they use power unnecessarily.

Figure 4.8 shows a simple register bank implementation using a multiplexer and a feedback loop and synchronous load-enable register with multiplexer when the synchronous load enable signal (EN) is at logic state 0, the register bank is disabled. In this state, the circuit uses the multiplexer to feed the Q output of each storage element in the register bank back to the D input. When the EN signal is at logic state 1, the register is enabled, enabling new values to load at the D input.
Figure 4.8 Synchronous Load-Enable Register With Multiplexer

Such feedback loops can unnecessarily use power. For example, if the same value is reloaded in the register throughout multiple clock cycles (EN equals 0), the register bank and its clock net consume power while values in the register bank do not change. The multiplexer also consumes power.

Latch based Clock gating eliminates the feedback net and multiplexer shown in Figure 4.8 by inserting a 2-input gate in the clock net of the register. Clock gating can insert inverters or buffers to satisfy timing or clock waveform polarity requirements. The 2-input clock gate selectively prevents clock edges, thus preventing the gated-clock signal from clocking the gated register.

Figure 4.9 shows a latch-based clock-gating style using a 2-input AND gate; however, depending on the type of register and the gating style, gating can use NAND, OR, and NOR gates instead.
Figure 4.9 Latch-Based Clock Gating

At the bottom of Figure 4.9 waveforms of the signals are shown with respect to the clock signal, CLK. The clock input to the register bank, ENCLK, is gated on or off by the AND gate. ENL is the enabling signal that controls the gating; it derives from the EN signal on the multiplexer shown in Figure 4.5. The register bank is triggered by the rising edge of the ENCLK signal.

The latch prevents glitches on the EN signal from propagating to the register’s clock pin. When the CLK input of the 2-input AND gate is at logic state 1, any glitching of the EN signal could, without the latch, propagate and corrupt the register clock signal. The latch eliminates this possibility because it blocks signal changes when the clock is at logic state 1.

In latch-based clock gating, the AND gate blocks unnecessary clock pulses by maintaining the clock signal’s value after the trailing edge. For example, for flip-flops inferred by HDL constructs of rising-
edge clocks, the clock gate forces the gated clock to 0 after the falling edge of the clock.

By controlling the clock signal for the register bank, we can eliminate the need for reloading the same value in the register through multiple clock cycles. Clock gating inserts clock-gating circuitry into the register bank’s clock network, creating the control to eliminate unnecessary register activity.

Clock gating reduces the clock network power dissipation, relaxes the datapath timing, and reduces routing congestion by eliminating feedback multiplexer loops. For designs that have large multi-bit registers, clock gating can save power and reduce the number of gates in the design. However, for smaller register banks, the overhead of adding logic to the clock tree might not compare favorably to the power saved by eliminating a few feedback nets and multiplexers.

4.5 CLOCK-GATING CONDITIONS

Before gating the clock signal of a register, Power Compiler checks if certain clock-gating conditions are satisfied. Power Compiler inserts a clock gate only if all clock-gating conditions are satisfied.

Registers in our processor design qualify for clock gating when the following conditions are met:

- The circuit demonstrates synchronous load-enable functionality.
- The circuit satisfies the setup condition.
The register bank or group of register banks satisfies the minimum number of bits we specify with the `set_clock_gating_style -minimum_bitwidth` command. The default value used for the minimum bitwidth is 3.

After clock gating is complete, the status of clock-gating conditions for gated and ungated register banks appears in the clock-gating report.

The register must satisfy all three of the following conditions before Power Compiler gates the clock signal of the registers:

- **Enable condition**
  
  This condition checks if the register bank’s synchronous load-enable signal is constant logic 1, reducible to logic 1, or logic 0. In these cases, the condition is false and the circuit is not gated. If the synchronous load-enable signal is not constant logic 1 or 0, the condition is true and clock gating goes on to check the setup condition. The enable condition is the first condition clock gating checks.

- **Setup condition**
  
  This setup condition applies to latch-free clock gating only. It checks that the enable signal comes from a register that is clocked by the same clock as the register being gated. Clock gating checks this condition only if the register satisfies the enable condition.

- **Width condition**
  
  The width condition is the minimum number of bits for gating registers or groups of registers with equivalent enable signals. The
default value is 3. We can set the width condition by using the -minimum_bitwidth option of the set_clock_gating_style command. Clock gating checks this condition only if the register satisfies the enable condition and the setup condition.

➤ Enable Condition

The enable condition of a register or clock gate is a combinational function of nets in the design. The enable condition of a register represents the states for which a clock signal must be passed to the register. The enable condition of a clock gate corresponds to the states for which a clock is passed to the registers in the fanout of the clock gate. Power Compiler utilizes the enable condition of the registers for clock-gate insertion.

Enable conditions can be hard to identify in the RTL netlist. By setting the power_cg_print_enable_conditions variable to true to report the enable conditions.

4.6 INSERTING CLOCK GATES

Power Compiler inserts clock-gating cells to our design if we compile our design using the -gate_clock option of the compile or compile_ultra command. We can also insert clock gates to our design using the insert_clock_gating command. The following sections discusses in detail these two ways of clock-gate insertion.

Using the compile_ultra -gate_clock Command

During the compilation process, Power Compiler can insert clock-gates to our design if we use the -gate_clock option of the compile or compile_ultra commands. With the -gate_clock option,
compile or compile_ultra commands can perform clock-gate insertion on the gate-level netlist and the RTL netlist as well as GTECH netlist. By default, when we use the -gate_clock option, the tool inserts clock gates only in the same level of hierarchy as the registers gated by the clock gate. For the tool to perform clock gating across the design hierarchy, set the compile_clock_gating_through_hierarchy variable to true. The compile_ultra -gate_clock command can also perform clock gating on design ware components.

In the Design Compiler topographical mode, when we perform clock gating with incremental compile, using the compile_ultra -incremental -gate_clock command, the tool performs incremental placement and gate-level clock-gating.

Using the insert_clock_gating Command

The insert_clock_gating command can be used to perform clock-gating on the GTECH netlist. We cannot use this command to perform clock gating on gate-level netlist. To perform clock gating on a gate-level netlist use the compile_ultra -gate_clock command. This command identifies clock-gating opportunities by combining different register banks that share common enable signal.

The insert_clock_gating command performs clock gating on all the subdesigns in the design hierarchy by processing each subdesign independently. Use the -no_hier option to limit the clock-gate insertion to the top level of the design hierarchy. Use the -global option to perform hierarchical clock gating, that is, to insert clock gates on all
levels of design hierarchy, considering the design as a whole and not considering each subdesign independently.

### 4.7 CLOCK GATING FLOWS IN COMPILER

The various clock-gating flows supported by the tool is discussed in detail in the following sections.

- Inserting clock gates in the RTL Design

To insert clock gating logic in our RTL design and to synthesize the design with the clock-gating logic, follow these steps:

- Read the RTL design.

Use the compile_ultra -gate_clock command to compile our design. During the compilation process clock gate is inserted on the registers qualified for clock-gating. By default, during the clock-gate insertion the compile_ultra command uses the default values of the set_clock_gating_style command and also honors the setup, hold, and other constraints specified in the technology libraries. To override the setup and hold values specified in the technology library, use the set_clock_gating_style command before compiling our design.

We can also use the insert_clock_gating command to insert the clock-gating cells.

Both, compile_ultra and insert_clock_gating commands use the default values of the clock-gating style during the clock gate insertion. The default values of the set_clock_gating_style command is suitable for most designs.
If we are using testability in our design, use the `insert_dft` command to connect the scan_enable and the test_mode ports or pins of the integrated clock-gating cells.

we have used the `report_clock_gating` command to report the registers and the clock gating cells in the design. Use the `report_power` command to get details of the dynamic power utilized by our design after the clock gate insertion.

In the following Processor Design, clock gating is implemented in the design during the compilation process. The default values of the `set_clock_gating_style` command are used during the clock-gate insertion. The `-scan` option of the `compileUltra` command enables the examination of our design for scan insertion for mission mode constraints.

dc_shell> read_verilog processor.v

dc_shell> create_clock -period 10 -name CLK

dc_shell> compile_ultra -gate_clock -scan

dc_shell> insert_dft

dc_shell> report_clock_gating

dc_shell> report_power

**4.7.1 INSERTING CLOCK GATES IN GATE-LEVEL DESIGN**

To insert clock gating logic in our gate-level netlist and to re-synthesize the design with the clock gating logic follow these steps:

- Read the gate-level netlist.

we have used the `compileUltra -gate_clock` command to compile our design. During the compilation process, clock-gating cells
are inserted on the registers qualified for clock-gating. During this process by default, the compile_ultra command

- Reads the setup and hold constraints that are specified in the technology libraries
- Propagates these constraints up the hierarchy.

To override the setup and hold values specified in the technology library, use the set_clock_gating_style before compiling our design. Using the compile_ultra -gate_clock command we can perform clock-gate insertion on designware elements as well.

The compile_ultra -gate_clock command uses the default values of the clock gating style during the clock-gate insertion. The default values of the set_clock_gating_style command are suitable for most designs. For more details on the default clock-gating style.

If we are using testability in our design, use the insert_dft command to connect the scan_enable and test_mode ports or pins of the integrated clock-gating cells.

Use the report_clock_gating command to report the registers and the clock gating cells in the design. Use the report_power command to get details of the dynamic power utilized by our design after the clock gate insertion.

dc_shell> read_ddc design.ddc
dc_shell> compile_ultra -inc -gate_clock -scan
dc_shell> insert_dft
dc_shell> report_clock_gating
dc_shell> report_power
Table 4.3  Clock-Gating Technique Comparison

<table>
<thead>
<tr>
<th></th>
<th>RTL</th>
<th>Gate-level</th>
<th>Power-driven</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input Netlist</td>
<td>RTL</td>
<td>Gate-level</td>
<td>Gate-level</td>
</tr>
<tr>
<td>Honor the set_clock_gating_style command</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes. Using the -minimum_bitwidth option is not recommended</td>
</tr>
<tr>
<td>Cost factor considered during clock-gate insertion</td>
<td>Design topology</td>
<td>Design topology</td>
<td>Dynamic power and switching activity</td>
</tr>
<tr>
<td>Performing clock gate insertion</td>
<td>Use compileUltraGate_clock or insert_clock_gating command</td>
<td>Use the compileUltraGate_clock or the compileUltraIncrementalGate_clock command</td>
<td></td>
</tr>
<tr>
<td>Tasks performed</td>
<td>Clock-gate insertion</td>
<td>Clock-gate insertion</td>
<td>Clock-gate insertion, optimization, and removal</td>
</tr>
<tr>
<td>Additional tasks performed in Design Compiler Topographical Technology</td>
<td>None</td>
<td>Power correlation</td>
<td>Power correlation</td>
</tr>
</tbody>
</table>
4.8 SPECIFYING CLOCK-GATE LATENCY

During synthesis, design compiler assumes that the clocks are ideal. An ideal clock incurs no delay through the clock network. This assumption is made because real clock-network delays are not known until after clock tree synthesis. In reality clocks are not ideal and there is a non-zero delay through the clock network. For designs with clock gating, the clock-network delay at the registers is different from the clock-network delay at the clock-gating cell. This difference in the clock-network delay at the registers and at the clock-gating cell results in stricter constraints for the setup condition at the enable input of the clock-gating cell.

For design compiler to account for the clock network delays during the timing calculation, specify the clock network latency using either the set_clock_gate_latency or the set_clock_latency command. The set_clock_gate_latency command can be used for both, gate-level and RTL designs. The set_clock_latency command can be used only on RTL netlist. More details of these two commands are described in the following sections.

- The set_clock_gate_latency Command

When we use the compile_ultra_gate_clock command, clock gates are inserted during the compilation process. To specify the clock network latency, even before the clock-gating cells are inserted by the tool, use the set_clock_gate_latency command. This command lets us specify the clock network latency for the clock-gating cells as a function of the clock domain, clock gating stage, and the fanout of the
clock-gating cell. The latency that we specify is annotated on the clock gating cells when they are inserted by the compile_ultra -gate_clock command.

- Applying Clock-Gate Latency

  The clock latency specified using the set_clock_gate_latency command is annotated on the registers during the compile_ultra -gate_clock command when the clock-gating cells are inserted. However, if we modify the latency values on the clock gates after the compilation, we must manually apply the latency values on the existing clock-gating cells using the apply_clock_gate_latency command. This command can be used on the clock-gating cells inserted by the tool during the compile_ultra -gate_clock command or by the insert_clock_gating command.

4.9 SPECIFYING SETUP AND HOLD

  During insertion of clock gates, the setup and hold time that we specify defines the margins within which the enable signal (EN) must operate to maintain the integrity of the gated-clock signal.

  The setup and hold values for the integrated clock-gating cell are specified in the technology library. The values specified in the technology library are honored by compile_ultra -gate_clock command during clock gate insertion. However, we can override these values in the following ways:

  Specifying the -setup and -hold options in set_clock_gating_style command. By doing so, all the clock gates in the design should have the setup and hold time that we specify.
For the clock-gating cells already existing in our design, use the set_clock_gating_check command to specify a desired setup and hold time. We cannot use this command if the clock gates are inserted during the compile_ultra -gate_clock command.

We use the report_timing -to command to the enable pin of the clock-gating cell to verify that the new values are correct.

In the design we have used the set_clock_gating_style command to specify the setup and hold values:

```plaintext
set_clock_gating_style \
    -max_fanout 16 \
    -positive_edge_logic integrated \
    -setup 6 \n    -hold 2
compile_ultra -gate_clock
# to validate the user-specified setup/hold time for
# integrated clock gating
report_timing -to clk_gate_out_top_reg/EN
    using set_clock_gating_check:
    set_clock_gating_style \
        -max_fanout 16 \
        -positive_edge_logic integrated \
        -control_point before \n        -control_signal test_mode
set_clock_gating_check -setup 3 -hold 2 [ get_cells clk_gate_out_top_reg/main_gate ]
```
set_clock_gating_check -setup 5 -hold 1.5 [ get_cells
clk_gate_out_top_reg_1/main_gate ]

compile_ultra -gate_clock
# to validate the user-specified setup/hold time for
# integrated clock gating
report_timing -to clk_gate_out_top_reg/EN report_timing
    -to clk_gate_out_top_reg_1/EN

The clock gate must not alter the waveform of the clock, other
than turning the clock signal on and off. If the enable signal operates
outside the properly chosen margins specified by -setup and -hold, the
resulting gated signal can be clipped or otherwise corrupted.

4.10 CHOOSING GATING LOGIC

The following options of the set_clock_gating_style command
specify the type of clock-gating logic or clock-gating cell used for
implementing clock gating:

- positive_edge_logic [gate_list] [cell_list]
- negative_edge_logic [gate_list] [cell_list]

We can specify a configuration of 1- and 2-input gates (simple
gating cells) to use for clock gating, or an integrated clock-gating cell
already defined in the target library. An integrated cell is a dedicated
clock-gating cell that combines all of the simple gating logic of a clock
gate into one fully characterized cell, possibly with additional logic
such as multiple enable inputs, active-low enabling logic, or an
inverted gated clock output.

- Choosing a configuration for gating logic
The `positive_edge_logic` and `negative_edge_logic` options can have up to three string parameters that specify the type of clock gating logic:

The type of 2-input clock gate (AND, NAND, OR, NOR)

An inverter or buffer on the clock network before the 2-input clock gate. An inverter or buffer on the clock network after the 2-input clock gate

The positions of the string parameters determine whether clock gating places a buffer or inverter before or after the 2-input clock gate. For example, if the value of `positive_edge_logic` is `{and buf}`, clock gating uses an AND gate and places a buffer in the fanout from the AND gate. If the value is `{inv nor}`, clock gating uses a NOR gate and places an inverter in the fan-in of the NOR gate. Both of these examples result in AND functionality of the clock gate.

The type of logic that is appropriate for gating our circuit depends on:

- Whether the gated register banks are inferred by rising- or falling-edge clock constructs in our HDL code
- Whether we use latch-based or latch-free clock gating
- When using latch-free clock gating, we must specify both the `positive_edge_logic` and `negative_edge_logic` options.

### 4.11 Selecting Clock-Gating Style

Apart from the standard style, in which one clock-gating cell is applied per register bank, clock gating can also be implemented in different modes, depending on the design architecture, to gain more
dynamic power savings. Figure 4.10 shows three different clock gating styles: general, multi-stage, and hierarchical.

In multi-stage clock gating, further gating is done for the clock-gating cells which have common enable signals. As a result, more dynamic power can be saved. In the hierarchical clock-gating style, clock gating is done at the logic hierarchy level by gating hierarchical logic modules that share a common enable and the same clock group. This reduces the number of clock-gating cells in the design. Power Compiler provides extensive features and control mechanisms that implement these clock-gating styles. The set_clock_gating_style command has options to specify clock-gating parameters such as integrated clock-gating cell usage, register bank width, and fanout. We can use the -gate_clock option of the compile or compile_ultra command to perform clock-gate insertion during compile. In that case, Power Compiler can perform clock-gate insertion on gate-level netlists as well as RTL and GTECH netlists.
In a multi-voltage design, clock tree synthesis is performed separately for each voltage domain because the cell and net delays can be different for different domains. Design Compiler adds any necessary level shifters on the clock and clock-enable nets that cross voltage domain boundaries. Level shifter insertion on clock nets can be controlled by setting the auto_insert_level_shifters_on_clocks variable.

Use the set_clock_gating_style command to select the clock-gating style. The compile_ultra_gate_clock and the insert_clock_gating commands use the specified clock-gating style to insert the clock-gating cells. The default value of the set_clock_gating_style command is suitable for most designs. If the default setting does not suit our design, use this command to change the default setting.
The clock-gating style that we specify is applied to the entire design, by default. We can also apply the clock-gating style only to specific power domains or hierarchical cells of the design. Using the set_clock_gating_style command we can,

- Specify the conditions when clock gating should be applied
- Specify a latch-based or latch-free clock-gating style (the default is latch-based, with or without an integrated cell)
- Assign values for setup and hold times at the enable input of the clock-gating cell. The default is 0.
- Specify the test logic to be added during clock gating to improve controllability and observability.

The set_clock_gating_style command has the following syntax:

```
set_clock_gating_style

[-sequential_cell none | latch]
[-minimum_bitwidth int]
[-setup sh_value]
[-hold sh_value]
    [-positive_edge_logic {cell_list | \integrated [active_low_enable][invert_gclk]}]
    [-negative_edge_logic {cell_list | \integrated [active_low_enable][invert_gclk]}]
[-control_point before | after]
[-control_signal scan_enable | test_mode]
[-observation_point true | false]
```
[-observation_logic_depth int]
[-max_fanout int]
[-num_stages int]
[-no_sharing]
[-instances instances]
[-power_domains power_domains]

The following sections describe how to use the set_clock_gating_style command:

4.12 CHOOSING A SPECIFIC LATCH AND LIBRARY

The -sequential_cell option allows us to use a specific latch when inserting clock-gating circuitry. To use a specific latch from the target library, specify the name of the latch after the element type, separating the two with a colon (:). For example:

-sequential_cell latch:LAH10

To designate a specific latch from a specific target technology library, insert the name of the technology library as shown in the following example. Clock gating uses a latch called LAH10 from the target library.

-sequential_cell latch:SPECIFIC_TECHLIB/LAH10

Here clock gating uses the LAH10 latch from a technology library called SPECIFIC_TECHLIB. We must previously have specified this technology library file name when setting the Design Compiler target_library variable.

    target_library = { "TSMC.db" "SPECIFIC_TECHLIB.db" }
By convention, the technology library name and file name are usually different. For example, CMOS8_max is the name of a technology library. The file name for it can be any name.lib. The .lib extension means the library is in Liberty text format. Here, TSMC.lib is the file name for this library in text format. CMOS8_MAX.db is the file name for this library in Synopsys proprietary binary format.

Clock gating introduces multiple clock domains in the design. Introducing multiple clock domains can affect the testability of our design unless we add logic to enhance testability.

In certain scan register styles, a gated register cannot be included in a scan chain, because gating the register’s clock makes it uncontrollable for test (assuming there is no dedicated scan clock). Without the register in the scan chain, test controllability is reduced at the register output and test observability is reduced at the register input. If we have many gated registers, this can significantly reduce the fault coverage in our design.

We can improve the testability of our circuit by using the options of the set_clock_gating_style command to determine the amount and type of testability logic added during clock gating. We can perform the following steps to improve testability:

- Add a control point for testing
- Choose test_mode or scan_enable
- Add observability logic
4.13 KEEPING CLOCK-GATING INFORMATION IN A STRUCTURAL NETLIST

Power Compiler applies several clock-gating attributes to the design and to the clock-gating cells and gated registers in the design. Commands such as report_clock_gating, rewire_clock_gating, remove_clock_gating and several placement optimization algorithms depend on these attributes for proper operation.

The power_cg_flatten variable specifies whether to flatten the clock-gating cells when we use commands that perform ungrouping, such as ungroup, compile -ungroup_all, or balance_registers. By default, the variable is set to false and the clock-gating cells are not flattened. This is recommended for most situations because ungrouping the clock gates could cause problems.

Ungrouped clock gates cannot be mapped to integrated clock gating cells. Power Compiler commands, such as report_clock_gating, remove_clock_gating, and rewire_clock_gating, require the original clock-gating hierarchy. Flattened clock gates are supported when we use integrated clock-gating cells, as long as the flattening is done only after executing the compile command.

We can write a structural netlist in ASCII format after clock-gate insertion, synthesis, or placement. Reading back this structural netlist causes the clock-gating attributes to be lost, possibly preventing clock-gating and optimization from operating properly.

If we have used the compile_ultra -gate_clock command to insert clock-gating cells, the tool can automatically retrieve the clock-
gating attributes and identify the clock-gating cells when we read back the ASCII netlist.

### 4.14 CLOCK-GATE OPTIMIZATION PERFORMED DURING COMPILATION

Compile performs logic synthesis on the design, taking into consideration the multiple voltages and multiple power domains defined by UPF commands. Compile makes sure that the logic inside each power domain is mapped to technology cells from the correct voltage corner. In Design Compiler topographical mode, all multi voltage features are completely supported.

In topographical mode, the create_voltage_area command creates a voltage area for placing the cells associated with a particular power domain, like the same command in IC Compiler. This command completes the physical constraints linked to the multi voltage design that can be defined in design compiler topographical mode. Note that compile_ultra, the default synthesis command in design compiler topographical mode, automatically ungroups all design hierarchies to achieve better timing results. However, the logic hierarchies associated with power domains are not ungrouped and the power domain boundary information is therefore preserved. Apart from the regular optimizations, compile also takes care of specific optimizations and synthesis operations relating to level shifter, isolation, and retention cells.

Compile automatically inserts buffer-type level shifters on the appropriate nets and purges all redundant level shifters. By default, it
does not insert level shifters on clock nets. To change this behavior, set the auto_insert_level_shifters_on_clocks variable to a list of clock names or to all before running compile. The size_only attribute specified on level shifter and isolation cells allows compile to perform sizing optimization on these special cells.

Retention register inference and control port hookup operations are performed by compile. Compile maps user-specified registers in the shutdown power domain with retention registers. Compile synthesizes applicable registers with retention registers and hooks up the save and restore pins to control ports. Library retention cells should have specific attributes at cell and pin level to identify the retention cell and the “save” and “restore” control pins. If we want to insert power management cells outside of a compile operation, we can do so with the insert_mv_cells command. This command inserts isolation, level-shifter, and enable level-shifter cells based on the UPF definition, without doing a full compile.

To further increase the power saving of our design, Power Compiler uses certain techniques during compilation to reduce the number of clock-gating cells in the design. Some of these techniques are multistage clock-gating, hierarchical clock gating. These techniques are described in detail in the following sections.

4.15 MULTISTAGE CLOCK GATING

When a clock-gating cell is driving another or a row of clock-gating cells, this is referred to as multistage clock gating. Power Compiler can identify common enables and factoring using another
clock-gating cell as shown in Figure 4.11. The tool can apply multistage clock gating not only on RTL designs but also on designs that contain gate cells, for further optimizations if available.

The multistage clock-gating feature allows us to combine as many register banks as possible so that the clock gating can be moved up closer to the top, leading to more power savings. As a result, the actual benefits are seen when combined with placement.

For the tool to perform multistage clock gating, we should set the maximum number of stages for multistage clock gating using the `num_stages` option of the `set_clock_gating_style` command.

![Figure 4.11 Multistage Clock Gating](image)

**4.16 HIERARCHICAL CLOCK GATING**

Generally clock gating techniques in Power Compiler extracts common enable conditions that are shared across the registers within the same block. In hierarchical clock gating, Power Compiler extracts
the common enables shared across registers in different levels of hierarchy in the design, during the clock-gate insertion. This technique looks for globally shared enables while inserting clock gating cells. This increases the clock-gating opportunity and also reduces the number of clock-gate insertion. With this technique and proper placement, more power savings can be obtained. Power Compiler inserts hierarchical clock-gating cells at various levels of design hierarchy. As a result, additional ports are created for the clock-gated enable signal as shown in Figure 4.12. These additional ports are added to the sub designs. Formality verifies the designs successfully as long as the designs being compared have the same number of primary ports. Power Compiler can perform hierarchical clock gating on RTL netlists as well as gate-level netlists. Use the compile_ultra -gate_clock or the insert_clock_gating -global command to perform hierarchical clock gating. We use the compile_ultra -gate_clock command to perform hierarchical clock-gating on both RTL and gate-level netlists. This command is especially useful for clock gating on gate-level netlists. To perform hierarchical clock gating using the compile_ultra -gate_ clock command, we must set the compile_clock_gating_through _hierarchy variable to true before compiling our design.
4.17 REPORTING COMMAND FOR CLOCK GATES AND CLOCK TREE POWER

The report_clock_gating command reports the clock-gating cells, the gated and the ungated registers in the current design. To see the dynamic power savings because of clock gate insertion, use the report_power command before and after the clock-gate insertion.

The top portion of the report indicates the name of each register, the clock-gating conditions the flip-flop satisfies, and whether or not the flip-flop’s clock was gated. The double question mark (??) indicates that the condition was not checked during clock gating because a previously checked clock-gating condition was not satisfied. All conditions must be satisfied to gate the clock, unless we use the set_clock_gating_registers command.
4.18 CLOCK-GATING REPORT USING DEFAULT SETTINGS

The gated group column contains arbitrary names for groups of register banks that have equivalent enable signals. Power Compiler creates the group names during clock gating and uses one clock gate to gate the register banks in each group.

Once the processor is optimized by Power Compiler, the optimized net list generated by the Power Compiler is given as input to design compiler and once again the functionality and timing is verified. The simulation results and Synthesis Report is discussed in the next chapter.