ABSTRACT

As the transistor count and operating frequencies of graphics chips increase, power consumption has become a critical problem in the design of these components. Often the location of these components on a system chassis makes it difficult to provide enough airflow to cool the parts or to mount a cooling device. So, the power consumption above certain limit is a critical problem for these components.

Significant power savings can be achieved in a chip by turning off portions of logic whose output may not be needed all the time. Shutting off clocks to unneeded sections of a design can do this. Determining when a section of the design can be shut off requires detailed analysis of the design. The Power Compiler can automate this process for certain types of logic structures.

This paper presents the use of an automated tool to implement clock gating for reducing power consumption in a design. The Power Compiler (v1998.02) from Synopsys was used to automatically insert clock-gating circuitry in the design. In this paper we discuss how we analyzed the risk factors and implemented the changes in design methodology to incorporate a clock gating methodology for power optimization.
1.0 Introduction:

Demand for high performance graphics in personal computers is driving the need to provide increasing levels of performance at an affordable cost. Increased complexity, larger transistor count and higher operating frequencies deliver improved performance but come at a cost of higher power consumption. Dissipation of this power poses significant design challenges.

Since graphics chips in personal computer market have low selling prices, even a small increase in the cost of cooling results in significant increase in the total system cost. Often the location of these components on system chassis makes it difficult to provide enough airflow to cool the parts or to mount cooling devices. Also, the need to prolong battery life of mobile products poses additional demand to reduce power dissipated in designs. So, minimizing power consumption is a critical problem for these products [1].

Traditionally, graphics designs have used several power saving techniques but power problem has not been addressed as aggressively as other design specifications like performance, area, and schedule.

Clock gating is one of the most effective and widely known techniques to reduce power. However, extensive clock gating is not commonly used due to complexity of implementation, impact on clock skew, layout area and clock synthesis issues. In many designs, extensive clock gating is not done due to the effort required to identify and implement clock-gating opportunities. These roadblocks were overcome by enhancing the design methodology to reduce the impact of clock gating and automating clock gating identification and implementation task. This paper describes the methodology used in a graphics controller design to automatically identify and implement clock gating using Power Compiler.

In this paper we present the results of our work on clock gating using Synopsys’ Power Compiler tool. First, we analyze power characteristics of a graphics design and identify methods that can reduce power significantly. In subsequent sections we describe how Power Compiler implements clock-gating structures in the design and its limitations. We discuss how we developed our methodology to overcome these limitations. Then we describe our design trade-offs and present data to justify our choices. Finally, we present results to show the effectiveness of our methodology.

2.0 Where is the power going?

In order to understand which sections of a design consume the most power we examined the power consumption characteristics of the Intel740 graphics controller, a predecessor to the current design in development. To get the absolute worst case power numbers a high power test was designed to exercise the chip so that it would consume the most power [2].
For the purpose of this analysis, we identified various sections of the design and their contribution to total power consumption. These sections were broadly classified into embedded RAMs, IO buffers, analog circuits (D-to-A converters and PLLs) and combinational and sequential gates. Since the clocks toggle the sequential circuits continuously during normal operation, we classified the power consumed by sequential cells as clock power. This data is shown in Figure 1.

![Power Consumption Diagram](image)

**Figure 1: Power consumption of various sections of intel740**

As can be seen, clocks consume the most power; 74% of the total power. Furthermore, 71% of clock power is consumed by the sequential cells, which are driven by clocks. These sequential cells represent largest loading on the clock network and switch every clock tick, causing significant power consumption even when the data inputs have low activity factors.

In graphic chips, the sequential cells are mostly used either for register files or as temporary data storage in the pipelines. These pipelines are continuously clocked even though they may not always be used for computing valid results. Thus, turning off the clocks to sequential elements when their outputs are not needed can result in significant power savings.

To understand what sections of the design contribute most to total power we analyzed activity factors of all non-clock nodes. Activity factor of a node is the number of times data value toggles in a clock cycle. It is a useful indicator for determining how active or busy a portion of the design is. Toggle data was collected on all nodes in the design...
while running power virus test. A histogram in Figure 2 shows the activity factors of data nodes (i.e. non-clock nodes) in the design. It shows that 80% percent of the nodes toggle only 2% of the time. These data nodes do not toggle frequently because either the logic is idle, unused or computing the same operand.

Figure 2: Activity factors of data nodes in intel740

As shown in Figure 2, nearly 80% of non-clock nodes in the chip are performing useful computations only 2% of the time. Theoretically, it should be possible to turn off clock to units that contain these nodes when they are not doing useful calculations thereby reducing wasted power. Practically, the 2% of useful computations are distributed in such a way that not all units can be turned off at the same time. To maximize the opportunities for turning off as much of the design as possible it is desirable to have clock-gating logic to turn off units or sub-units within these units [3].

3.0 Clock Gating for Power Reduction:

As mentioned earlier, the clock network is the largest contributor to total power consumption. A practical way to achieve power savings is to shut off those portions of the design that are not needed during an operation. This is accomplished by clock gating. It is implemented by qualifying a clock by an enable signal. A regular clock buffer can be changed into a qualifying clock gate with low area and performance
overhead. There are two common methods for implementing clock gating. These are described in following sections.

3.1 Central Clock Gating:

In this scheme all the clocks are generated in a central clock unit. Additional logic is included to gate these clocks based on certain conditions. The output of central clock unit is a set of gated clock signals that fan out to various units. The design is partitioned into separate units that are fed by separate clocks that are independently controlled. Figure 3 shows this scheme.

Figure 3: Central Clock Gating

The central clock unit method has been used successfully in many designs. The popularity of this method is mainly due to the ease of its implementation. There is no modification required in the HDL code, synthesis or timing analysis flow. The central clock unit is designed as a separate unit and other unit designers do not have to worry about clocking issues. However, there are some disadvantages of this method.

- All clock-gating elements are in the central clock unit and gated clocks are routed to destination logic sections. This requires more routing tracks to be used for clock network.
- This method is not very practical if the number of clocks is large, say few hundred clocks. Thus, this method is applicable only to simple designs in which the unit clocks are derived by gating the main clock.
3.2 Distributed Clock Gating:

The distributed clock gating method does not have a central clock-gating unit. All clocks from the main clock unit are distributed to various units or sections of the chip where they are gated at or near the destination logic. This allows the designers to minimize the routing tracks dedicated to clocks. Also, multiple levels of clock gating on selected sections of logic within a unit can be done easily. This scheme is shown in Figure 4. This scheme requires every unit designer to analyze the local clock gating logic.

![Figure 4: Distributed Clock Gating](image)

Distributed clock gating scheme is made possible by enhancing the design flow to handle clock gating circuits within units without impacting layout, timing analysis and clock skew.

4.0 Distributed Clock Gating Plan:

In our design, we chose to implement a distributed clock-gating scheme. This type of clock-gating scheme is more difficult to implement as it affects the design of individual units. In order to manage the complexity of clock synthesis and maximize power savings, we decided to implement two levels of clock gating in our design.
• **Unit Level Clock Gating:**
  The designer of each unit specified *unit idle condition* that identifies when the said unit can be turned off. A clock entering this unit was gated for this *unit idle condition*.

• **Flop Level Clock Gating:**
  At lower level, synchronous enabled flip flops, also called re-circulating flip flops, were converted to gated clock flip flops.

Some portions of logic in each unit are required to be active all the time for functional reasons. Therefore, there is a need to provide un-gated clocks to this logic. We implemented unit level clock gating by providing both gated and un-gated clocks to every unit. Clock gate macro cells were instantiated in VHDL to gate clocks. We were concerned about the difference in delays on gated and un-gated clocks due to clock gate macro. To balance this, we instantiated a “dummy” clock gate macro on un-gated clock paths and disabled its gating logic. This is shown in Figure 5.

![Figure 5: Insertion of Dummy Clock Gating cell for balancing the delays](image)

The decision to enable or disable a clock gate for a flip-flop is made after a detailed analysis of the design and it’s various modes of operation. This is a complex task and represents significant effort on behalf of the designer. There are a large number of re-circulating flip-flops in the design. Thus analysis of gating conditions for all re-circulating flip-flops to determine their suitability for flop-level clock gating is a
monumental task. The Power Compiler can do this analysis automatically. So we chose to use Power Compiler for flop-level clock gating.

5.0 Power Compiler’s Clock Gating

In this section we briefly describe the automatic clock-gating feature of Synopsys’ Power Compiler. Detailed description of Power Compiler’s algorithm for clock gating is available in [4].

The Power Compiler looks for flip-flops with multiplexers on inputs (also called re-circulating flip-flops or synchronous enabled flip-flops) and replaces them with a simple flop and a clock gate. The Power Compiler analyzes the design to determine conditions for clock gating and creates logic that drives the enable signal for clock gate.

The user can choose between using a simple AND gate for controlling a clock versus a combination of a latch and gate. Using simple AND gate structure can cause glitches in the clock path and is, therefore, not preferred. The latch structure guarantees that the design is glitch free. We chose to use the latch structure for our designs. The clock gating circuit is shown in Figure 6.

![Figure 6: Clock Gating by Power Compiler](image)

5.1 Limitations of Power Compiler:

During the development of our design methodology we found several limitations of Power Compiler. Here we briefly discuss these limitations and then describe how we worked around these issues.

- Power Compiler inserts a clock gating sub-module that consists of discrete cells for clock gating (a latch, an AND and an OR gate). There is a possibility that clock
gating AND gate and enable latch may be placed far apart during layout resulting in a significant clock skew between the AND gate and the clock pin of the enable latch. This could create timing problems that may be very difficult to address.

- Power Compiler inserts clock gates during elaboration time. Hence it does not understand the total loading on the gated clock net. Sometimes these nets could get very heavily loaded (i.e. a very large fan out). This may cause clock skew problems in the design.

- Power Compiler inserts a separate gating element for each vector. For example, even if vectors A[3:0] and B[7:0] are controlled by same enable signal, Power Compiler creates two separate clock-gating elements. This could make the tool use more clock gate elements than is necessary thus incurring area penalty.

5.2 Solutions for Power Compiler limitations:

The limitations of Power Compiler, mentioned in previous sections, were addressed by developing dc_shell scripts and post processing Perl programs. In following sections we describe the working of our solutions.

- We created a new macro cell in our design library that implements the clock-gate function shown in Figure 6 previously. This cell was so designed that it is always treated as one complex macro in layout. This guarantees that the net between latch and gate will always be fixed in length thus eliminating possibility of glitch. To use this cell, we wrote a post processor that replaced the clock gate sub-module inferred by Power Compiler with this macro cell.

- We created a post processing script that analyzes enable signals of all clock gates in a design. If the enables are functionally equivalent they are merged and the same clock-gating element is used for both sets of flip-flops. At this time we also size the clock gate macro based on the number of fan outs.

The Power Compiler allows the user to control the clock gating by defining a minimum width below which a bank of flip-flops (i.e. a vector) will not be gated. If every flip-flop that can potentially be gated is actually gated we may incur a high area penalty due to the large number of clock gating macros in netlist. To minimize this cost, we analyzed our design for clock gating opportunities. Our design has 26304 flip-flops, of which 16175 are re-circulating flip-flops and can be gated. As mentioned earlier, we created a script that allowed us to merge the clock-gating logic for those flip-flops that had common enable signals. Table 1 shows the cost of clock gating for various vector widths with and without merging common enable logic.

<table>
<thead>
<tr>
<th>Bank Width</th>
<th>Clock Gate Cells</th>
<th>% Flops Covered</th>
</tr>
</thead>
</table>

SNUG 1999 9 Automatic Clock Gating for Power Reduction
The cost of gating every flop individually is significantly higher than when 8 or more flip-flops share a common clock gate. Also, the data clearly shows that higher coverage is achieved when flops with common enable logic are merged. Based on this data we decided to gate banks of width 8 or more and merge those banks that have common enable logic.

### 6.0 Power Savings:

The effectiveness of our power saving methodology was determined by estimating power using Design Power. Toggle data, from a typical application, was annotated to post layout netlist with parasitics. With all clock-gating features disabled, we determined the base line power. Then we enabled unit-level clock gating and estimated power. Finally we turned on flop level of clock gating and computed power again.

For this typical application, a 22% power reduction in synthesized core logic was observed when unit-level clock gating was enabled. A further 18.6% power reduction was observed when flop-level clock gating was enabled. Thus, a total saving of up to 40.6% was observed in synthesized core logic of the chip when both unit- and flop-level clock gating was enabled.

### 7.0 Conclusions:

Management of power in modern graphics chips is an important design requirement. Significant power reduction is possible by selectively switching off power to unused units of a design. However, the complexity of the task requires an automated solution.

An effective methodology for reducing power in a large graphics controller has been shown to successfully reduce power of synthesized core logic up to 40%. This power saving was made possible by two levels of clock gating; block level clock-gating by manual changes to design and flip-flop level clock gating by using Power Compiler.

### 8.0 Acknowledgment:

The authors would like to thank Omar Malik for his leadership in setting the direction and scope of this effort. We would also like to thank Nick Sadowy and Balaji Veeraswamy for their help.
9.0 References: