RTL Power Optimization with Gate-level Accuracy

Qi Wang
Cadence Design Systems, Inc
555 River Oaks Parkway, San Jose 95125 2903
qwang@cadence.com

Sumit Roy
Calypto Design Systems, Inc
Bunker Hill Lane, Suite 208, SantaClara 95054
sroy@calypto.com

ABSTRACT
Traditional RTL power optimization techniques commit transformations at the RTL based on the estimation of area, delay and power. However, because of inadequate power and delay information, the power optimization transformations applied at the RTL may cause unexpected results after synthesis, such as worsened delay or increased power dissipation. Our solution to this problem is to divide RTL power optimization into two steps, namely *RTL exploration* and *gate-level commitment.* During RTL exploration phase potential candidates for applying some specific RTL transformation are identified where high level information permits faster and more effective analysis. These candidates are simply "marked" on the netlist. Then during the gate-level commitment phase when accurate power and delay information is available, the final decision of whether accepting or rejecting the candidate is made to achieve the best power and delay trade-offs.

1. INTRODUCTION
Power consumption has become an increasingly important optimization metric in the design of microelectronic systems. Power optimization can be achieved at different levels within a design cycle, though it is well known that the higher the level of abstraction where power optimization techniques are applied, the higher the potential power savings[1,2].

Clock-gate transformation is probably by far the most commonly used RTL power optimization technique[3]. In many design situations, data is loaded into registers infrequently, but the clock signal continues to switch at every clock cycle, driving a large capacitive load. This makes the clock signal a major source of dynamic power dissipation. Identifying periods of inactivity in the registers and disabling the clock during those periods can save significant amounts of power [1,3]. However, clock gating is restricted to saving power on sequential elements only. Figure 1.1 shows a design where a register bank takes the result of the multiplier only as its input. The power dissipated by the multiplier is wasted during the cycle when the register bank is not loading the data. Clock gating cannot be used for this problem since it only reduces the power of the register bank. One solution is to shut down the multiplier when its outputs are not used, as shown by the two banks of shaded AND gates in the Figure 1.2. This technique is known as operand isolation [5] or sleep-mode optimization.

In Figure 1.2, the multiplier is called a sleep-mode candidate; the signal *e* is called the enable signal for the sleep-mode candidate; and the added logic of AND gates for turning off the multiplier are called the sleep-mode logic. Sleep-mode transformation of a given design involves the tasks of identifying the set of sleep-mode candidates and the corresponding enables, and making a decision on whether inserting the sleep-mode logic or not so that power can be optimized under some constraints such as delay or area.

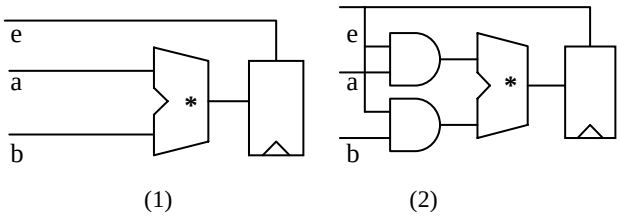

Figure 1. A design (1) without sleep-mode (2) with sleep-mode transformation.

2. Related Work
In [4], a control-signal gating technique is proposed to reduce the switching activities on datapath buses. The idea is to use observability don't care conditions of buses to gate control signals going to the bus driver modules so that the switching activity on the module inputs does not propagate to the bus. Unlike the sleep-mode transformation, the proposed method does not work for datapath buses or modules which are not driven by registers. The other problem of the method is that it computes observability don't care based on the structural netlist of a design, which may be very expensive or even prohibitive on real designs with intensive datapath elements.

To the best of our knowledge, work of [5] presents the first comprehensive approach that automates sleep-mode (operand isolation) on RTL netlist. Unlike the work in [4], this paper presents a complicated RTL power model to estimate the power savings by introducing the sleep-mode logic. Additionally, rather than using an existing signal for the enable signal for the sleep-mode logic, a new function is used and computed by structural analysis of the transitive fanout of a module.

The work of [6] is different from above introduced techniques in that it achieves power saving by redesigning control logic to reconfigure the existing datapath components under idle conditions. The drawback of this technique is that it works well on control-flow intensive designs but not on data-flow intensive designs. Additionally, the technique may significantly change the final netlist to reduce power. This may not be favored in real industrial designs when power is a secondary optimization target as compared to delay in most design situations.

All above solutions suffer from a common problem: committing an optimization move at the RTL in hopes of saving power after logic and physical optimization. It is well known that power estimations at RTL can significantly differ from the real chip power or even gate level power estimations [2,3]. Since there is no accurate power and timing information available at RTL to perform the correct trade-off, these solutions may end up

ICCAD'03, November 11-13, 2003, San Jose, California, USA.

increasing the actual power or delay of the synthesized netlist. Even if some provides capability to undo the transformations at the end of the synthesis run[5], the optimizer has already spent a lot of effort optimizing the wrong critical paths which are mostly not reversible. This may cause extra long run time or, design with worse power and timing compared to the one without applying the RTL transformation.

In this paper, we present a novel solution to solve the above problem. This unique approach consists of two steps, namely *RTL exploration* and *gate-level commitment*. First, during RTL exploration, we use the Control Data Flow Graph (CDFG) generated from the RTL description of a design to identify all the sleep mode transformation candidates, including the enable signals for each candidate. The enable signal is computed by performing behavioral level observability analysis (BLOA) on CDFG, as to be described in Section 3. Since CDFG contains all high-level information (control and data flow), we are able to identify all possible sleep-mode transformation opportunities presented in a design. More importantly, since the enable signals are computed directly from the control flow, it eliminates the concern that some enable signals may be synthesized into some other logic during synthesis, thus losing the opportunity of any possible transformations associated with the signal.

After identifying all the candidates, we mark the gate level netlist to "remember" these potential transformations without altering the data path or the control path of the design. Noticeably, because the original data and control paths of the design are maintained, the logic and timing optimization applied later will work on the true critical path and no effort is wasted. After logic and timing optimization has been completed, each "mark" of the potential sleep mode transformation will be evaluated and committed or removed to make sure it saves power without violating timing or area constraints. This is referred as gate-level commitment in the paper. It needs to be pointed out that because of the availability of accurate power and timing information at this stage any decision made at this point reflects the real status of the design. In addition to the improved accuracy, the complexity of the algorithm is significantly reduced since no complicated and error-prone RTL power and delay models are required.

It needs to be noted that we are not proposing a new RTL power transformation. Instead, we are proposing a new approach on how to apply the RTL power transformation known as operand isolation or sleep-mode transformation. With the separation of RTL exploration and gate-level commitment, the proposed approach takes the advantage of both rich control-data-flow information at the RTL and the accurate power and delay information at the gate-level so that the best possible power delay tradeoff can be achieved. Our technique guarantees that a design with sleep-mode transformations inserted has less power and same or better timing compared to a design without the transformations without significant run time overhead on the existing timing optimization flow.

3. Behavioral Observability Analysis

Observability analysis in logic circuits has been widely used in logic synthesis, test generation, and many other ECAD problems [8]. In this section, we extend the idea of observability analysis to the behavioral level description of a digital system, i.e. CDFG. It is referred as behavioral level observability analysis (BLOA) for the rest of the paper.

3.1 Introduction of CDFG

A CDFG is a graphical representation of the behaviors of digital hardware. The graph is generated from hardware behavioral description languages, such as Verilog or VHDL, and serves as an interface to different RTL and architectural synthesis packages. More details about CDFG can be found in [7].

A CDFG consists of nodes and directed edges. The nodes represent operations in the behavioral description. The edges model the transfer of values between the nodes, i.e. the result of one operation (node) is passed to the argument of another one. Every argument of an operation can be seen as an input port of the corresponding node of the CDFG. Every result of an operation can be seen as an output port of the corresponding node of the CDFG. A token is defined as a single data value instance. There are four major types of nodes.

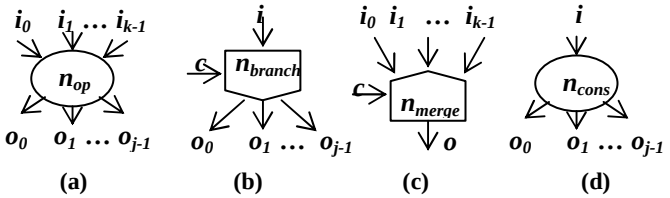

Figure 2. CDFG nodes. (a) Operation node (b) branch node (c) merge node (d) construct node.

a) Operation nodes: Operations can be arithmetic, like +,-,×, ÷, or Boolean, like ∧, ∨, or can be more complex functions or an instantiation of another graph, which models the procedures and functions in the behavioral specification. The graphical representation of an operation node n_{op} with input edges i_0 to i_{k-1} and output edges o_0 to o_{j-1} is shown in Figure 2.a. The node performs the operation defined by n_{op} on the data values of the input edges and transfer the result to all output edges.

b) Branch nodes: A branch node has two input ports, a control port and a data port, and at least one output data port. Based on the value of token on the control port, one and only one output port is selected and the token on the input data port is passed to the selected output port. The graphical representation of a branch node n_{branch} with input data edge i and input control edge c and output edge o_0 to o_{j-1} is shown in Figure 2.b. For example, if the data value on the control edge c is 0, then the data value on the input edge i is transferred to the output edge o_0.

c) Merge nodes: Merge nodes are dual to branch nodes. A merge node has one output data port, one input control port and several input data ports. It passes the token on one input data port, selected by the value of the token on the control port, to the output port. The graphical representation of a merge node n_{merge} with input data edges i_0 to i_{k-1} and input control edge c and output edge o which is shown in Figure 2.c. For example, if the data value on the control edge c is 0, then the data value on the input edge i_0 is transferred to the output edge o.

d) Construct nodes: These are special nodes, which only serve the purpose to generate tokens to control the branch and merge nodes. It has one input data port and at least one output control ports. Only one output port is selected to pass the token from the input port according to the value of the token. The graphical representation of a merge node is shown in Figure 2.d.

Other types of nodes include input, output, register and constant nodes. Every graph requires at least one input node and one output node. Constant nodes are nodes which generate a constant data value at their single output port. Input, output and register nodes correspond to input, output ports and registers of the design represented by the CDFG respectively.

3.2 Observability of CDFG Nodes and Edges

A token in a CDFG is the counterpart of a signal in a logic circuit. In the rest we refer to the behavioral level observability analysis as the *token observability analysis* on CDFG.

Definition 1: **token observable condition** (TOC) of an edge e_{ij}, denoted by TOC(e_{ij}), is the condition under which the token on that edge can be observed at one or more output nodes of a CDFG.

Definition 2: **node observable condition** (NOC) of a node n_i, denoted by NOC(n_i), is the condition under which the token on any output edge of the node can be observed at one or more output nodes of a CDFG.

Given a CDFG, the TOC and NOC for all edges and nodes can be computed by traversing the graph from the output nodes to the input nodes and applying appropriate Boolean operations. Let $\wedge$, $\vee$, and $'$ denote the Boolean AND, OR, inversion operations respectively. For each different type of node, the TOCs of all edges and NOCs of all nodes are computed as follows.

1) Output nodes: By definition, for an output node n_{out} with input edge i,

$$NOC(n_{out}) = 1 , \quad TOC(i) = 1$$

2) Operation nodes: For an operation node n_{op} such as the one in Figure 2.a with input edges of $i_0, \ldots, i_{k-1}$ and output edges of $o_0, \ldots, o_{j-1}$, we have

$$NOC(n_{op}) = TOC(o_0) \vee TOC(o_1) \vee \ldots \vee TOC(o_{j-1})$$
$$TOC(i_p) = NOC(n_{op}), \ \forall p \in \{0, 1, \ldots, k-1\}$$

For the sake of TOC/NOC analysis, input and construct nodes are treated the same as operation nodes.

3) Branch nodes: For a branch node n_{branch} as shown in Figure 2.b, with input data edge of i and input control edge of c and output edges of $o_0, \ldots, o_{j-1}$, we have

$$NOC(n_{branch}) = TOC(o_0) \vee TOC(o_1) \vee \ldots \vee TOC(o_{j-1})$$
$$TOC(c) = NOC(n_{branch})$$
$$TOC(i) = (c_0 \wedge TOC(o_0)) \vee (c_1 \wedge TOC(o_1)) \vee \ldots \vee (c_{j-1} \wedge TOC(o_{j-1}))$$

where c_p, $\forall p \in [0,j-1]$ is a Boolean encoding of the variables for the value of the token at the control port to select output port p. For example, if the token on c is two bit wide ($c= c_1c_0$) and output port 2 is selected, i.e. c is (10) in binary, then the condition to select a port is $c_1 \wedge c_0'$.

4) Merge nodes: For a merge node n_{merge} as shown in Figure 2.c, with input data edges of $i_0, \ldots, i_{k-1}$ and input control edge of c and output edge of o, we have

$$NOC(n_{merge}) = TOC(o)$$
$$TOC(c) = TOC(o)$$
$$TOC(i_p) = c_p \wedge TOC(o), \ \forall p \in [0,k-1]$$

The definition of c_p is the dual to that described for branch nodes, i.e. the condition under which the input port p is selected by n_{merge} to pass the token to the output port.

As in [5], to avoid the computational complexity we assume the NOC of a register node to be **1**. In other words, we do not analysis observabilities across register boundary.

3.3 Computation of TOC/NOC

Computing TOC/NOC requires Boolean operations. A common way to perform Boolean operations is to use BDD (Binary Decision Diagram [9]). The problem of this approach is that BDD is not robust enough to handle large Boolean functions.

In this section, an efficient and yet simple approach to compute TOC/NOC of a CDFG is proposed. The idea of the approach is the separation of traversing CDFG and computing TOC/NOC. Instead of building a BDD for the TOC/NOC of each edge and node, a logic network is constructed whenever a Boolean operation is required to compute the TOC/NOC. The logic network is then optimized using traditional logic optimization techniques such as algebraic transformations [8], to obtain the optimized Boolean function of the TOC/NOC. The transformations used in the optimization process are chosen to be simple enough so that computing TOC/NOC will be almost invisible to the overall optimization flow. This method is ideal for the front-end RTL synthesis tool where a CDFG is translated into a gate level netlist and there exists a one-to-one correspondence between the objects of a CDFG and the objects of the logic network. The method is best explained by an example.

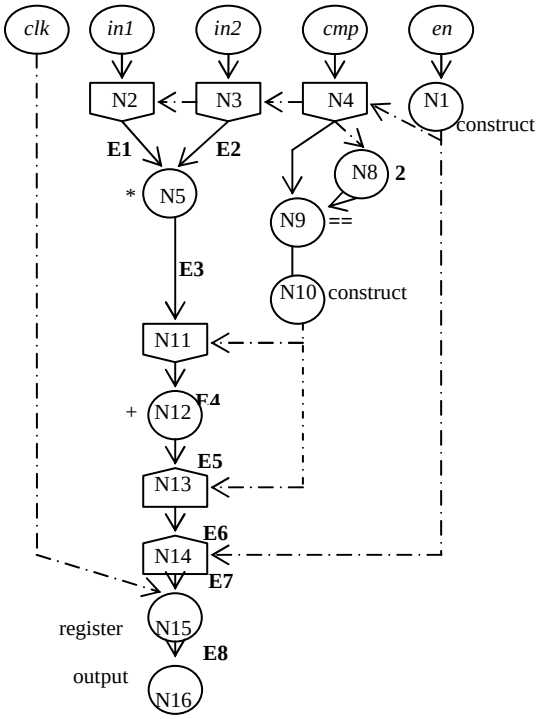

Figure 3. An example CDFG

In Figure 3, the CDFG of a simple digital system is shown. As stated above, after a gate level netlist is generated from this CDFG, there is an one-to-one correspondence between the objects of CDFG and the objects of the generated logic network. For example, let a be the corresponding signal for node N1 and b be the corresponding signal for node N10 in the netlist. We will explain the computation of TOC for edges E1 and E2 in Figure 4. The computation starts from the output nodes using the formulas in the Section 3 depended on the type of the node visited. It is easy to see the TOC of edges E8 and E7 is simply logic **1**, as highlighted in Figure 4. For edge E6, the TOC is the Boolean AND of the TOC of E7 and the variable of the control port of the

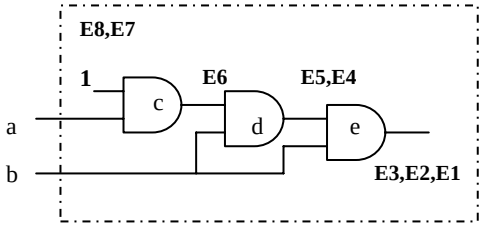

Figure 4. Compute TOC for edges of CDFG in Figure 3.

node N14 which is the same as the variable at the output port of construct node N1. Therefore in Figure 4 the AND gate *c* is constructed whose output represents the TOC of E6. Similarly, the TOC of edge E5 is the AND of TOC of edge E6 and the variable of the control port of node N13. The AND gate *d* is created for the TOC of edge E5. The process continues until the edges E1 and E2 are reached. As shown in Figure 4, the output of the AND gate *e* represents the TOC of edges E2 and E1. After applying simple logic optimizations, the TOC of edges E1, E2 and E3 is simply the AND of *a* and *b*. Note that only a single traversal from the output nodes to the inputs is required to compute the TOCs of these edges. The complexity of the algorithm is O(|V|+|E|) where V and E are the number of nodes and edges of a CDFG respectively. In our experience, the number of nodes and edges of a CDFG is much less than those of the corresponding structural netlist. As a result, the proposed technique for TOC/NOC computation is extremely fast even for very big state-of-art digital designs.

4. Proposed Sleep-mode Transformations

As described earlier, the proposed sleep-mode transformation consists of two steps, namely RTL exploration and gate-level commitment.

4.1 RTL Exploration

The application of BLOA techniques introduced in Section 3 to sleep-mode transformation is straightforward. The corresponding logic of an operation node is a perfect candidate for sleep-mode transformation where the computed TOC/NOC can be used to generate the corresponding enable signal. Nodes with TOC equal to 1 can not be candidates for sleep mode transformation because the outputs of the corresponding logic block are always observable. During the exploration phase, a complete BLOA is performed on the CDFG. Operation nodes with NOC not equal to 1 are selected as candidates. These candidates are then "marked" on the gate-level netlist. No actual implementation is performed in this phase. Commitment of these transformations is delayed after gate-level optimization has been done where accurate power and timing information is available.

For example, BLOA analysis of the design shown in Figure 3 finds that under the condition of *a*& *b* the outputs of the multiplier (node N5) are not used, where *a* and *b* are the signals in the netlist corresponding to the condition represented by construct node N1 and N10 respectively. The netlist of the design after RTL exploration is shown in Figure 5. Compared to the normal netlist after RTL synthesis, the RTL sleep-mode explored netlist has two new inserted modules called SleepModeModule, one in front of each multiplier input. The inserted modules have following characteristics. First the data bus is a feed-through within the module. Second, within the SleepModeModule,, there is another module called SleepModeControlModule. It contains the complex

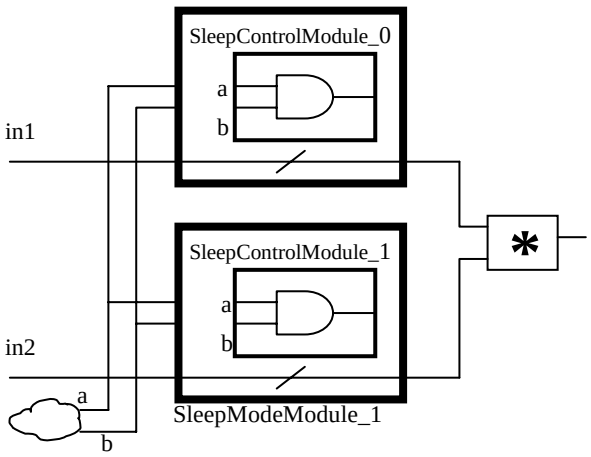

Figure 5. Netlist after RTL sleep-mode exploration for the multiplier of the example in Figure 3.

enable function for the sleep-mode candidate. In this case, it is the AND function of *a* and *b*. There exists a one-to-one correspondence between the input pins of the SleepModeControlModule and the input ports of the SleepModeModule, but they are not connected.

The SleepModeModule serves as the "mark" for the candidate of sleep-mode transformation, which is the multiplier in this case. Note that even though the multiplier hierarchy may be dissolved during synthesis, the "mark" for the sleep-mode candidate will be preserved, because the SleepModeModule is not dissolved, which could be easily done. The SleepModeControlModule serves as the vehicle to remember the enable logic for this candidate. The one-to-one correspondence imposed by pin names between the input ports of the SleepModeModule and inputs of the SleepModeControlModule is required during the committing phase where the enable logic is actually connected. Because no extra load is added to the control and the data path, the critical path of the original design will be preserved. As a result, the later logic and timing optimization will be able to focus on the real critical path and optimize the netlist as if no sleep-mode transformations are performed. Therefore, as long as the sleep-mode candidates are not committed, there is no concern that the they may cause new timing violations when the original design meets timing after logic and timing optimization. Another advantage of the technique is that it allows only a portion of the enable function to be used as the enable logic for the sleep-mode candidate during the gate-level commitment phase, known as partial sleep-mode committing.

4.2 Gate-level Commitment

After logic and timing optimization, a decision to commit each individual candidate is made based on whether or not the power is reduced without violating the timing constraints. The algorithm is shown in Figure 6. First, we sort all sleep mode candidates based on the potential power savings. The commitment starts from the sleep-mode candidate which has the biggest potential power savings. Then for each selected candidate, the input ports of SleepModeModule and the input pins of SleepModeControlModule that have the same name are connected. The actual gating logic is inserted and an incremental power and timing analysis is performed. If the gate level

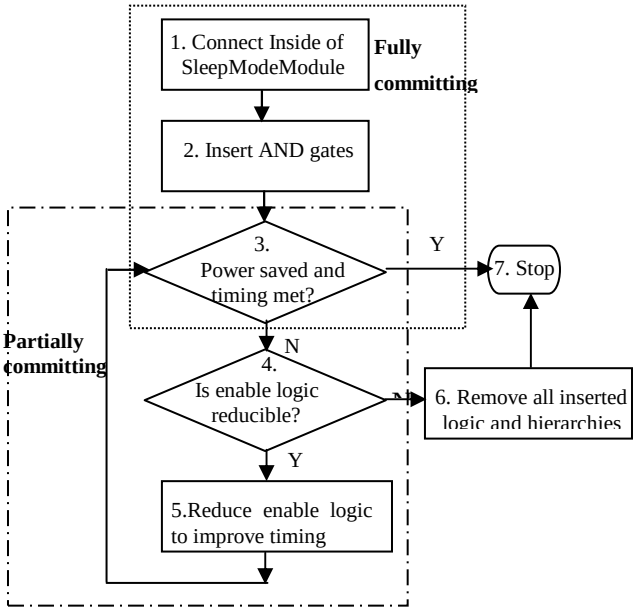

Figure 6. Gate level sleep-mode commit algorithm.

commitment succeeds at step 3, then it is called fully committed. That is, whatever candidates and enable logic identified at RTL is fully committed to save power without violating timing constraints. For example, to fully commit the sleep-mode modules inserted in Figure 5, we can simply connect the control portion of the circuit within the SleepModeModule and insert AND gates on the data signals, as shown in Figure 7.

4.3 Gate-level Partially Committing

However, the full commit may cause new timing violations. For example, let's assume that after fully committing all sleep-mode modules, the path from *b* to the second input of multiplier violates timing constraints, as highlighted in Figure 7. To avoid the newly introduced timing violation, one solution is to completely remove the sleep-mode logic at input *in2* of the multiplier, but doing this will significantly limit the power reduction that can be achieved. A much better solution is to partially commit the sleep-mode enable logic by executing steps 3 through 5 (in Figure 6). In this example, we know that the enable logic for this candidate is *a&b*. This means that either *a* or *b* can also be used as the enable logic. Since we do not want *b* to be part of the enable function because of timing violation, we can use *a* as the enable logic for input bus *in2*. In other words, by removing the input of *b* from the original enable function, the new enable function is reduced from *a&b* to *a*. Note that partial commitment is possible because the enable logic and the inputs for each sleep mode candidate are separately derived during RTL synthesis and preserved after logic and timing optimization. As shown in Figure 6, the partial committing loop stops when the enable logic cannot be reduced further. The partial commitment capability of the proposed approach differs significantly from the traditional approach where, if a sleep mode transformation violates timing constraints, it is completely removed. Our approach provides a much larger scope to perform power delay trade-offs.

Finally, if committing a sleep-mode candidate causes power increase or timing violation even after partial committing, we can simply un-commit the sleep-mode candidate by removing all inserted logic on the control and data path, and then dissolve the inserted hierarchical modules.

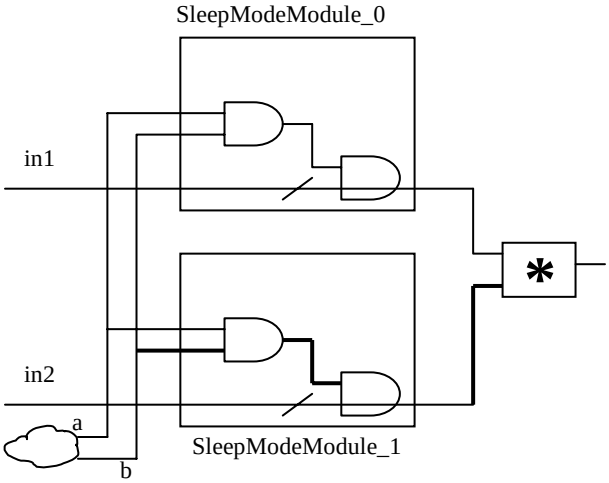

Figure 7. Netlist after gate-level sleep-mode full commitment for the example in Figure 5.

5. Flow Integration and Implementation

The proposed RTL power optimization technique can be seamlessly integrated into any existing design flow. The RTL exploration can be added at the end of normal RTL synthesis, and the gate-level commitment can be added after timing optimization completes. Existing flow needs no modification to apply the technique. More importantly, the separation of RTL exploration from gate-level eliminates possible iterations between the RTL and the gate level that can occur in traditional approaches. In traditional approaches, it is possible that timing constraints can not be met even by undoing all the sleep-mode transformations after gate level optimization, because they may have shifted the critical paths. This is not a problem for the proposed technique because the decision for commitment of a candidate is delayed until the circuit has been optimized for timing. At this stage, the accurate power and delay estimation can be easily obtained from the gate level power and timing analysis engine. As a result, each commitment makes sure that it saves power without violating timing constraints. The delay of the circuit after gate-level committing is guaranteed to be no worse than the delay of the

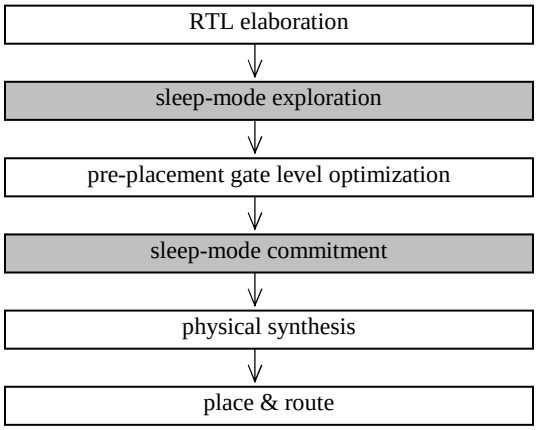

Figure 8. Flow integration

	# Inst.	Slack	Tot. Pow.	DP Power	
		(ns)	(mW)	(mW)	(%)
D. 1	6474	2.64	0.56	0.18	32.1%
D. 2	7461	1.35	6.87	1.82	26.6%
D. 3	26786	0.15	9.78	4.82	49.3%
D. 4	25434	0.24	15.21	1.45	9.5%
D. 5	65533	0.00	237.85	51.41	21.6%
D. 6	325216	-0.37	19.76	4.32	17.3%

Table 1. Designs used for experiments

	RTL Exploration		Gate-level commitment			
	SM cand.	CPU	SM Comm	Slack (ns)	Pow. Saved	CPU
D. 1	26	0.6%	18	0.75	8.11%	16.3%
D. 2	87	0.2%	4	1.24	1.10%	6.4%
D. 3	125	1.8%	45	0.01	13.60%	20.0%
D. 4	26	0.3%	14	0.24	3.00%	4.1%
D. 5	28	2.7%	21	0.00	6.62%	14.7%
D. 6	1602	1.9%	18	-0.36	4.31%	23.5%

Table 2. Experimental results with proposed method

circuit before committing. Finally, since the decision of sleep-mode commitment is delayed to gate-level, it eliminates the need for complicated and generally inaccurate RTL power and delay models [5].

The proposed algorithm is implemented in Cadence® PKS/LPS 5.0 release [10]. As shown in Figure 8, the standard flow consists of blocks with no shades and the sleep-mode flow includes the two shaded boxes. The flow starts from the RTL of a design. During RTL exploration, sleep-mode logic is inserted but not fully connected. After normal flow of pre-placement timing and datapath optimization, including resource sharing and operator merging [10], it then starts the gate-level commitment phase of the sleep-mode transformation. This is done before the physical placement of the design. The reason we do not perform the commitment phase after placement because the commitment phase may introduce a significant amount of new logic which could cause the incremental placement failure. The sign-off quality power and timing analysis engine provided by PKS are used to compute the power and delay (Step 3 of Figure 6) accurately in the commitment phase. More specifically, after insertion of the gating logic (Step 2 of Figure 6), the PKS/LPS incremental power and timing analysis engine is used to obtain the accurate power and delay information to compare it with the power and delay information before the transformation. If power is reduced and delay is not worsened, the sleep-mode transformation will then be accepted and committed into the netlist. Otherwise, the sleep-mode logic is removed. In other words, the delay and power penalty due to the inserted gating logic is considered during the commitment phase.

6. Experiment Results

Six industrial circuits were chosen for experiments. Design 4 is a microprocessor, Design 5 and 6 are communication chips. The types of the rest circuits are unknown. All designs except Design 6 have customer provided test benches, which were used to simulate the circuit to get the switching activities for power analysis. For Design 6, random input patterns were used to simulate to obtain the switching activities for power analysis. Table 1 shows the characteristics of each design. The column **Inst** shows the number of instances of the design. The columns **Slack** and **Tot. Pow** are the slack and power dissipation of the design after normal synthesis and optimization. The last column **DP Power** shows the total data-path component power and its percentage of the total power in column **Tot. Pow**. Since sleep-mode transformations target only for power optimization on these blocks, in the experiment, we only use the power of these blocks to show the power savings of the proposed technique.

The experimental results are shown in Table 2. Column **SM cand**. shows the number of sleep-mode candidates identified during RTL exploration. The second column **CPU** shows the run time *increase* in percentage of sleep-mode RTL exploration compared to that of normal RTL elaboration. The results show that the RTL exploration is very efficient and introduces almost no additional computation time. Column **SM comm.** shows the number of sleep-mode candidates committed in the gate-level commitment phase. Column **Slack** shows the slack after gate level optimization. Column **Pow. Saved** shows the total power *saved* as a percentage of the datapath power, as shown in Column 5 in Table 1. The last column shows the CPU run time of the gate-level commitment as a *percentage* of the total run time of the normal gate level optimization.

The area increase due to the final inserted sleep-mode modules are not shown because the maximum area overhead among all designs is less than 1%. Since the proposed method will not modify the original timing path after RTL exploration and the commitment of each sleep-mode logic is delayed after timing optimization, *the timing optimization will work as if there is no sleep mode logic inserted*. The target slack of gate level optimization for each design is 0. For designs with negative slack in Table 1, the final slack after gate level commitment is not worse than that in Table 1. For designs with positive slack in Table 1, the final slack is smaller but not less than 0 after the gate level commitment. The power savings can be achieved strongly depend on the timing constraints, the switching activities of the enable function, and the power dissipation of the data path blocks. For example, in Design 2, most of the sleep-mode candidates are not committed at the gate level because the probabilities of enable signals are close to 1 which leaves little room for power savings. For Design 6, most of the sleep-mode candidates are not committed due to the tight delay constraint.

As a comparison we did another set of experiments. In these experiments, after RTL exploration, we simply map the netlist without any other optimization and commit the sleep-mode candidates based on the power and delay information obtained at this level. This is similar to the traditional approach [5] where the sleep-mode transformations are committed based on RTL power and delay estimation. The results are shown in Table 3. The column **SM inserted** shows the number of sleep mode logic inserted. The column **slack** shows the slack after timing optimization. The column **power saved** shows the datapath power saved in percentage compared to the netlist without any sleep-

	SM Inserted	Slack (ns)	power saved	CPU overhead
D. 1	26	0.25	6.43%	0.98
D. 2	8	1.12	1.10%	1.01
D. 3	76	-0.21	-1.35%	1.63
D. 4	18	0.24	3.00%	0.96
D. 5	28	-0.78	2.10%	1.57
D. 6	853	-1.56	-11.56%	2.10

Table 3. Results of committing candidates at RTL

mode inserted, column 5 of Table 1. The column of **CPU overhead** is the ratio of the total CPU time of this experiment to the CPU time using the proposed technique. Compare the Table 2 and 3 we can see that for Design 1, 2 and 4, the results are very similar. However, the results of Design 3, 5 and 6 from Table 2 are much better than those from Table 3. For example, for Design 6 in Table 3, a lot of sleep mode candidates are committed based on the inaccurate power and delay information. Unfortunately, it was found during timing optimization that most of the inserted sleep-mode logic become part of the timing critical paths and significantly worsen the performance of the design. Therefore the timing optimization spends significantly more run time and gate resources to improve the slack. As a result, the final design consumes more power than the one without the sleep-mode transformations and misses the timing target. Overall, for Design 6, the proposed method achieved 4.3% power savings with no degradation of circuit delay and the traditional method produced a final netlist with more power, worst delay and ran twice slower.

To show the robustness of the proposed technique in terms of achieving best power delays trade-offs, we select Design 6 to run gate-level commitment with different and relaxed timing constraints. The result is shown in Table 4. Each row shows the experimental results with different set of timing constraints. Row 1 shows the results with the original timing constraints. Rows 2 to 4 show the results with the relaxed timing constraints. The column **Relaxed constr.** shows the degree of the relaxation of timing constraints in percentage. It can be seen that, for Design 6 when the timing constraint is relaxed, more sleep-mode candidates can be committed at the gate level and significant more power savings, as much as 45%, can be achieved. The results show that the achievable power savings of the proposed technique are mostly bounded by the timing constraints and the switching activities of the enable signals. The lower the activities of the enable signals and the looser the timing constraints, the more power reduction can be achieved. Given the specified timing constraints, the proposed technique will apply the sleep-mode transformations to save the most power without worsening the timing.

	Relaxed constr.	SM Inserted	power saved
not relaxed	0%	18	4.3%
relaxed 1	25%	281	38.3%
relaxed 2	50%	511	43.0%
relaxed 3	100%	673	45.3%

Table 4. Results of Design 6 with relaxed timing constraints

7. Conclusion and Future Work

In this paper we propose a new approach to apply known RTL power optimization technique to achieve the best possible power delay trade-offs. The concept of breaking the optimization into two steps, i.e. RTL exploration and gate-level commitment, has a lot of advantages over traditional techniques as shown in the experimental results. First, it has little impact on the normal logic and timing optimization because all sleep-mode modules inserted after RTL exploration preserve the original connectivity and loads of the data and control paths of the design. Therefore the run time and final slack of timing optimization are the same as if there is no sleep-mode logic inserted. This makes the proposed technique very practical for current main stream design methodology where the first design target is timing and power is normally a secondary concern. Secondly, since the committing of the sleep-mode is delayed until the design is timing optimized, accurate power delay trade-off can be achieved given the exact timing and power information available at that point. Finally it has the capability of partial committing of the sleep-mode modules which leads to a much larger solution space being explored to achieve the optimal power and delay trade-off. Overall, no iteration between RTL and gate-level optimization required to achieve timing closure and save power when using the proposed technique. The experimental results show that the proposed method can achieve meaningful power savings with no impact on existing normal timing optimization at all where the traditional methods may produce a design with worse timing and power and longer run time.

Currently we are working on apply the same concept to other RTL power transformations such as clock gating.

8. REFERENCES

[1] A.P. Chandrakasan and R. W. Brodersen, "Low Power Digital CMOS Design", Kluwer Academic Publishers, 1995.

[2] K. Keutzer and P. Vanbekergen, "The Impact of CAD on the Design of Low Power Digital Circuits"

[3] M. Pedram, "Power Minimization in IC Design: Principles and Applications," *ACM Transactions on Design Automation of Electronics Systems*, January 1996, Vol 1-1, pp. 3-56.

[4] H. Kapadia, L. Benini, and G. De Micheli, "Reducing Switching Activity on Datapath Buses with Control-Signal Gating", IEEE Journal of Solid-State Circuits, Vol. 34, No. 3, March 1999, pp. 405-414.

[5] M. Munch, and et. al., "Automating RT-Level Operand Isolation to Minimize Power Consumption in Datapaths", Proceedings of Design and Test Automation Conference in Europe.

[6] S. Dey and et. al., "Controller-Based Power Management for Control-Flow Intensive Designs", IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems, Vol. 18, No. 10, October 1999, pp. 1496-1508.

[7] J. van Eijndhoven, G. de Jong, and L. Stok. The ASCIS data flow graph: semantics and textual format. Technical Report EUT-Report 91-E-251, Eindhoven University of Technology, Eindhoven, Netherlands, June 1991.

[8] De Micheli. Synthesis and Optimization of Digital Circuits. McGraw-hill, Inc., 1994.

[9] R. Bryant. Graph-based algorithms for boolean function manipulation. IEEE Transactions on Computers, C-35(8):677-691, Aug. 1986.

[10] Cadence® Low Power Synthesis (LPS) User's Guide for Cadence® PKS.