Power-Gating-Aware High-Level Synthesis

Eunjoo Choi[§], Changsik Shin[†], Taewhan Kim[‡], and Youngsoo Shin[†]
[§]System IC Business Team, LG Electronics, Seoul 135-985, Korea
[†]Department of Electrical Engineering, KAIST, Daejeon 305-701, Korea
[‡]Department of Electrical Engineering, Seoul National University, Seoul 151-742, Korea

ABSTRACT

A problem inherent in designing power-gated circuits is the overhead of the *state-retention storage* required to preserve the circuit state in standby mode. Reducing the amount of retention storage is known to be the most influential factor in minimizing the loss of the benefit (i.e. power saving) by power-gating. In this paper, we address a new problem of high-level synthesis with the objective of minimizing the size of retention storage to be used in the power-gated circuits. Specifically, we propose a complete design framework, called HLS-pg, that starts from the power-gating-aware scheduling, allocation, and controller synthesis down to the final circuit layout. The key contribution of the work is to solve the power-gating-aware scheduling problem, namely, scheduling operations that minimizes the number of retention registers required at the power-gating control step, while satisfying resource and latency constraints. In experiments on benchmark designs implemented in 65-*nm* CMOS technology, HLS-pg generates circuits with 27% less leakage current, with 6% less circuit area and wirelength, compared to the power-gated circuits produced by conventional high-level synthesis.

Categories and Subject Descriptors: B.7.1 [**Integrated Circuits**]: Types and Design Styles—*VLSI*

General Terms: Algorithms, Design

Keywords: High-level synthesis, power-gating, leakage

1. INTRODUCTION

Leakage current has been continuously growing to the point where it is now comparable to switching power. In recent nanometer CMOS technologies below 90-*nm*, it is not uncommon to see that leakage current is responsible for more than 50% of the total power consumption [1]. Leakage current comes from many sources [2], but subthreshold leakage takes the largest portion in most technologies.

There have been many circuit techniques to suppress subthreshold leakage, such as power-gating, reverse body bias, and so on [3]. The power-gating scheme [4], which is the most popular and has been extensively used in industry [5–7], reduces the standby leakage by cutting the circuit off from its power supply. The power-cutting is realized through controlling a current switch (called a footer) that is located between a logic block and V_{ss}, as shown in Figure 1, or between V_{dd} and a logic block (called a header). When the footer is turned off, V_{ssv} rises slowly towards V_{dd}; this damages the current states in the storage elements, so that alternative storage elements, which are capable of state retention, must be introduced.

ISLPED'08, August 11–13, 2008, Bangalore, India.

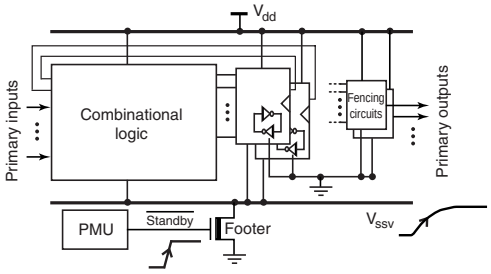

Figure 1: An example showing the operation of a power-gated circuit.

They include extra latches, which are not power-gated, so that they preserve the state, and are collectively called *state-retention storage*[1], or retention storage for short. In addition to the extra latches in the retention registers, which are fully biased, fencing logic connected to the ground is needed to avoid the floating of primary outputs values [8].

There are several variants [6, 9–11] on the implementation of state retention storage. However, they invariably incur a substantial amount of overhead in terms of area, wirelength, and leakage current. A retention flip-flop has been reported to require 68% more area than a conventional flip-flop [8]. This is the main reason for the increase in area of power-gated sequential circuits, which has been observed to be in the range 13% to 28% [8]. In addition, the total wirelength of power-gated circuits typically increases by 29% to 60% [8], due to the wires for extra control signals needed for the retention flip-flops and wiring congestion of other signals that is caused by wiring the extra control signals. A retention flip-flop usually preserves its state in an extra latch, which is fully biased during standby mode since it is not power-gated, and this extra latch induces continuous gate leakage, which is the main contributor to total leakage current, and this situation rapidly deteriorates as the CMOS technology scales [12].

Consequently, reducing the size of retention storage is an important issue in designing power-gated circuits. However, minimizing the size of state retention storage is ineffective or less effective at the logic synthesis or physical design stage, since the structure or many key design parameters of the circuits, or both, have already been fixed. In this work, we address a new problem of synthesizing power-gated circuits in high-level synthesis, with the objective of minimizing the number of retention registers. Our main contributions are:

- A complete framework for power-gating, called HLS-pg, that starts from the power-gating-aware scheduling, allocation, and controller synthesis to the final circuit layout. The proposed design flow captures many implementation details, such as footer sizing, timing closure, placement and routing.

- An optimal solution of the scheduling problem for power-

[1]In high-level synthesis, the storage refers to registers, while in logic and circuit synthesis, it refers to flip-flops or latches.

gated circuits, based on integer linear programming (ILP), which minimized the number of retention registers while satisfying resource and latency constraints.

- Extensive experimental results from commercial 65-*nm* technology applied to behavioral benchmark designs.

2. PRELIMINARIES AND DESIGN FRAMEWORK

2.1 Architecture and Problem Definition

Our target architecture consists of a data-path that has functional units, registers, and their connections, a controller for the data-path, and a power management unit (PMU) (see Figure 1) that initiates state changes (from active to standby and vice versa) through a `sleep` signal, as well as resetting the circuit. The registers in the data-path are classified into two types: normal registers and registers that can retain data in the standby state. The controller, which is a finite state machine (FSM), is assumed to receive an asynchronous `sleep` signal from the PMU, and subsequently generates a `standby` signal, which in turn power-gates the data-path, and a `ret` signal, which enforces preservation of the states in the retention registers. When it receives the de-asserted `sleep`, the controller wakes up the data-path by de-asserting `standby` and `ret`.

To clarify the high-level synthesis problem that we are addressing, we assume that:

1. Power-gating is applied to the entire data-path. That is, partial power-gating of a data-path is not considered.
2. An m-bit retention register has exactly m extra latches to hold the m-bit data during standby. Thus, minimizing the total number of bits of the retention registers also minimizes the overhead of the retention logic.
3. The time between the detection of `sleep` and effective power-gating should not be greater than a designer-specified value L, which is the latency of the design. This means, among the control steps in the latency, only one of them involves the actual power gating, and we call this the *power-gating control step* C_{pg}.

Note that relaxing Assumptions (1) and (2) above would result in a less regular design and increase control overheads. Assumption (3) is reasonable if L is relatively short. For large L, several power-gating control steps may be required, which our synthesis can easily be extended to support.

Let G be a scheduled data-flow graph (DFG), and let $S(i)$ be a set of variables in G that are alive during control step i. Then $|S(i)|$ will be the exact number of registers required to store the variables during control step i. (For simplicity of presentation, we will assume that the widths of all the variables are the same number of bits).

Problem 1 Given an unscheduled data-flow graph G with latency (L) and resource (functional unit) constraints, and a power-gating control step C_{pg}, the power-gating-aware HLS problem is *to generate a data-path by finding a schedule of operations in G, and allocating functional units/registers/connections to operations/variables/data-transfers with the objective of minimizing* $|S(C_{pg})|$ *while satisfying the latency and resource constraints.*

Note that Problem 1 can be generalized, so that C_{pg} is a parameter to be determined rather than a fixed one. Consequently, the generalized problem is to generates a data-path that minimizes

$$\kappa = min\{|S(1)|,\ |S(2)|,\ \cdots,\ |S(L)|\}. \quad (1)$$

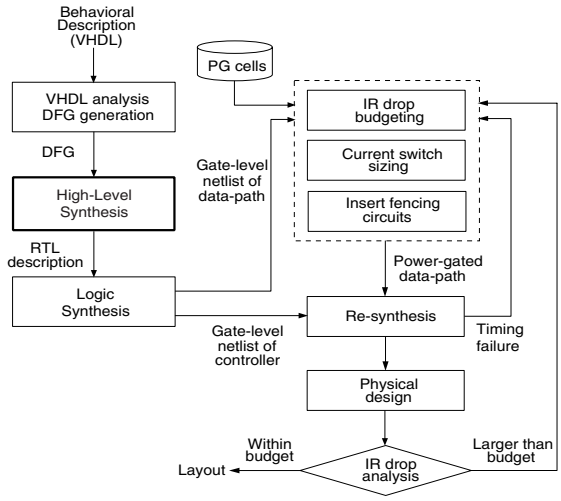

Figure 2: Overall design flow based on HLS-pg.

Clearly, the generalized problem can be solved by solving Problem 1 L times. The key consideration in solving Problem 1 is to schedule the operations so that the number of variables (i.e. data values) that are alive in control step C_{pg} is as small as possible, since two variables whose lifetimes are not disjoint cannot be stored in the same register.

2.2 Design Framework

The overall design flow based on HLS-pg is shown in Figure 2. A behavioral description written in VHDL is first analyzed [13] and is then transformed into a DFG [14]. The DFG will then be an input to the HLS system, which will be described in Section 3. The RTL description generated by the HLS system then goes through a standard logic synthesis to create an initial gate-level netlist.

From the data-path component of the netlist, we size the current switches (i.e. footer or header), which affects the active-mode circuit delay. To do this, we must first determine the voltage drop that is allowed across the switches when they are turned on during active mode, in an empirical process that we call *IR drop budgeting*. We can also determine the average current through the data-path by applying random logic patterns to the inputs of a circuit simulation of that part of the netlist. Using this estimate of the average current and the chosen voltage drop, we can then decide on the size of the current switches [15], which in turn determines the number of switch cells required. Fencing circuits [6] are also required to stop the primary output from the data-path floating when it is power-gated.

The whole netlist is then re-synthesized with V_{dd} set to the voltage swing that each gate will experience. In the data-path component of the netlist, this is the original V_{dd} minus the chosen voltage drop across a current switch, and in the controller component it is the original V_{dd}. If the timing constraints are not satisfied by this re-synthesis, we reduce the voltage drop across the current switches, at the expense of an increase in switch size. This process is repeated until the timing constraints are met.

In the physical design stage, the current switches are placed in evenly spaced locations on the left and right hand sides of the placement region, followed by an automatic placement and routing. The voltage drop across the current switches is then checked in the layout to see if it dose not exceed the chosen allowance. If it violates the allowance, the design process is repeated as indicated in the right side of Figure 2.

In the physical design stage, we first partition the placement re-

gion into two parts, one for the controller and the other for the data-path, which will require different power rails: the controller needs V_{dd} and V_{ss}, because it is not power-gated; and the data-path needs V_{dd} and V_{ssv} (see Figure 1). The current switches for the data-path are placed at evenly spaced locations on the left- and right-hand sides of the placement region. This is followed by automatic placement and routing [16] of the whole netlist. The voltage drop across the current switches is then checked [17] in the layout to ensure that it does not exceed the chosen allowance. If the voltage drop violates the allowance, the design process is repeated, as indicated on the right-hand side of Figure 2. An example layout produced by the physical design process will be presented in Section 4.

3. POWER-GATING-AWARE DATA-PATH SYNTHESIS

The high-level synthesis in Problem 1 consists of three tasks: operation scheduling, resource allocation, and controller synthesis. Since the number of retention registers exactly matches the total number of data values that should be alive at control step C_{pg}, the objective of operation scheduling is to find a schedule that minimizes the number of data values that are alive at control step C_{pg}. On the other hand, the tasks of resource allocation and controller synthesis are not likely to affect the number of retention registers once the schedule of operations are determined.

3.1 Power-Gating-Aware Scheduling

We formulate the scheduling component of Problem 1 as a 0-1 linear programming (ILP) problem, to which we can find an optimal solution. For a large size DFG, we first apply a list scheduling [18] and then partition the scheduled DFG into components of reasonable size. Our algorithm is then applied to the partitioned DFG that contains C_{pg}, in order to reschedule the operations so as to find a minimum number of retention registers.

The ILP formulation is given in the following two subsections. In Sec. 3.1.1, we will formulate the ILP under the simplifying assumption that the output value of each operation in the DFG is consumed by exactly one other operation. In Section 3.1.2, we will present a fuller ILP formulation in which multiple consumers can be supported without any serial data dependencies.

3.1.1 Basic ILP Formulation

We wish to find a schedule of operations such that the number of variables whose lifetimes include control step C_{pg} is minimized under the latency and resource constraints, while assuming that each variable is consumed by only one operation. This is unlike conventional ILP-based schedulers (e.g. [19]) which seek to minimize latency under resource constraints or resources under a latency constraint.

Let $V = \{v_1, v_2, \ldots, v_n\}$ be a set of operations in the DFG, and let E be a set of data dependencies among these operations. A directed edge $(v_i, v_j) \in E$ indicates that the variable produced by v_i is used as an input to v_j. We use the following notation in our formulation:

- C_{pg}: the control step for power-gating,
- L: the latency bound,
- A_k: the number of available functional units of type k,
- $f(v_i)$: the type of functional unit on which v_i can be performed,
- d_i: the number of control steps (i.e. delay) for executing v_i,

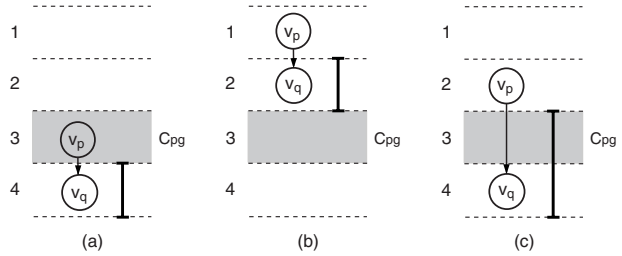

Figure 3: Linear expressions of variables $x_{i,j}$ represent the intersection of the lifetime of a data value (denoted as intervals on the right of DFGs) with C_{pg}: (a) lifetime starts after C_{pg}, $\Sigma_{i>3} x_{q,i} - \Sigma_{i>3} x_{p,i} = 1 - 1 = 0$; (b) lifetime ends before C_{pg}, $\Sigma_{i>3} x_{q,i} - \Sigma_{i>3} x_{p,i} = 0 - 0 = 0$; and (c) lifetime crosses C_{pg}, $\Sigma_{i>3} x_{q,i} - \Sigma_{i>3} x_{p,i} = 1 - 0 = 1$.

- t_i^S: the earliest control step at which operation v_i can be scheduled, obtained by as soon as possible (ASAP) scheduling [20],
- t_i^L: the latest control step at which operation v_i can be scheduled, obtained by as late as possible (ALAP) scheduling [20] with the latency bound L,
- $x_{i,j}$: a Boolean variable that indicates the beginning of the lifetime of the variable produced by v_i. If that lifetime starts from control step j then $x_{i,j} = 1$; otherwise $x_{i,j} = 0$.

Note that we use $x_{i,j}$ as a variable start time rather than as the more usual operation start time [19]. Therefore, solving for $x_{i,j}$ implicitly determines the operation start time, which is $j - d_i$, for the value of j that makes $x_{i,j}$ equal to 1.

Objective function: For the purpose of ILP, we require a linear expression of the variables $x_{i,j}$ that expresses whether the lifetime of a data value includes C_{pg}.

- (Relation-1) For the data-dependency edge $(v_p, v_q) \in E$, we can assert $\sum_{i>C_{pg}} x_{p,i} = 1$ if its lifetime starts after C_{pg}, but otherwise $\sum_{i>C_{pg}} x_{p,i} = 0$.
- (Relation-2) For the data-dependency edge $(v_p, v_q) \in E$, we can assert $\sum_{i>C_{pg}} x_{q,i} = 1$ if its lifetime ends at or after C_{pg}, but otherwise $\sum_{i>C_{pg}} x_{q,i} = 0$.

If $\Sigma_{i>C_{pg}} x_{p,i} = 1$ we know that $\Sigma_{i>C_{pg}} x_{q,i} = 1$, because starting later than C_{pg} implies ending later than C_{pg}, as shown in Figure 3(a). Similarly, $\Sigma_{i>C_{pg}} x_{q,i} = 0$ means that $\Sigma_{i>C_{pg}} x_{p,i} = 0$, because ending earlier than C_{pg} implies starting earlier than C_{pg}, as shown in Figure 3(b). The remaining case involves starting at or earlier than C_{pg} and ending at or later than C_{pg}, as shown in Figure 3(c), which can be expressed by $\Sigma_{i>C_{pg}} x_{p,i} = 0$ and $\Sigma_{i>C_{pg}} x_{q,i} = 1$. Therefore, for any edge $(v_p, v_q) \in E$, if $\Sigma_{i>C_{pg}} x_{q,i} - \Sigma_{i>C_{pg}} x_{p,i} = 1$ then the lifetime of the variable produced by v_p and consumed by v_q crosses C_{pg} (so that the corresponding data value has to be preserved), otherwise it does not. This allows us to formulate the following objective for the scheduling problem:

$$\text{Minimize} \sum_{\forall (v_p, v_q) \in E} \left\{ \sum_{i>C_{pg}} x_{q,i} - \sum_{i>C_{pg}} x_{p,i} \right\}. \quad (2)$$

The constraints for our ILP formulation can now be expressed as follows:

$$\sum_{j=t_i^S+d_i}^{t_i^L+d_i} x_{i,j} = 1, \ \forall v_i \in V \quad (3)$$

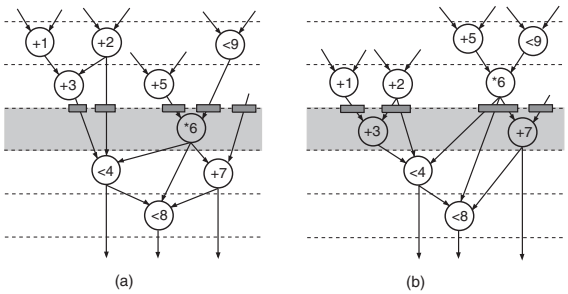

Figure 4: A simple DFG to show how scheduling affects the number of retention registers: (a) result obtained by list scheduling ($|S(C_{pg}=3)|=5$); and (b) result obtained by HLS-pg ($|S(C_{pg}=3)|=4$).

$$\sum_{j=t_i^S+d_i}^{t_i^L+d_i} j \cdot x_{i,j} \;\le\; L+1, \;\; \forall v_i \in V \tag{4}$$

$$\sum_{i:f(v_i)=k} \sum_{j=l+1}^{l+d_i} x_{i,j} \le A_k, \; l=1,2,\ldots,L, \; k=1,2,\ldots \tag{5}$$

$$\sum_i i \cdot x_{p,i} \;+\; d_q \;\le\; \sum_j j \cdot x_{q,j}, \;\; \forall (v_p,v_q) \in E. \tag{6}$$

Constraint (3) ensures that the control step at which the variable is produced by v_i is unique, which implies that the control step at which v_i itself is scheduled is also unique. The latency constraint is (4), the data dependency constraint is (6), and the resource constraint (5) ensures that the maximum number of operations of the same type ($f(v_i)=k$) executed at each control step does not exceed the number of available functional units A_k. This constraint (5) is evaluated for each resource type (k) at each control step (l).

Example 1: Consider the DFG in Figure 4(a), which has 5 additions, 1 multiplication, and 3 comparisons. It has been scheduled by a resource-constrained list scheduling algorithm which was allowed 2 adders, 1 multiplier, and 1 comparator. Each operation is assumed to take one clock cycle for its execution. The resulting schedule takes 5 control steps; when $C_{pg}=3$, 5 retention registers ($|S(C_{pg})|=5$) are required, as indicated by the small boxes in Figure 4(a). We now re-schedule the DFG with our ILP formulation under the same resource constraints and with the same latency ($L=5$). We first run ASAP and ALAP schedulers to obtain a sequence of control steps at which each operation can exist. We then identify the edges that we need to include (i.e. those that have the potential to cross C_{pg}) in our objective function (2): $(v_1,v_3),(v_2,v_4),(v_3,v_4),(v_5,v_6),(v_6,v_8)$, and (v_9,v_6). Note that v_2 and v_6 have multiple consumers, but both of them have serialized dependencies. We now re-schedule the DFG with our ILP formulation under the same resource constraints and with the same latency ($L=5$):

$$\begin{aligned}\text{Minimize} \quad & (x_{3,4}-0)+(x_{4,4}+x_{4,5}-0)+(x_{4,4}+x_{4,5}-x_{3,4}) \\ & +(x_{6,4}-0)+(x_{8,5}+x_{8,6}-x_{6,4})+(x_{6,4}-0).\end{aligned}$$

Constraints (3), (4), (5), and (6) now become:

$$\begin{aligned} & x_{1,2}+x_{1,3}=1, \;\ldots \\ & 5x_{8,5}+6x_{8,6} \le 6, \;\ldots \\ & x_{1,2}+x_{2,2}+x_{5,2} \le 2, \;\ldots \\ & 2x_{1,2}+3x_{1,3}+1 \le 3x_{3,3}+4x_{3,4}, \;\ldots \end{aligned}$$

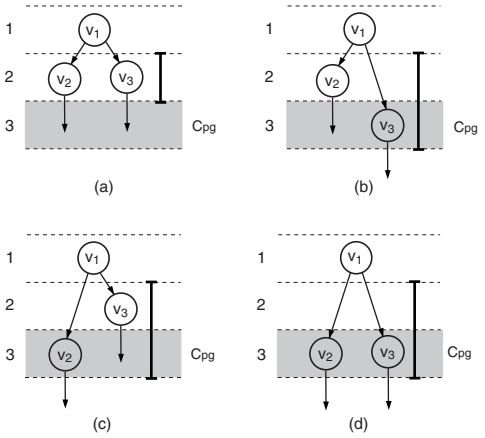

Figure 5: An example of a multiple fanout operation.

Figure 4(b) shows the result of solving the ILP formulation where $|S(C_{pg}=3)|$ is reduced to 4. □

3.1.2 *ILP formulation supporting multiple fanout operations*

The lifetime of a data value that is produced by v_p and consumed by v_q alone is determined by $x_{p,i}$ and $x_{q,j}$, and spans the control steps from i to $j-1$, for values of i and j that make $x_{p,i}$ and $x_{q,j}$ equal to 1. If a data value is consumed by more than one operation, and those operations have mutual dependencies (see Example 1), its lifetime is still determined by just two operations: the producer, and the consumer at the bottom of the dependency chain.

However, if the multiple operations that consume a data value are independent, the situation is different. Assume that v_1 produces a data value which is then consumed by v_2 and v_3, as shown in Figure 5. We will also assume that v_1 can only be placed at the first control step; and that v_2 and v_3 can be placed either at the second or at the third control step. The lifetime of the data produced by v_1 is determined by the edge (v_1,v_3) in the schedule of Figure 5(b), but in Figure 5(c) it is determined by (v_1,v_2). In the schedules of Figure 5(a) and (d), either (v_1,v_2) or (v_1,v_3) can be used to determine the lifetime. Since the operation control steps are not known, the edge we need to include in the ILP formulation is not fixed.

To resolve this problem, we introduce an imaginary variable, which we call *group variable* $y_{p,j}$, which inherits $x_{q,j}$ in the sense that v_q is one of the consumers of the data produced by v_p (i.e. $(v_p,v_q) \in E$) and the lifetime of the data produced by v_q starts at the latest control step (i.e. at the maximum j which makes $x_{q,j}$ equal to 1). In the example of Figure 5, the group variable $y_{1,j}$ can be defined as follows:

$$y_{1,j} = \begin{cases} x_{2,j} & \text{if } \sum_j j \cdot x_{2,j} \ge \sum_j j \cdot x_{3,j} \\ x_{3,j} & \text{otherwise.} \end{cases} \tag{7}$$

These group variables allow us to continue to use the ILP formulation in the previous section with only slight modification. In Objective (2), we replace $x_{q,i}$ with the group variable $y_{p,i}$, for a data value with multiple consumers. In Figure 5, for example, we use

$$\sum_{i>C_{pg}} y_{1,i} - \sum_{i>C_{pg}} x_{1,i}$$

as the inner sum of the objective for edges (v_1,v_2) and (v_1,v_3).

We also need new constraints, which are added to the basic ILP

formulation (Constraints (3) to (6)). These new constraints are:

$$\begin{aligned} y_{1,3}+y_{1,4} &= 1 \\ 3x_{2,3}+4x_{2,4} &\le 3y_{1,3}+4y_{1,4} \\ 3x_{3,3}+4x_{3,4} &\le 3y_{1,3}+4y_{1,4} \\ x_{2,3}+x_{2,4}+x_{3,3}+x_{3,4} &\ge y_{1,3}+y_{1,4} \\ x_{2,4}+x_{3,4} &\ge y_{1,4}. \end{aligned}$$

The first constraint ensures that the start time of the group variable is unique, and therefore corresponds to the original Constraint (3), but this is in addition to the constraints for $x_{1,j}$, $x_{2,j}$, and $x_{3,j}$. The next four constraints correspond to Constraint (7), and ensure that the group variable $y_{1,j}$ inherits $x_{2,j}$ if the lifetime of the data produced by v_2 starts later than that of the data produced by v_3, and that otherwise $y_{1,j}$ inherits $x_{3,j}$.

3.1.3 Complexity

Let the number of operations in a DFG be n and let the number of edges be m. The constraints that ensure that all the normal and group variables start at their own unique control steps (Constraint (3)) involve n group variables at most, so that no more than $m+n$ equations are generated. Likewise, the number of inequalities for the latency constraint (4) and resource constraint (5) is $m+K\cdot L$, where K is the number of resource units. The number of inequalities in the data dependency constraint (6) for the normal and group variables are bounded by $O(m^2)$ and $O(n(m+L))$ respectively. Thus the total number of constraints used in the ILP formulation is $O(m^2+nm)$, and the number of variables is $O((n+m)L)$.

3.2 Allocation and Control Synthesis

The allocation of functional units to operations, registers to variables, and connections to data transfers are tightly inter-related. Minimization of the number of retention registers in the scheduling phase can lead to an unsatisfactory allocation phase in which the total number of registers actually increases. The register allocation is formulated as vertex coloring of a register conflict graph: each vertex in this graph corresponds to a lifetime of a variable, and there is an edge between two vertices if there is an overlap of lifetimes. We use the heuristic vertex coloring algorithm [21] for this register allocation phase.

The allocation of functional units and connection are also formulated as vertex coloring of resource conflict graph and connection conflict graph, respectively. Since, these conflict graphs in our DFG belong to interval graphs, which can be colored optimally, we use the exact left-edge algorithm [22, 23].

The FSM controller is synthesized as a hard-wired sequential circuit; thus, it is described as a state transition graph [20]. Allocation and control synthesis generates the data-path and controller as a Verilog HDL. The remaining logic and physical synthesis shown in Figure 2 are then performed.

4. EXPERIMENTAL RESULTS

We carried out experiments on a set of behavioral benchmark designs [24] to assess the effectiveness of HLS-pg in reducing the number of retention registers and the leakage current, area, and wirelength of the final circuit. HLS-pg was implemented in C under SunOS 5.8, and each design was synthesized in commercial 0.9 V, 65-*nm* bulk CMOS technology. We used a public ILP-solver package [25] to solve the ILP formulation produced by HLS-pg. Footer switches were implemented as high-V_t nMOS devices, and placed in evenly spaced locations on the left- and right-hand sides of the placement region for the data-path. We forced at least 70% of the placement region to be occupied by cells during automatic placement; metal layers up to M5 were allowed for routing.

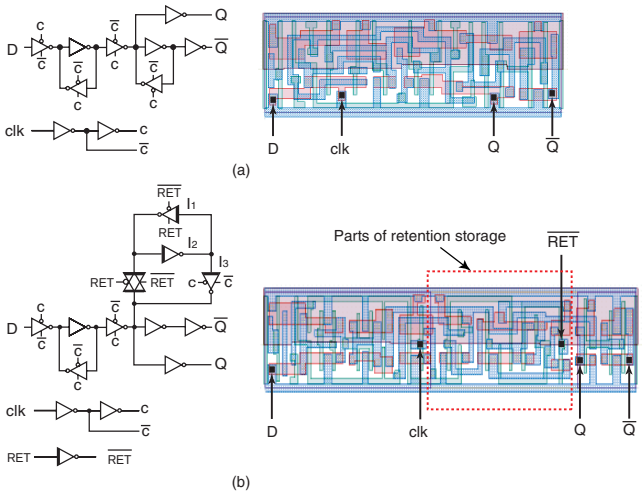

Figure 6: Schematic and layout of (a) a normal D flip-flop and (b) a state retention D flip-flop.

Fig. 6(a) shows a schematic and layout of the normal D flip-flop used to construct normal registers; and Fig. 6(b) shows the D-type retention flip-flop used to construct retention register. The retention flip-flop preserves the state in the latch consisting of the inverters I_1 and I_2 (both use high V_t) during standby mode (RET is high); the state is saved in the latch consisting of I_1 and I_3 during active mode. Note that all the elements except I_1 and I_2 are connected to the footer, and so they are power-gated during standby. The retention flip-flop uses 52% more area and 10 times more leakage current than the normal flip-flop (when both are power-gated).

We compared results from HLS-pg and conventional list scheduling [18] under the same resource and latency constraints. All the steps in Fig. 2 other than the scheduling are the same for both approaches. The comparisons are summarized in Table 1. The second column shows the resource constraint, expressed as the number of multipliers and ALUs. The third column corresponds to latency constraint. The results produced by conventional HLS using a list scheduler [18] are shown in the next four columns, and the results from HLS-pg are shown in the following four columns. The last three columns show the reduction in leakage current, area, and wirelength achieved by HLS-pg over conventional HLS, which does not consider the minimization of retention registers, thus implementing the entire storage with retention registers.

Note that, during standby mode, the major sources of leakage current are the retention registers and the fencing circuits (leakage from the footer switches is relatively small and can be safely ignored). The total leakage values given in the table are the sum of the leakage from the retention registers and the fencing circuits. To summarize Table 1, the total leakage current is reduced by 26.8% on average. We noticed that a fencing circuit leaks out 65% less current than a retention flip-flop from experiments. Moreover, since the number of fencing circuits for both methods are the same, the number of retention registers used in the design is the most influential factor in reducing the leakage current in power-gated circuits.

In addition to the big saving in the leakage current, both area and wirelength were also reduced by 6.3% on average. This saving comes directly from a reduced number of retention registers. Note that the saving in area and/or wirelength is large in some benchmarks due to the change of the number of total registers and/or multiplexers (see, for example, IIR7 and WAVELET in Table 1).

Table 1: Comparison of results produced by a conventional list scheduler [18] and by our HLS-pg

Benchmark	Res. (*, +)	L	HLS with list scheduling				HLS-pg				Savings		
			# Retention registers	Leakage (nA)	Area (μm^2)	Wirelength (mm)	# Ret. / # Total registers	Leakage (nA)	Area (μm^2)	Wirelength (mm)	Leakage (%)	Area (%)	Wirelength (%)
IIR7	(1,2)	16	19	18.8	19938	178	14 / 19	14.3	19325	172	24.1	3.1	3.5
	(3,2)	14	23	23.0	32778	309	14 / 20	14.9	26588	221	35.3	18.9	28.5
FIR11	(1,1)	12	16	15.4	16169	127	8 / 16	8.2	14999	123	46.9	7.2	2.6
	(2,1)	11	16	15.7	19206	196	8 / 16	8.5	20974	178	45.9	8.4	8.8
ELLIPTIC	(1,3)	16	17	18.9	24345	270	14 / 17	16.2	22820	243	14.1	6.3	9.8
	(2,3)	14	18	20.4	29381	327	14 / 18	16.8	29216	333	17.7	0.6	-1.6
LATTICE	(1,1)	12	14	14.9	17381	147	10 / 14	11.3	16224	140	24.3	6.6	4.6
	(2,2)	9	14	15.5	21959	203	10 / 14	12.0	21686	202	23.2	1.2	0.3
VOLTERRA	(3,1)	13	19	20.1	29195	277	12 / 20	13.7	30425	294	31.5	-4.2	-6.1
	(4,1)	13	20	21.3	34795	336	12 / 18	14.1	32200	301	33.9	7.5	10.6
WDF7	(2,3)	13	38	40.4	43371	503	29 / 37	32.3	41297	478	20.1	4.8	4.8
	(3,2)	13	39	41.3	49017	563	29 / 36	32.3	47216	534	21.9	3.7	5.1
WAVELET	(2,2)	16	29	32.3	40287	518	25 / 29	28.7	39258	493	11.2	2.6	4.8
	(3,2)	15	36	39.0	51254	610	25 / 29	29.0	42771	557	25.5	16.6	8.7
Average											**26.8**	**6.3**	**6.3**

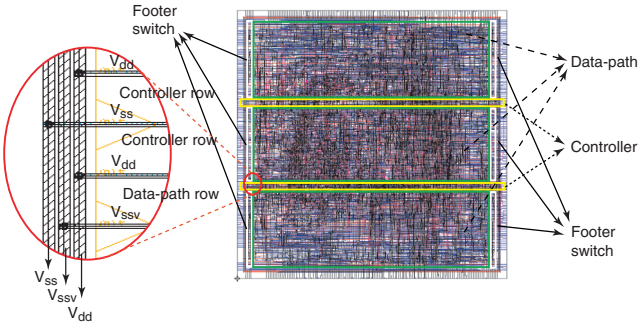

Figure 7: Layout produced by HLS-pg for design FIR11.

Figure 7 shows the layout of design FIR11 produced by HLS-pg following the design flow shown in Figure 2. The current switches are located evenly on the left- and right-hand side of the placement region to minimize the current path through V_{ssv} and V_{ss}. The FSM controller, which is not power-gated, has V_{dd} and V_{ss} rails; and the data-path, which is power-gated, has V_{dd} and V_{ssv} rails, as shown on the left of the layout in Figure 7. Due to the large number of control signals that need to be routed from the controller to the data-path, the controller was partitioned into two segments which were placed between data-path segments. This turned out to alleviate the routing congestion in the region between the controller and the data-path.

5. CONCLUSION

We have presented a method of high-level synthesis of power-gated circuits, focusing on the primary problem of minimizing the amount of storage needed for data retention. The HLS-pg framework includes the complete design flow for synthesizing power-gated circuits, from operation scheduling to circuit layout, using commercial 65-*nm* technology. The core of HLS-pg is an optimal solution to the problem of finding a schedule with a minimum number of retention registers. HLS-pg includes a new solution to this scheduling problem, which is achieved by formulating it as an integer linear programming problem with a concise choice of variables, objective function, and constraints. Experiments on benchmark designs showed that HLS-pg can reduce leakage current by 27% on average, while cutting area and wirelength by 6% over a conventional high-level synthesis which is not specialized for power-gating. In future work, we could consider high-level synthesis, where we minimize the number of total registers and/or multiplexers as well as retention registers.

References

[1] J. Friedrich et al., "Design of the Power6 microprocessor," in *Proc. ISSCC*, Feb. 2007, pp. 96–97.

[2] K. Roy, S. Mukhopadhyay, and H. Mahmoodi-Meimand, "Leakage current mechanisms and leakage reduction techniques in deep-submicrometer CMOS circuits," *Proceedings of the IEEE*, vol. 91, no. 2, pp. 305–327, Feb. 2003.

[3] S. G. Narendra and A. Chandrakasan, Eds., *Leakage in Nanometer CMOS Technologies*, Springer, 2005.

[4] S. Mutoh et al., "A 1-V power supply high-speed digital circuit technology with multithreshold-voltage CMOS," *IEEE JSSC*, vol. 30, no. 8, pp. 847–854, Aug. 1995.

[5] S. V. Kosonocky et al., "Enhanced multi-threshold (MTCMOS) circuits using variable well bias," in *Proc. ISLPED*, Aug. 2001, pp. 165–169.

[6] H.-S. Won et al., "An MTCMOS design methodology and its application to mobile computing," in *Proc. ISLPED*, Aug. 2003, pp. 110–115.

[7] P. Royannez et al., "90nm low leakage SoC design techniques for wireless applications," in *Proc. ISSCC*, Feb. 2006, pp. 138–139.

[8] H.-O. Kim and Y. Shin, "Semicustom design methodology of power gated circuits for low leakage applications," *IEEE TCAS II*, vol. 54, no. 6, pp. 512–516, June 2007.

[9] S. Shigematsu et al., "A 1-V high-speed MTCMOS circuit scheme for power-down application circuits," *IEEE JSSC*, vol. 32, no. 6, pp. 861–869, June 1997.

[10] J. Kao and A. Chandrakasan, "MTCMOS sequential circuits," in *Proc. ESSCIRC*, Sept. 2001, pp. 317–320.

[11] V. Zyuban and S. V. Kosonocky, "Low power integrated scan-retention mechanism," in *Proc. ISLPED*, Aug. 2002, pp. 98–102.

[12] Y. Shin et al., "Supply switching with ground collapse: simultaneous control of subthreshold and gate leakage current in nanometer-scale CMOS circuits," *IEEE TVLSI*, vol. 15, no. 7, pp. 758–766, July 2007.

[13] T. Ahn et al., "Incremental analysis and elaboration of VHDL description," in *Proc. APCHDL*, Jan. 1996, pp. 128–131.

[14] J. Jeon et al., "High-level synthesis under multi-cycle interconnect delay," in *Proc. ASP-DAC*, Jan. 2001, pp. 662–667.

[15] S. Mutoh et al., "Design method of MTCMOS power switch for low-voltage high-speed LSIs," in *Proc. ASP-DAC*, Jan. 1999, pp. 113–116.

[16] Synopsys, "Astro user guide," June 2006.

[17] Synopsys, "Astro-rail user guide," June 2006.

[18] T. C. Hu, "Parallel sequencing and assembly line problems," *Operations Research*, vol. 9, no. 6, pp. 841–848, Dec. 1961.

[19] C.-T. Hwang, J.-H. Lee, and Y.-C. Hsu, "A formal approach to the scheduling problem in high level synthesis," *IEEE TCAD*, vol. 10, no. 4, pp. 464–475, Apr. 1991.

[20] G. De Micheli, *Synthesis and Optimization of Digital Circuits*, McGraw-Hill, Inc., 1994.

[21] Daniel Brelaz, "New methods to color the vertices of a graph," *Communications of the ACM*, vol. 22, no. 4, pp. 251–256, Apr. 1979.

[22] A. Hashimoto and J. Stevens, "Wire routing by optimizing channel assignment within large apertures," in *Proc. Design Automation Workshop*, June 1971, pp. 155–169.

[23] F.J. Kurdahi and A.C. Parker, "REAL: a program for register allocation," in *Proc. DAC*, June 1987, pp. 210–215.

[24] "High level synthesis benchmark," http://bears.ece.ucsb.edu/cad/.

[25] "GNU linear programming kit," http://www.gnu.org/software/glpk/.