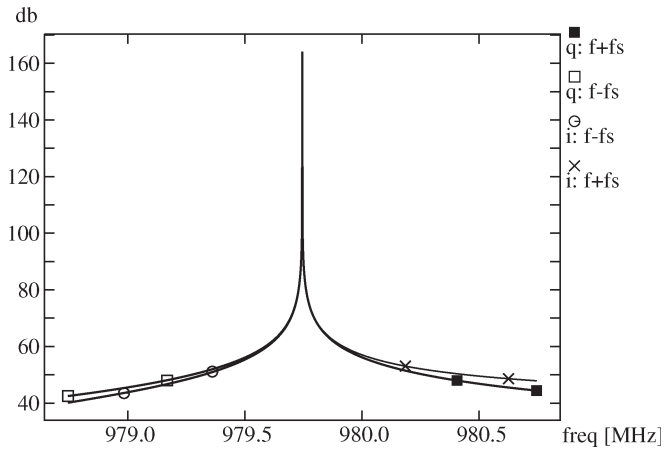

Fig. 4. Spectra of the Qa − Qb (prefix q:) and Ia − Ib (prefix i:) output voltages due to the small-signal perturbation Ao. The $f + f_s$ and $f - f_s$ beats computed by PAN are shown.

closed-form expressions for the determination of the effects, assumed additive, due to a small signal perturbing the orbit of a stable oscillator, or a nonautonomous circuit in the phase plane. Simulation results about an RF oscillator have been shown and compared to time-domain ones in [11].

REFERENCES

[1] A. Brambilla, P. Maffezzoni, and G. Storti-Gajani, "Computation of period sensitivity functions for the simulation of phase noise in oscillators," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 52, no. 4, pp. 681–694, Apr. 2005.

[2] M. Okumura, T. Sugawara, and H. Tanimoto, "An efficient small signal frequency analysis method for nonlinear circuits with two frequency excitations," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 9, no. 3, pp. 225–235, Mar. 1990.

[3] R. Telichevesky, K. Kundert, and J. White, "Efficient AC and noise analysis of two-tone RF circuits," in *Proc. DAC*, Las Vegas, NV, pp. 292–297.

[4] K. S. Kundert, J. K. White, and A. Sangiovanni-Vincentelli, *Steady-State Methods for Simulating Analog and Microwave Circuits*. Norwell, MA: Kluwer.

[5] A. Demir and J. Roychowdhury, "Phase noise in oscillators: A unified theory and numerical methods for characterization," *IEEE Trans. Circuits Syst. I, Fundam. Theory Appl.*, vol. 47, no. 5, pp. 655–674, May 2000.

[6] M. Farkas, *Periodic Motions*. New York: Springer-Verlag, 1994.

[7] W. J. Rugh, *Linear System Theory*. Englewood Cliffs, NJ: Prentice–Hall, 1996.

[8] J. Collantes, I. Lizarraga, A. Anakabe, and J. Jugo, "Stability verification of microwave circuits through Floquet multiplier analysis," in *Proc. IEEE Asia-Pacific Conf. Circuits Syst.*, Dec. 6–9, 2004, pp. 997–1000.

[9] F. Bonani and M. Gilli, "Analysis of stability and bifurcation of limit cycles in Chua's circuit through the harmonic-balance approach," *IEEE Trans. Circuits Syst. I, Fundam. Theory Appl.*, vol. 46, no. 8, pp. 881–890, Aug. 1999.

[10] S. Sancho, A. Suarez, and F. Ramirez, "Phase and amplitude noise analysis in microwave oscillators using nodal harmonic balance," *IEEE Trans. Microw. Theory Tech.*, vol. 55, no. 7, pp. 1568–1583, Jul. 2007.

[11] A. Brambilla and G. Storti-Gajani, "Computation of all the Floquet eigenfunctions in autonomous circuits," *Int. J. Circuit Theory Appl.*, vol. 36, no. 5/6, pp. 717–737, Jul. 2008.

[12] A. Dec, L. Toth, and K. Sunyma, "Noise analysis of a class of oscillators," *IEEE Trans. Circuits Syst. II, Analog Digit. Signal Process.*, vol. 45, no. 6, pp. 757–760, Jun. 1998.

[13] C. Yao and A. N. Willson, "Energy-circulant quadrature LC-VCO," in *Proc. ISCAS*, Kos, Greece, 2006, pp. 4006–4009.

HLS-pg: High-Level Synthesis of Power-Gated Circuits

Eunjoo Choi, Changsik Shin, and Youngsoo Shin

***Abstract*—A problem inherent in power-gated circuits is the overhead of state-retention storage required to preserve the circuit state in standby mode. HLS-pg is a new design framework that takes power gating into account, from scheduling, allocation, and controller synthesis to the final circuit layout. Its main feature is a new scheduler that minimizes the number of retention registers required at the power-gating control step. In experiments on benchmark designs implemented in 0.9-V 65-nm technology, HLS-pg reduced leakage current by 20.7% on average, with 5.0% less area and 4.1% less wirelength, compared to the power-gated circuits produced by conventional high-level synthesis.**

***Index Terms*—High-level synthesis, leakage, power gating.**

I. INTRODUCTION

Leakage current has been continuously growing to the point where it is now comparable to switching power. Leakage current comes from many sources, but subthreshold leakage is the most significant in contemporary technologies. The power-gating scheme [1] is the most popular circuit technique to suppress subthreshold leakage. It reduces the subthreshold leakage when a circuit is in standby mode by cutting the circuit off from its power supply by means of a current switch. When the footer, located between a logic block and $V_{\rm ss}$, is turned off, $V_{\rm ssv}$ rises slowly toward $V_{\rm dd}$; this damages the current states in the storage elements, so that alternative storage elements, which are capable of state retention and called *retention storage*, must be introduced.

There are several variants on the implementation of 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 [2]. This is the main reason for the increase in area of power-gated sequential circuits, which has been observed to be in the range of 13% to 28% [2]. In addition, the total wirelength of power-gated circuits typically increases by 29% to 60% [2] due to the extra wires for control signals needed for the retention flip-flops and resulting increase of wiring congestion. 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.

The problem of minimizing the size of retention storage can only be tackled when the architecture of a circuit is being determined. We will go on to address this new problem of high-level synthesis of power-gated circuits, with the objective of minimizing the number of retention registers. Our main contributions are as follows.

1) A complete framework for power gating, called HLS-pg, that covers scheduling, allocation, controller synthesis, timing closure, and placement and routing.
2) An optimal solution of the scheduling problem for power-gated circuits, based on integer linear programming (ILP), which minimizes the number of retention registers while satisfying resource and latency constraints.

Manuscript received February 11, 2008; revised July 11, 2008. Current version published February 19, 2009. This work was supported in part by Samsung Electronics and in part by Brain Korea 21 Project, the School of Information Technology, KAIST, in 2007. This paper was recommended by Associate Editor R. Camposano.

The authors are with the Department of Electrical Engineering, KAIST, Daejeon 305-701, Korea (e-mail: youngsoo@ee.kaist.ac.kr).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCAD.2009.2013283

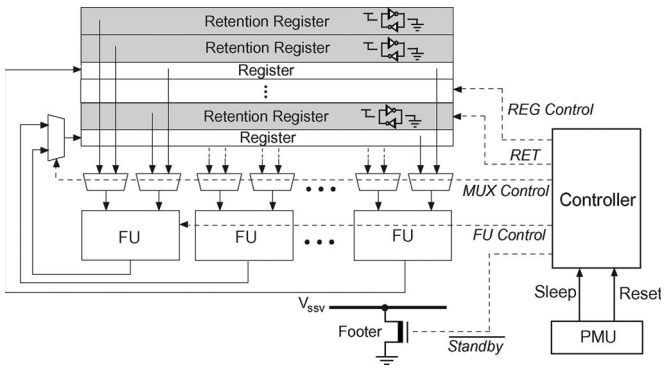

Fig. 1. Target architecture based on a register file.

II. Preliminaries and Design Framework

A. Architecture and Problem Definition

Our target architecture, shown in Fig. 1, consists of a data path that has functional units, registers, and their connections, a controller, and a power management unit (PMU). The registers are classified into normal registers and retention registers. The controller receives 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 deasserted `sleep`, the controller wakes up the data path by deasserting `standby` and `ret`.

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

1) Power gating is applied to the entire data path; the controller is not power gated, since it is responsible for changing the state of the data path between active and standby.
2) The time between the detection of `sleep` and effective power gating should not be greater than the latency of the design, L. This means only one of the control steps in the latency involves the actual power gating, and we call this the *power-gating control step* $C_{\rm pg}$.

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.

Problem 1: Given an unscheduled DFG G with latency (L) and resource (functional unit) constraints, and a power-gating control step $C_{\rm pg}$, the HLS problem for power gating 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 $|S(C_{\rm pg})|$ while satisfying the latency and resource constraints.

Note that Problem 1 can be generalized to make $C_{\rm pg}$ a parameter to be determined rather than a designer-specified parameter. Consequently, the generalized problem is to generate a data path that minimizes $\kappa = min\{|S(1)|, |S(2)|, \ldots, |S(L)|\}$. Clearly, the generalized problem can be solved by solving Problem 1 L times.

B. Design Framework

The overall design flow based on HLS-pg is shown in Fig. 2. A behavioral description written in very high speed integrated circuit hardware description language is analyzed and transformed into a DFG [3], which will be an input to the HLS system. The RTL description generated by the HLS system goes through a standard logic synthesis to create an initial gate-level netlist. The parts of that netlist that correspond to the data path and the controller follow different paths

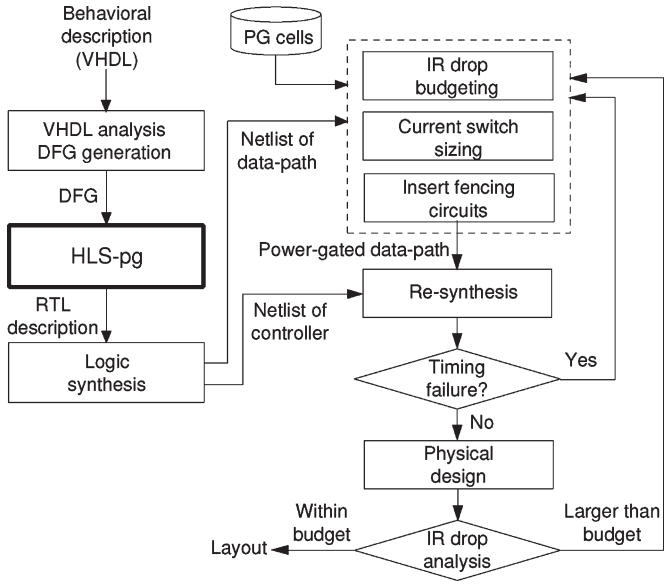

Fig. 2. Overall design flow based on HLS-pg.

in the design flow, as shown in Fig. 2, since the data path is power gated while the controller is not.

From the data-path component of the netlist, we size the current switches (i.e., footers or headers), 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, 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, which in turn determines the number of switch cells required. Fencing circuits are also required to stop the primary output from the data path floating when it is power gated.

The whole netlist is then resynthesized with $V_{\rm dd}$ set to the voltage swing that each gate will experience. In the data path, this is the original $V_{\rm dd}$ minus the chosen voltage drop across a current switch, and in the controller, it is the original $V_{\rm dd}$. If the timing constraints are not satisfied by this resynthesis, 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, we first partition the placement region into two parts, one for the controller and the other for the data path, which will require different power rails: The controller needs $V_{\rm dd}$ and $V_{\rm ss}$, and the data path needs $V_{\rm dd}$ and $V_{\rm ssv}$. The current switches 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 of the whole netlist. The voltage drop across the current switches is then checked 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.

III. Data Path Synthesis for Power Gating

A. Operation Scheduling for Power Gating

We formulate the scheduling component of Problem 1 as a 0–1 linear programming (ILP) problem. If the DFG is large, we first use a conventional scheduling algorithm and then partition the scheduled DFG into components of reasonable size. Our algorithm is then applied to the component of the DFG that contains $C_{\rm pg}$, in order to reschedule the operations to minimize the number of retention registers.

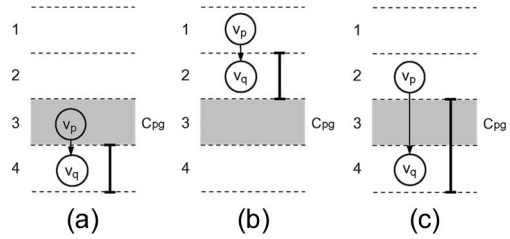

Fig. 3. Linear expressions of variables $x_{i,j}$ represent the intersection of the lifetime of a data value (denoted as intervals on the right of DFGs) with $C_{\rm pg}$: (a) lifetime starts after $C_{\rm pg}$, (b) lifetime ends before $C_{\rm pg}$, and (c) lifetime crosses $C_{\rm pg}$.

1) Basic ILP Formulation: We wish to find a schedule of operations such that the number of variables whose lifetimes include control step $C_{\rm pg}$ is minimized under the latency and resource constraints, while assuming that each variable is consumed by only one operation.

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:

1) A_k: the number of available functional units of type k;
2) $f(v_i)$: the type of functional unit on which v_i can be performed;
3) d_i: a number of control steps required for executing v_i;
4) t_i^S: the earliest control step at which operation v_i can be scheduled, obtained by ASAP scheduling;
5) t_i^L: the latest control step at which operation v_i can be scheduled, obtained by ALAP scheduling with the latency bound L;
6) $x_{i,j}$: a Boolean variable that indicates the beginning of lifetime of 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 [4]. 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 one.

Objective function: Under Assumption 2) from Section II-A, that power gating takes place at only one control step $C_{\rm pg}$, we only preserve those values that are created before $C_{\rm pg}$ and used at or after $C_{\rm pg}$. 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 $C_{\rm pg}$.

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

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

$$\text{Minimize} \sum_{\forall (v_p, v_q) \in E} \left\{ \sum_{i>C_{\rm pg}} x_{q,i} - \sum_{i>C_{\rm pg}} x_{p,i} \right\}. \qquad (1)$$

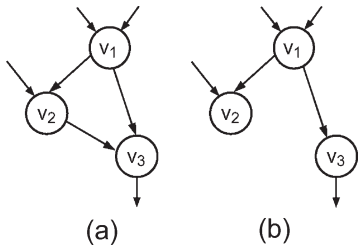

Fig. 4. Operation v_1 with multiple consumers. (a) With serialized dependence. (b) Without serialized dependence.

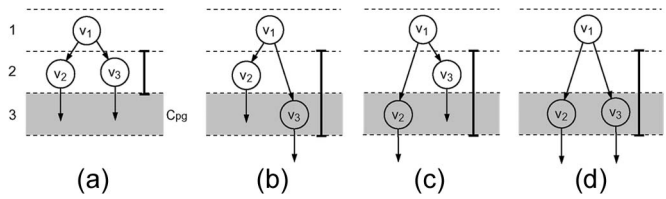

Fig. 5. Example of a multiple fan-out operation.

Note that the inner sums are taken up to $t_q^L + d_q$ and $t_p^L + d_p$, respectively, since further control steps cannot be included in the lifetimes. Note also that the edge (v_p, v_q) is excluded from the outer sum if $t_q^L < C_{\rm pg}$ or $t_p^S \geq C_{\rm pg}$.

The constraints can now be expressed as follows:

$$\sum_{j=t_i^S+d_i}^{t_i^L+d_i} x_{i,j} = 1 \qquad \forall v_i \in V \qquad (2)$$

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

$$\sum_{i:f(v_i)=k} \sum_{j=l+1}^{l+d_i} x_{i,j} \leq A_k, \qquad l = 1, 2, \ldots, L, k = 1, 2, \ldots \qquad (4)$$

$$\sum_i i \cdot x_{p,i} + d_q \leq \sum_j j \cdot x_{q,j} \qquad \forall (v_p, v_q) \in E. \qquad (5)$$

Constraint (2) 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 (3), the data dependence constraint is (5), and the resource constraint (4) 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 (4) is evaluated for each resource type (k) at each control step (l).

2) ILP Formulation Supporting Multiple Fan-Out 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 one. If a data value is consumed by more than one operation, and those operations have mutual dependencies [see v_2 and v_3 in Fig. 4(a)], its lifetime is still determined by just two operations: the producer and the consumer at the bottom of the dependence chain.

However, if the multiple operations that consume a data value are independent, as shown in Fig. 4(b), 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 Fig. 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 Fig. 5(b),

but in Fig. 5(c), it is determined by (v_1, v_2). In the schedules of Fig. 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 one). In the example of Fig. 5, the group variable $y_{1,j}$ can be defined as follows:

$$y_{1,j} = \begin{cases} x_{2,j} & \text{if } \sum_j j \cdot x_{2,j} \geq \sum_j j \cdot x_{3,j} \\ x_{3,j} & \text{otherwise.} \end{cases} \tag{6}$$

These group variables allow us to continue to use the ILP formulation in the previous section with only slight modification. In Objective (1), we replace $x_{q,i}$ with the group variable $y_{p,i}$, for a data value with multiple consumers. In Fig. 5, for example, we use $\Sigma_{i>C_{\rm pg}} y_{1,i} - \Sigma_{i>C_{\rm pg}} x_{1,i}$ 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 (2) to (5)]. These new constraints are the following: $y_{1,3} + y_{1,4} = 1$, $3x_{2,3} + 4x_{2,4} \leq 3y_{1,3} + 4y_{1,4}$, $3x_{3,3} + 4x_{3,4} \leq 3y_{1,3} + 4y_{1,4}$, $x_{2,3} + x_{2,4} + x_{3,3} + x_{3,4} \geq y_{1,3} + y_{1,4}$, and $x_{2,3} + x_{3,4} \geq y_{1,4}$. The first constraint ensures that the start time of the group variable is unique, and therefore corresponds to the original Constraint (2), 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 (6), 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 $x_{3,j}$.

B. Extending the Basic ILP Formulation

1) Operation Chaining: To handle chaining during scheduling, we use a new precedence relation between two operations, $v_i \Rightarrow v_j$ [4], where v_j is the nearest successor of v_i, such that the sum of the execution times (i.e., the propagation delay) from v_i to v_j along a dependence chain is greater than one clock cycle. Using this precedence relation for chaining, the data dependency constraint (5) is modified to become:

$$\sum_i i \cdot x_{p,i} + k \cdot d_q \leq \sum_j j \cdot x_{q,j} \tag{7}$$

where $k = 1$ for all $v_p \Rightarrow v_q$, and $k = 0$ for $\forall (v_p, v_q) \in E$. This implies that, if $v_p \Rightarrow v_q$, the operation v_p has to be scheduled in the control step before that at which v_q is scheduled; otherwise, the control steps of v_p and v_q would be the same (i.e., they would be chained) or v_p would be scheduled before v_q. All other constraints and the objective of the ILP formulation remain the same.

2) Multicycle Operations: Nonpipelined multicycle operation is already allowed by our ILP formulation by means of the variables d_i, which indicate the number of control steps required to execute v_i.

3) Structural Pipelining: To handle pipelined multicycle operations, we have to modify the original DFG. Let the data introduction interval $\delta_i (< L)$ be the number of control steps between two successive introductions of new input data into the functional unit. Then, the functional unit requires extra latches at every δ_i control steps to hold the intermediate results of computations. We now divide each pipelined multicycle operation into several consecutive suboperations, with a different resource type for each suboperation but the same delay δ_i. For example, the multicycle operation v_1 with $\delta_1 = 2$ is divided into v_2 and v_3, and each of them is allocated to different resources, i.e., respectively, to the resources responsible for the first and second half of the execution. An intermediate variable, denoted by a', is generated by v_2.

With this modification of the DFG, the objective and all the constraints remain the same, except that we need an extra constraint to ensure that the start time of v_3 is equal to the start time of v_2 plus δ_1: $\sum_i i \cdot x_{3,i} = \sum_j j \cdot x_{2,j} + \delta_1$. The extra latch to hold the intermediate data a' has to be of the retention type if we are to allow $C_{\rm pg}$ to occur between pipelined functional units.

4) Functional Pipelining: In a pipelined data path, a new DFG is evaluated at every δ control step, where δ is the data introduction interval of data path. Since this is equivalent to having the same DFG repeated at every δ control step, the total number of retention registers is the sum of the retention registers required for control steps $C_{\rm pg} - p\delta$, where p is a nonnegative integer. Thus, our new objective for the ILP formulation is to minimize

$$\sum_{k=C_{\rm pg}-p\delta} \sum_{\forall (v_p, v_q) \in E} \left\{ \sum_{i>k} x_{q,i} - \sum_{i>k} x_{p,i} \right\} \tag{8}$$

where p is nonnegative integer. The operations at every δ control steps cannot share the same resource. Therefore, the resource constraint also has to be modified to become:

$$\sum_{i:f(v_i)=k} \sum_{p=0}^{\lfloor \frac{L-j}{p} \rfloor} \sum_{j=l+p\cdot\delta+1}^{l+p\cdot\delta+d_i} x_{i,j} \leq A_k \tag{9}$$

where k is positive integer and $l = 1, 2, \ldots, L$.

C. Allocation and Control Synthesis

The allocation of registers, functional units, and connection is formulated as vertex coloring of a corresponding conflict graph; we use the heuristic vertex coloring algorithm for this allocation phase. The FSM controller is synthesized as a hard-wired sequential circuit; thus, it is described as a state transition graph. Allocation and control synthesis generates the data path and controller as a Verilog hardware description language. The remaining logic and physical synthesis shown in Fig. 2 are then performed.

IV. Experimental Results

We carried out experiments on a set of behavioral benchmark designs to assess the effectiveness of HLS-pg. 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 to solve the ILP formulation produced by HLS-pg.

We compared results from HLS-pg and conventional list scheduling under the same resource and latency constraints, which are summarized in Table I. The second column shows the resource constraint, expressed as the number of multipliers and ALUs. The third column is the latency constraint, and the fourth column is the control step $C_{\rm pg}$ at which power gating takes place. The results produced by conventional HLS using a list scheduler are shown in the next four columns, and the results from HLS-pg are shown in the following four columns. The next three columns show the reduction in leakage current, area, and wirelength achieved by HLS-pg over conventional HLS. The area is the sum of the areas of all the cells in the design, and the wirelength is obtained after detailed routing. The total leakage values given in the Table I are the sum of the leakage from the retention registers and the fencing circuits.

To summarize Table I, the total leakage current is reduced by a maximum of 46.9%, and by 19.8% on average. Since fencing circuit leakage is 65% less than that of the retention flip-flop, and the number of fencing circuits in both designs is the same, the number of retention

TABLE I
COMPARISON OF RESULTS PRODUCED BY A CONVENTIONAL LIST SCHEDULER AND BY OUR HLS-pg

Benchmark	Res. (*, +)	L	C_{pg}	HLS with list scheduling				HLS-pg				Savings			ILP run -time (sec)
				# Ret. / # Total registers	Leak-age (nA)	Area (μm^2)	Wire-length (mm)	# Ret. / # Total registers	Leak-age (nA)	Area (μm^2)	Wire-length (mm)	Leak-age (%)	Area (%)	Wire-length (%)	
IIR7	(2,2)	14	8	21 / 21	19.6	26911	254	18 / 21	16.9	25681	234	13.8	4.6	7.8	35.80
	(3,2)	14	7	22 / 23	20.5	31867	305	18 / 20	16.9	30005	284	17.6	5.8	6.7	15.73
FIR11	(2,1)	11	7	12 / 16	11.2	19984	170	11 / 16	10.3	19075	169	8.1	4.5	0.8	0.02
	(3,1)	11	5	14 / 16	13.0	24063	197	13 / 16	12.1	23374	183	7.0	2.9	7.3	0.04
ELLIPTIC	(1,3)	16	7	16 / 17	17.0	23601	242	15 / 17	16.1	22924	231	5.3	2.9	4.4	0.14
	(2,2)	16	14	19 / 19	19.7	27412	283	17 / 18	17.9	26981	285	9.1	1.6	-0.9	0.55
LATTICE	(1,1)	12	8	14 / 14	14.3	17074	150	10 / 14	10.6	16074	144	25.3	5.9	4.1	0.95
	(2,2)	9	7	14 / 14	14.3	21363	199	12 / 12	12.4	20548	193	12.7	3.8	3.2	0.02
VOLTERRA	(3,1)	13	4	19 / 19	18.8	28575	264	14 / 19	14.3	27589	263	24.1	3.5	0.5	2.03
	(4,1)	13	3	20 / 20	19.7	34054	301	14 / 19	14.3	31345	280	27.5	8.0	6.9	7.02
WAVELET	(2,2)	16	15	29 / 29	31.0	39704	508	26 / 28	28.3	38455	474	8.7	3.1	6.7	0.90
	(3,2)	15	10	36 / 36	37.3	50502	553	27 / 31	29.2	44902	531	21.7	11.1	3.9	0.34
AR	(2,2)	12	10	10 / 16	10.3	23315	234	5 / 16	5.8	23374	232	43.8	-0.3	0.9	9.33
	(3,2)	11	9	9 / 15	9.4	27974	298	7 / 17	7.6	27146	272	19.2	3.0	8.8	8.38
FIR7	(1,1)	10	7	14 / 16	12.7	18278	173	12 / 14	10.9	17713	172	14.2	3.1	0.9	0.62
	(2,1)	9	5	16 / 17	14.5	23498	238	12 / 15	10.9	21560	227	24.8	8.2	4.6	0.16
DIFFEQ	(1,1)	10	6	11 / 11	11.5	13918	119	5 / 8	6.1	12159	112	46.9	12.6	5.6	2.70
	(2,1)	8	4	10 / 10	10.6	17352	160	5 / 9	6.1	16289	159	42.4	6.1	0.8	0.37
Average												**20.7**	**5.0**	**4.1**	

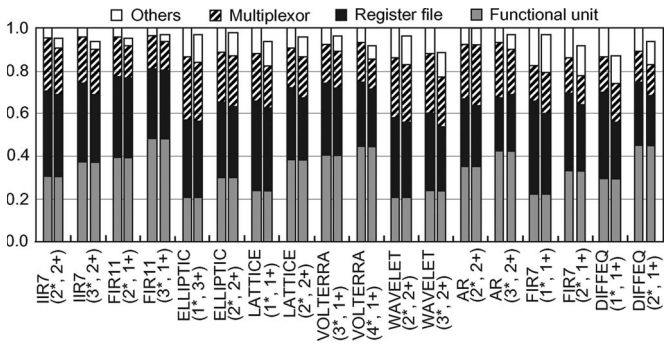

Fig. 6. Area of circuits produced by HLS-pg normalized to the result for conventional list scheduling.

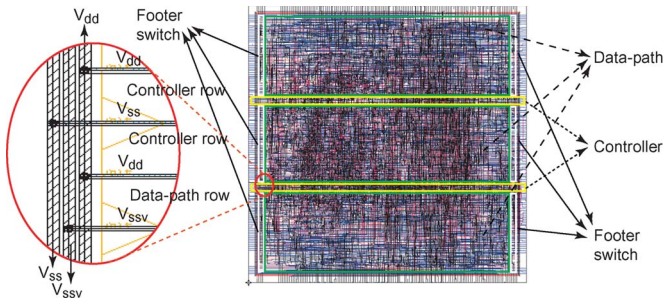

Fig. 7. Layout produced by HLS-pg for design FIR11.

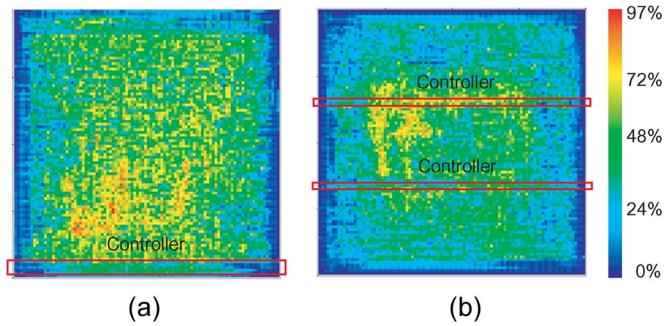

Fig. 8. Congestion maps of FIR11 produced by HLS-pg with the controller (outlined in red). (a) Placed as a single segment. (b) Divided into two segments and placed between data-path segments.

registers (columns 5 and 9 of Table I) is the most influential factor in reducing the leakage current in power-gated circuits.

Fig. 6 shows the relative areas of the circuits produced by HLS-pg (right bar) and list scheduling (left bar). HLS-pg achieved area reduction of 4.7% on average, and reduced the wirelength by 3.7% on average (columns 14 and 15 of Table I). This saving comes directly from a reduced number of retention registers, and, in some examples, from a reduction in the total number of registers and multiplexers (e.g., WAVELET in Table I). The reduction in area is much less significant than the saving in leakage because the number of functional units, which take a significant proportion of the total area, is unchanged, and the number of multiplexers, which we do not try to minimize, may actually increase.

The last column of Table I shows the time required to solve the ILP formulation on a 1.28-GHz SPARC processor with 2 GB of memory; only four examples take more than 10 s.

Fig. 7 shows the layout of design FIR11 produced by HLS-pg following the design flow shown in Fig. 2. The current switches are located evenly on the left- and right-hand side of the placement region to minimize the current path through $V_{\rm ssv}$ and $V_{\rm ss}$. The FSM controller, which is not power gated, has $V_{\rm dd}$ and $V_{\rm ss}$ rails; and the data path, which is power gated, has $V_{\rm dd}$ and $V_{\rm ssv}$ rails, as shown in the left of the layout in Fig. 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 decision can be understood from the wiring congestion maps shown in Fig. 8. Without partitioning [Fig. 8(a)], the data-path cells that communicate with the controller tend to be placed close nearby, leading to congested routing in the lower part of the placement region. When the controller is split [Fig. 8(b)], data-path cells can be placed much more uniformly over the placement region. Layouts of all the other examples were obtained similarly by splitting the placement region of the controller.

V. 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. Experiments on benchmark designs showed that HLS-pg can reduce leakage current by 20.7% on average, while cutting area by 5.0% and wirelength by 4.1% over a conventional high-level synthesis which is not specialized for power gating.

Extending HLS-pg to minimize the total number of registers and multiplexers as well as retention registers, and employing a heuristic rather than ILP-based formulation are left as a future work.

ACKNOWLEDGMENT

The authors would like to thank Prof. T. Kim of Seoul National University, Seoul, Korea, for help with the ILP formulation.

REFERENCES

[1] S. Mutoh, T. Douseki, Y. Matsuya, T. Aoki, S. Shigematsu, and J. Yamada, "A 1-V power supply high-speed digital circuit technology with multithreshold-voltage CMOS," *IEEE J. Solid-State Circuits*, vol. 30, no. 8, pp. 847–854, Aug. 1995.

[2] H.-O. Kim and Y. Shin, "Semicustom design methodology of power gated circuits for low leakage applications," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 54, no. 6, pp. 512–516, Jun. 2007.

[3] J. Jeon, D. Kim, D. Shin, and K. Choi, "High-level synthesis under multi-cycle interconnect delay," in *Proc. ASP-DAC*, Jan. 2001, pp. 662–667.

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

Spare Cells With Constant Insertion for Engineering Change

Yu-Min Kuo, Ya-Ting Chang, Shih-Chieh Chang, and Malgorzata Marek-Sadowska

***Abstract*—Engineering change (EC) is the process of modifying a VLSI design implementation to eliminate design errors, to add new specifications, or to correct design constraint violations. Usually, an EC problem is resolved by using spare cells that have been inserted into unused spaces on a chip. In this paper, we describe an iterative method to determine feasible mapping solutions for an EC problem considering spare cells whose inputs can be connected to Vdd or Gnd. Setting some of the cell inputs to fixed values is referred to as *constant insertion*. Constant insertion can increase cells' functional flexibility. Our experimental results suggest that constant insertion reduces the area required to find a feasible mapping solution to 80% of that with no constant insertion for the selected EC equations. We also show a procedure for modifying the initial feasible EC solution such that the routing or timing improves.**

***Index Terms*—Boolean function, constant insertion, engineering change (EC), logic synthesis.**

Manuscript received February 14, 2008; revised August 31, 2008. Current version published February 19, 2009. This work was supported in part by the National Science Council under Grants NSC 97-2220-E-007-041 and NSC 96-2220-E-007-023. This paper was recommended by Associate Editor I. Bahar.

Y.-M. Kuo and S.-C. Chang are with the Department of Computer Science, National Tsing Hua University, Hsinchu 30013, Taiwan (e-mail: ymkuo@cs.nthu.edu.tw; scchang@cs.nthu.edu.tw).

Y.-T. Chang is with MediaTek Inc., Hsinchu 30078, Taiwan.

M. Marek-Sadowska is with the Department of Electrical and Computer Engineering, University of California, Santa Barbara, CA 93106 USA (e-mail: mms@ece.ucsb.edu).

Digital Object Identifier 10.1109/TCAD.2009.2013537

I. INTRODUCTION

Often, in the late design stages, a design implementation must be modified to meet new specifications or to satisfy new constraints. To meet a tight time-to-market schedule and save design costs, it is necessary to make corrections by introducing small modifications to the original design implementation instead of redesigning the circuit from scratch. How to determine and implement the desired changes is the problem referred to as engineering change (EC) [6]–[8], [10]–[13].

Traditionally, an EC problem is resolved or alleviated by using *spare cells*. Spare (redundant) cells may be inserted in unused spaces. When an EC is needed, spare cells are utilized to resolve it. First, a Boolean equation, called the *EC equation*, captures any modifications to the original design. Several papers [1], [5], [7], [9]–[11] describe how to automate the process of obtaining minimal change EC equations. Then, technology mapping attempts to realize the EC equations using available spare cells. Finally, the physical spare cells are selected taking into account routing and timing.

For example, in Fig. 1, suppose the EC equation is $out = (a * b)' + (c + d)'$ and the available spare cells are as listed in the table in Fig. 1(a). One mapping solution obtained with these four types of spare cells is shown in Fig. 1(b). It requires three INV gates and three NAND2 gates. We observe that this solution requires gates fewer than the available spare cells. In this case, the mapping solution can be constructed with the available spare cells.

Because spare cells are inserted into the layout when unused spaces are known, their types and quantities are limited. For this reason, mapping solutions may not be always realizable. We refer to the requirement of building the solution from only the available spare cells as the *quantity constraint*. An example EC equation and the available spare cells are shown in Fig. 1(a). Two mapping solutions are shown in Fig. 1(b) and (c). The solution in Fig. 1(b) satisfies the quantity constraint. The solution in Fig. 1(c) does not satisfy the quantity constraint because it requires two NOR2 gates when only one NOR2 cell is available.

Traditional technology mapping is based on dynamic programming. This algorithm cannot be applied directly to a mapping problem with a quantity constraint because dynamic programming can only be applied when the optimal solutions of subproblems can be used to find the optimal solution of the overall problem. If the subcircuits satisfy the quantity constraint, this does not imply that the whole circuit still satisfies the quantity constraint. In addition, dynamic programming cannot iteratively find another mapping solution.

Most industrial designers insert enough spare cells for resolving EC problems and scatter them throughout a chip. It is likely that spare cells with specific functionalities are far away from the site where an EC is needed. To obtain the required function types, a mapping solution may need to use those far-away spare cells. This may introduce timing and routing violations. We aim to find a mapping solution that satisfies the quantity constraint in the region neighboring the site that requires an EC.

A spare cell with the needed functionality can often be created from another cell by connecting some of its inputs to Vdd or Gnd. We refer to this process as *constant insertion*. For example, in Fig. 2, an AOI21 cell can become an INV, NAND2, or NOR2 cell when constants are inserted to some inputs. Our experimental results show that utilizing spare cells with constant insertion can significantly increase the flexibility of a mapping solution. For example, the mapping solution in Fig. 1(c) requires one INV, one NAND2, and two NOR2 gates. Although there is only one NOR2 cell available, an AOI21 cell can be configured into a NOR2 cell by inserting constants. This mapping solution also satisfies the quantity constraint. Checking the quantity constraint